DATA ENGINEERING – LINUS STUHLMANN

DATA WRANGLING UND EXPLORATION

Typische Datenqualitätsprobleme können verschiedene Aspekte umfassen:

- Das **Parsen**, das Aufteilen eines Texts in kleinere Bestandteile oder Felder, kann Schwierigkeiten bereiten, insbesondere bei Trennzeichen.
- **Unterschiedliche Benennungskonventionen** können zu Inkonsistenzen führen, z.B. "NYC" vs. "New York".
- **NaN** - Fehlende erforderliche Felder, wie z.B. fehlende Sozialversicherungsnummer.
- **Unterschiedliche Darstellungen** von Daten, z.B. "2" vs. "Two".
- Felder, die zu lang sind und gekürzt werden müssen.
- **Verletzung von Primärschlüsseln**, z.B. wenn zwei Personen dieselbe Sozialversicherungsnummer haben.
- **Redundanz** - Redundante Datensätze, die entweder exakte Duplikate oder ähnliche Datensätze enthalten.
- **Formatierungsprobleme**, insbesondere bei Datumswerten.
- **Lizenz- oder Datenschutzprobleme**, die die Nutzung der Daten einschränken.

Die Kenntnis des Domänenkontexts ist entscheidend, da die gleichen Datenwerte in einem anderen Kontext möglicherweise keine Fehler aufweisen. Die Datenbereinigung kann dazu beitragen, solche Probleme zu beheben und die Datenqualität zu verbessern.

Ein Beispiel für die Bedeutung der Datenbereinigung ist das Loch in der Ozonschicht über der Antarktis. Als Wissenschaftler dieses Phänomen 1976 erstmals entdeckten, glaubten sie zunächst, dass ihre Instrumente fehlerhaft seien, da die Daten als unvernünftig eingestuft und von den Algorithmen zur Datenqualitätskontrolle abgelehnt wurden.

REGEX

Reguläre Ausdrücke (Regex) sind ein wichtiges Werkzeug zur Datenbereinigung. Sie ermöglichen das Festlegen von Musterregeln, um Daten abzugleichen und zu transformieren. Beispielsweise können sie verwendet werden, um alle nicht-zeichenbasierten Zeichen aus einer Spalte zu entfernen oder Text in Kleinbuchstaben umzuwandeln.

Die Python-Bibliothek "re" bietet verschiedene Funktionen zur Arbeit mit regulären Ausdrücken, darunter "re.match()", "re.search()", "re.findall()", "re.finditer()" und "re.sub()". Durch die Verwendung dieser Funktionen können Muster in Texten gefunden, ersetzt oder extrahiert werden. Es ist auch möglich, reguläre Ausdrücke vorab zu kompilieren, um die Leistung zu optimieren und die Wiederverwendbarkeit zu erhöhen.

WICHTIGSTE META-CHARAKTER

- [and] werden verwendet, um eine Zeichenklasse anzugeben.
- [abc] wird jede der Zeichen a, b oder c abgleichen und ist das gleiche wie `[a-c]`.
- . passt zu jedem Zeichen.
- ^ definiert das Komplement.
- Backslash: `\`.
    - Wird entweder von Sonderzeichen gefolgt, um spezielle Sequenzen anzugeben.
    - Zum Beispiel: `\w` passt zu jedem alphanumerischen Zeichen, `\d` passt zu jeder Dezimalziffer, `\s` passt zu Leerzeichen (wenn die Buchstaben großgeschrieben sind, erhalten wir das Komplement).
- *: Das vorherige Zeichen (Klasse) kann null oder mehrmals wiederholt werden.
- +: Das vorherige Zeichen (Klasse) kann einmal oder mehrmals wiederholt werden.
- ?: Das vorherige Zeichen (Klasse) kann null oder einmal wiederholt werden.

WICHTIGSTEN REGEX FUNKTIONEN IN PYTHON

- re.match(pattern, string): Sucht nach einem Muster **am Anfang des Strings** und versucht, die **erste Übereinstimmung** zu finden und zurückzugeben.
- re.search(pattern, string): Sucht nach einem Muster im gesamten String und versucht, die **erste Übereinstimmung** zu finden und zurückzugeben.
- re.findall(pattern, string): Verwendet findall(), um **alle** Übereinstimmungen zu finden und als **Liste** zurückzugeben.
- re.finditer(pattern, string): Eine der leistungsstärksten Funktionen in der "re"-Bibliothek. Gibt einen Iterator von Übereinstimmungsobjekten zurück.
- re.sub(pattern, replacement, string): Sucht nach allen Vorkommen des Musters im gegebenen String und **ersetzt sie** durch einen Ersatzwert.
- re.compile(pattern, flags=0): Kompiliert ein reguläres Ausdrucksmuster als String in ein Regex-Musterobjekt. Das Verhalten des Ausdrucks kann durch die Angabe von Regex-Flag-Werten geändert werden.

FLAG-VALUES

Hier ist eine Liste von Flag-Werten, die in regulären Ausdrücken verwendet werden können:

- re.I (auch `re.IGNORECASE`): Ignoriert die Groß-/Kleinschreibung bei der Übereinstimmung von Zeichen.
- re.M (auch `re.MULTILINE`): Verändert das Verhalten von `^` und `$`, sodass sie auch zu Beginn bzw. Ende jeder Zeile in einem mehrzeiligen Eingabetext passen.
- re.S (auch `re.DOTALL`): Veranlasst den Punkt (`.`), auch Zeilenumbrüche zu erfassen.
- re.X (auch `re.VERBOSE`): Erlaubt es, Leerzeichen und Kommentare in einem regulären Ausdruck zu ignorieren, um ihn lesbarer zu machen.
- re.A (auch `re.ASCII`): Begrenzt die Interpretation von `\w`, `\W`, `\b`, `\B`, `\d`, `\D`, `\s` und `\S` auf ASCII-Zeichen.
- re.U (auch `re.UNICODE`): Erlaubt die Verwendung von Unicode-Zeichenklassen und erweitert die Funktionalität von `\w`, `\W`, `\b`, `\B`, `\d`, `\D`, `\s` und `\S` auf Unicode-Zeichen.
- re.L (auch `re.LOCALE`): Veranlasst die Interpretation von `\w`, `\W`, `\b`, `\B`, `\d`, `\D`, `\s` und `\S` gemäß den Einstellungen der aktuellen lokalen Umgebung.

Diese Flags können einzeln verwendet werden oder durch `|` kombiniert werden, wie im Beispiel:

➔ `re.findall(pattern, string, flags=re.I|re.M|re.X)`

DATENBEREINIGUNG

Hier sind die typischen Schritte zur Datenbereinigung:

1. **Entfernen redundanter Spalten**: Entfernen Sie Spalten, die nicht erforderlich oder redundant sind, mit der Funktion "drop". Geben Sie den Spaltennamen und den Achsenparameter an, um anzugeben, dass Sie eine Spalte entfernen möchten. Verwenden Sie den "inplace"-Parameter, um die Änderungen zu speichern.

```
df.drop("Unnamed: 0", axis=1, inplace=True)
```

2. **Standardisierung der Datumsformate**: Konvertieren Sie Daten in ein standardisiertes Format, um Konsistenz und Kompatibilität für weitere Analysen sicherzustellen. Transformieren Sie die Daten, um sicherzustellen, dass alle Datumsangaben im gleichen Format vorliegen.

```
df["Date"] = df["Date"].astype("datetime64[ns]")
```

3. **Standardisierung von Namen**: Standardisieren Sie die Darstellung von Namen, z.B. durch Verwendung von Groß- oder Kleinschreibung und Festlegung einer Reihenfolge (z.B. Nachname, Vorname). Passen Sie die Namen an eine einheitliche Formatierung an.

```
df["Name"].str.split(",", expand=True)
```

4. **Bereinigung von Zahlendaten**: Bereinigen Sie Zahlendaten, z.B. entfernen Sie Währungssymbole oder Tausendertrennzeichen, um die Daten in ein numerisches Format umzuwandeln und für weitere Analysen geeignet zu machen.

Diese Schritte helfen dabei, die Daten sauber und konsistent zu machen, um sie für Analysen und andere Datenverarbeitungsprozesse besser nutzbar zu machen.

STRUKTURIERTE DATEN

- Speicherung von strukturierten Daten in relationalen Systemen
- Grundlagen des Data Warehousing
- Grundlagen des ETL-Prozesses

SPEICHERUNG VON STRUKTURIERTEN DATEN IN RELATIONALEN SYSTEMEN

FLAT FILES

- Sind einfache Datensätze in Zeilenform, ohne hierarchische Strukturen oder komplexe Verknüpfungen zwischen den Datensätzen
- → bspw. CSV-Files

RELATIONALE SYSTEME

- In Schemas organisiertes Datenmodell, das die Möglichkeit bietet, verschiedene Tabellen und Datensätze über Schlüssel miteinander zu verknüpfen.
- Mehrere User können darauf zugreifen und mit der gleichen Datenbank arbeiten
    - Transaktionen
- Eignet sich für einfache statistische Analysen und Reports
- Sehr effizient
- Nachteile von relationalen Modellen
    - Sehr beschränkte Möglichkeiten für semi-, und unstrukturierte Daten
    - Nur für maximal mittelgrosse Datensätze

DATA WAREHOUSE

Ein Data Warehouse ist eine Sammlung von strukturierten Daten, die für die Managemententscheidungsfindung eingesetzt werden können. Einsatzgebiete eines DWH sind:

- Finanz- und Bankdienstleistungen
- Konsumgüter
- Einzelhandel
- Fabrikation

DWHs können zu verschiedenen Zwecken eingesetzt werden.

- Informationsverarbeitung (Information Processing)
    - Verwaltung von Information
- Analytische Zwecke (Analytical Processing/ Data Mining)
    - Operationen wie Slice-and-Dice, Drill-Down, Drill-Up und Pivoting
    - Analyse und generieren von neuen (verborgenen) Informationen
    - Data Mining umfasst auch den Aufbau analytischer Modelle sowie Klassifikationen und Vorhersagen.

OPERATIONAL DATABASES (OLTP) VS. DATA WAREHOUSES (OLAP)

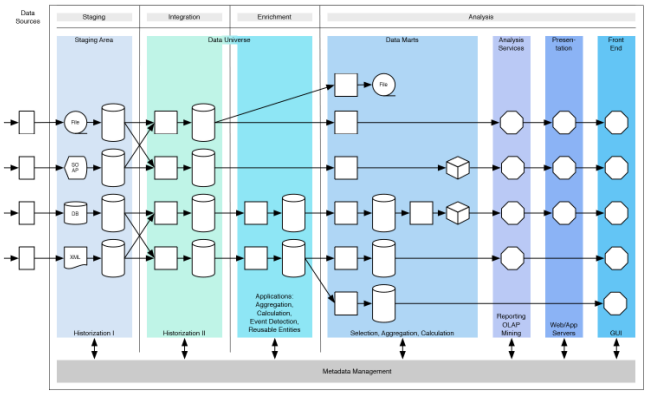
Operational Databases:

- OLTP – Online Transaction Processing
- Fokus liegt auf schneller Verarbeitung von Transaktionsdaten in Echtzeit
- Täglicher Betrieb im Unternehmen (Transaktionen, Kundeninteraktionen…)
- 100mb-100gb
- **Relationales Modell**

Data Warehouses:

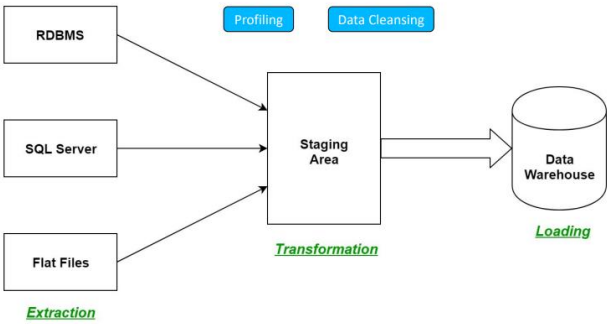
- OLAP – Online Analytical Processing
- Fokus auf Analyse von historischen Daten zur Unterstützung von Entscheidungsfindung.
- Analyse des Business
- 100gb-100tb
- **Star Schema oder Snowflake**

ARCHITEKTUR EINES DATA WAREHOUSE



1. **Data Sources** (Extraction - ETL)
    a. Daten werden aus verschiedenen Quellen extrahiert (ERM, operative DB, Internet…)
2. **Staging Area** (Transform - ETL)
    a. Daten werden zwischengespeichert
    b. Heterogene Daten werden homogenisiert und aufbereitet
3. **Integration layer** (Load - ETL)
    a. Daten werden in einer relationalen Datenbank gespeichert
4. **Enrichment**
    a. Integrierte Daten werden für eine bessere Analyse mit zusätzlichen Informationen, oft aus externen Quellen, erweitert (Geodaten, Demografie…)
5. **Data Marts**
    a. Ist ein Sub-Data-Warehouse, das für ein spezifisches Team oder eine Abteilung ausgerichtet ist. Dadurch sind Abfragen viel effizienter.
6. **Analyse Services**
    a. Daten werden analysiert und Berichte erstellt. Daten Analyse Tools werden verwendet um Muster und Trends zu identifizieren, die für die Geschäftsleitung relevant sind.
7. **Benutzerschnittstellen** (Präsentation/ Frontend)
    a. Ermöglichen den Benutzern auf die Daten zuzugreifen in Form von von bspw. Dashboards, Berichten, Ad-hoc-Abfragen

ETL-PROZESS (EXTRACT, TRANSFORM, LOAD)



Der ETL-Prozess ist eine zentrale Methode im Data-Warehousing, bei dem Daten aus verschiedenen Quellen extrahiert, transformiert und in ein Ziel-Data-Warehouse geladen werden.

EXTRACT

- Daten werden aus vielen unterschiedlichsten Quellen extrahiert und zwischengespeichert und sind aus diesem Grund sehr heterogen.
- Weitere Problematiken können darstellen:
  - Volumen
  - Komplexität der Daten (Schema)
- Validierung von Daten, um sicherzustellen, dass sie den Anforderungen des Data Warehouses entsprechen
- Time Setting
  - Periodisch (fixer Extraktionszeitpunkt),
  - Event-driven (ausgelöst durch Ereignis)
  - Query-driven (spezifische Abfrage)
- Unterschiedliche Methoden für die Extraktion, einschließlich synchroner und asynchroner Benachrichtigung, periodischer Extraktion und abfragebasierter Extraktion
- Unterschiedliche Arten von Datenextraktion, einschließlich Snapshots, Logs und Netlogs

TRANSFORM

- Daten werden für die Analyse und das Data Warehouse-Format verarbeitet
- Transformation beinhaltet die Verarbeitung von Daten, um sicherzustellen, dass sie im Data Warehouse konsistent und für die Analyse geeignet sind
- Transformation kann Probleme aufgrund von struktureller Heterogenität verursachen

Profiling

- Analyse des Inhalts und der Struktur einzelner Attribute (Datentyp, Wertebereich, Verteilung und Varianz, NULL-Werte/Fehlende Werte, Kardinalität, Muster).
- Analyse der Abhängigkeiten zwischen Attributen in einer Beziehung (funktionale Abhängigkeiten, Primärschlüsselkandidaten, implizite Integritätsbedingungen).
- Analyse der Überlappungen zwischen Attributen verschiedener Beziehungen (Redundanzen, Fremdschlüsselbeziehungen).

Data Cleansing

- Erkennung und Beseitigung von Inkonsistenzen, Widersprüchen und Fehlern in den Daten zur Verbesserung der Datenqualität.
- Normalisierung und Standardisierung (Datentypkonvertierung, Mapping auf einheitliches Format, Kategorisierung von numerischen Werten, Domänenspezifische Transformationen).
- Beispiele für Domänenspezifische Transformationen: Umwandlung von Initialen zu vollen Namen, Umwandlung von Abkürzungen zu vollständigen Wörtern (z.B. St. zu Street), Nutzung von Adressdatenbanken für Adressen, Branchenspezifische Produktbezeichnungen.

Missing Values

- Der Umgang mit fehlenden Werten ist ein wichtiges Thema in der Datenanalyse und kann auf verschiedenen Ebenen auftreten. Die Erkennung von fehlenden Werten kann durch einfache Analysen wie das Zählen von NULL-Werten oder das Vergleichen mit erwarteten Werten erfolgen. Die Handhabung von fehlenden Werten kann durch die Schätzung von Werten mithilfe statistischer Methoden wie Mittelwert oder Standardabweichung oder durch die Ausnutzung von Attributbeziehungen wie Regressionsanalyse oder neuronale Netzwerke erfolgen.

LOAD

- Transfer von transformierten Daten in ein permanentes Ziel-Data-Warehouse
- **Arten des Ladens**
    - **Bulk-Uploads** sind eine Methode zum Laden großer Datenmengen in eine Datenbank oder ein Data Warehouse und erfordern spezielle DB-Extensionen für das Laden grosser Datenmengen und können aufgrund der fehlenden Überprüfung von Triggern und Einschränkungen schneller sein.
    - **Record-basiertes Laden** - Bei dieser Methode werden einzelne Datensätze nacheinander in die Datenbank geladen, anstatt große Datenmengen in einem Stapel oder Batch zu übertragen, wie es beim Bulk-Upload der Fall ist.
- **Risiken**
    - Blockierung des gesamten Datenbankmanagementsystems
    - Aktivierung von Triggern
    - Wartung von Indizes
    - Constraints

ETL VS. ELT

ELT (Extract, Load, Transform) ist ein alternativer Ansatz zum ETL-Prozess, bei dem Daten zuerst in das Ziel-Data-Warehouse geladen werden und dann transformiert werden. Im Gegensatz zum ETL-Prozess werden bei ELT die Transformationsprozesse in der Datenbank durchgeführt, was bei der Verarbeitung großer Datenmengen Vorteile in Bezug auf Performance und Skalierbarkeit bietet.

BATCH VS. REAL-TIME PROCESSING

Batch-Processing verarbeitet eingehende Daten periodisch, bspw. einmal täglich. Realtime Processing verarbeitet die Daten, während sie anfallen.

- Batch-Processing eignet sich eher für grosse Datenmengen
- Realtime-Processing eignet sich, wenn schnelle Reaktionszeiten benötigt sind

DATA LAKE VS. DATA WAREHOUSE

DATA LAKE DEFINITION

Ein Data Lake ist eine große, zentrale Speicherumgebung, die strukturierte, unstrukturierte und halbstrukturierte Daten in Rohform aufnimmt. Im Gegensatz zu einem Data Warehouse werden die Daten im Data Lake nicht vorab transformiert, bereinigt oder strukturiert, sondern bleiben in ihrer ursprünglichen Form erhalten.

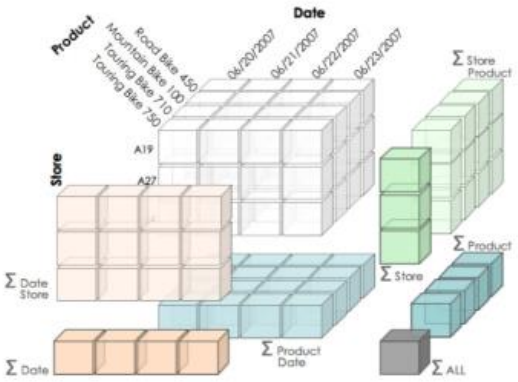
UNTERSCHIEDE ZU DATA WAREHOUSE

- Struktur: Ein Data Warehouse ist stark strukturiert, d.h. die Daten werden in vordefinierten Tabellen und Spalten gespeichert, die in einem relationalen Schema organisiert sind. Ein *Data Lake hingegen ist unstrukturiert* und kann verschiedene Arten von Daten speichern, einschließlich strukturierter, halbstrukturierter und unstrukturierter Daten.
- Datenquellen: Ein Data Warehouse wird typischerweise aus internen Quellen wie ERP-Systemen, CRM-Systemen und anderen Unternehmensanwendungen gespeist. Ein Data Lake hingegen kann aus einer *Vielzahl von Datenquellen* gespeist werden, einschließlich externer Datenquellen wie Social-Media-Feeds und Sensordaten.
- Verarbeitung: Ein Data Warehouse ist in der Regel auf die Analyse von Daten ausgerichtet und wird häufig im Rahmen von Business-Intelligence-Projekten verwendet. Ein Data Lake hingegen ist darauf ausgelegt, *Daten zu speichern, ohne dass im Voraus bekannt ist, wie sie verarbeitet werden sollen*. Eine Verarbeitung kann auch erst später, nach Bedarf stattfinden.
- Analyse: Ein Data Warehouse bietet in der Regel vordefinierte Analyse-Tools, die speziell für Business-Intelligence-Analysen entwickelt wurden. Ein Data Lake hingegen bietet *keine vordefinierten Analysetools*, da es nicht auf spezifische Analysezwecke ausgerichtet ist. Stattdessen müssen Benutzer ihre *eigenen Tools oder spezialisierte Analyse-Tools einsetzen*, um aus den Daten des Data Lakes Erkenntnisse zu gewinnen.

DATA WAREHOUSE MODELLING

Das Datenmodell im Data Warehouse ist auf die Bedürfnisse der Analyse ausgerichtet und dient der Entscheidungsunterstützung. Zentrale Elemente sind Geschäftsleistungsindikatoren wie z.B. Gewinn, Umsatz oder Verlust, welche als Maßeinheiten bezeichnet werden. Diese werden aus verschiedenen Perspektiven untersucht, wie z.B. zeitlich, regional oder produktbezogen, welche als Dimensionen bezeichnet werden. Die Dimensionen können strukturiert werden, z.B. nach Jahr, Quartal oder Monat, mit Hierarchien oder Konsolidierungsebenen. Eine Masseinheit zusammen mit ihren zugehörigen dimensionalen Merkmalen bildet eine Tatsache.

MULTIDIMENSIONALES DATEN MODELL



		Measures							
		• Quantity				• Amount			
Dimensions		Store							
Product	Date	A19	A27	A34	Total Store	A19	A27	A34	Total Store
– Road Bike 450	06/20/2007	2	7	4	13	498	1743	996	3237
	06/21/2007	9	12	10	31	2241	2988	2490	7719
	06/22/2007	3		7	10	747		1743	2490
	06/23/2007	5	1	9	15	1245	249	2241	3735
Total Road Bike 450		19	20	30	69	4731	4980	7470	17181
– Mountain Bike 100	06/20/2007	8	10	3	21	6392	7990	2397	16779
	06/21/2007	5	11	4	20	3995	8789	3196	15980
	06/22/2007	9	7		16	7191	5593		12784
	06/23/2007	6		4	10	4794		3196	7990
Total Mountain Bike 100		28	28	11	67	22372	22372	8789	53533
– Touring Bike 710	06/20/2007	5		9	14	2995		5391	8386
	06/21/2007	7	2	12	21	4193	1198	7188	12579
	06/22/2007	4		13	17	2396		7787	10183
	06/23/2007		2	8	10		1198	4792	5990
Total Touring Bike 710		16	4	42	62	9584	2396	25158	37138
+ Touring Bike 750		19	12	15	46	10621	6708	8385	25714
Total Product		82	64	98	244	47308	83765	49802	133566

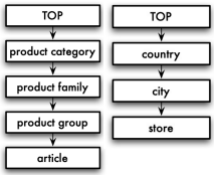
Dimensionen

Die Dimensionen in diesem Fall sind: **Produkt, Datum, Standort.** Diese können verschiedene Hierarchien aufweisen, was so viel heisst, wie granular sie sind.

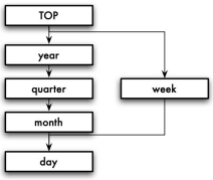
Hierarchien

Die Hierarchien für Standort

- Einfache Hierarchien



- Parallele Hierarchien
    - Dabei können Zwischenschritte übersprungen werden.



DWH-OPERATIONEN EINES MULTIDIMENSIONALEN MODELLS

PIVOTING & ROTATION

Rotieren einer Tabelle, um Daten in einer anderen Perspektive darzustellen.

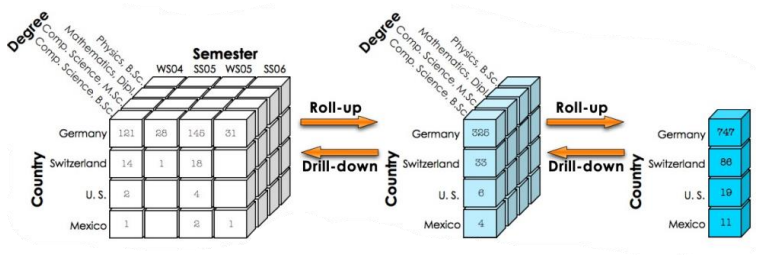
ROLL-UP & DRILL-DOWN

Roll-Up: **Aggregieren** von Daten in einer Hierarchie von Dimensionen, um eine höhere Ebene der Zusammenfassung zu erstellen. Es geht um die

- (Semester → akademisches Jahr → Dekade)
- Akademisches Jahr → über alle Jahre

Drill-Down: Detailliertere Daten werden angezeigt, indem eine Hierarchieebene durchlaufen wird.

- Gegenteil von Roll-Up
- Dekade → Jahr → Semester



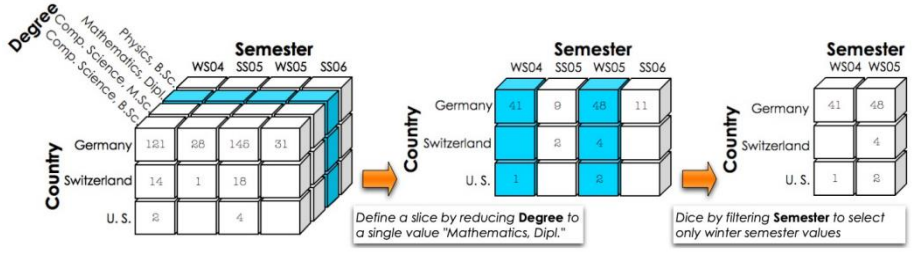
SLICE & DICE

Slice: Auswahl von Daten, die nur eine bestimmte Dimension oder einen bestimmten Wertbereich betreffen.

- Nur Mathematiker über alle Semester

Dice: Selektion von Daten, die auf mehreren Dimensionen basieren.

- Nur die Wintersemester



RANKING (TOP / BOTTOM)

Sortieren der Daten nach einem bestimmten Kriterium und Anzeigen der besten oder schlechtesten Ergebnisse.

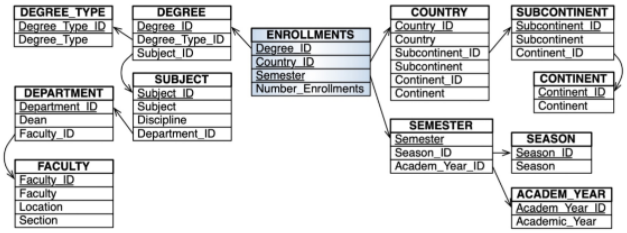
PUSH & PULL

Daten können in OLAP-Systemen entweder von der Datenquelle zum OLAP-Server (Pull) oder vom OLAP-Server zur Datenquelle (Push) übertragen werden.

SNOWFLAKE-SCHEMA & STAR-SCHEMA

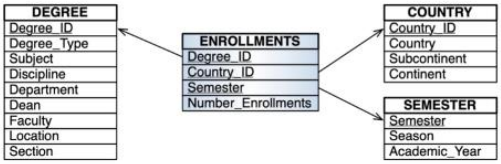
Snowflake-Schema

- Für jedes Klassifikationslevel ein eigener Table.
- Viele Joins notwendig
- Keine Redundanzen



Star-Schema

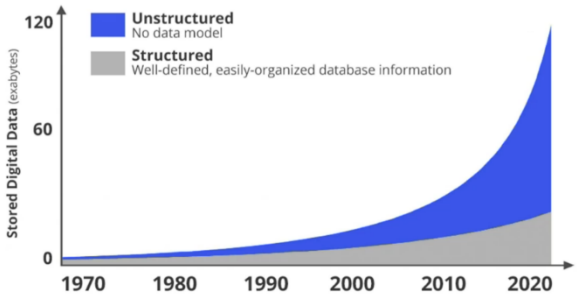
- Nur eine Dimension pro Table
- Einfacherer Struktur
- Einfacher und effiziente Query-Verarbeitung
- Redundanzen



UNSTRUKTURIERTE DATEN

INFORMATION RETRIEVAL

- Suchen von Informationen in unstrukturierten Daten. Such-Maschinen wie Google machen genau das. Ist die Aktivität relevante Informationen für einen Nutzer zu finden in einer Sammlung von vielen Informationen.
- Dafür werden gezielte Anfragen/Queries benötigt. Dies können, Worte, Satzfragmente oder Bilder sein.
- Warum braucht es Information Retrieval?
    - Viele Informationen in Bibliotheken erhältlich
- Es gibt mehr unstrukturierte Daten als strukturierte Daten



TEXTE

Text ist die häufigste Form unstrukturierter Daten, da es eine Vielzahl von Quellen gibt, die unstrukturierten Text enthalten, wie z.B. E-Mails, Nachrichtenartikel, soziale Medien, Webseiten und Dokumente.

UNSTRUKTURIERTE DARSTELLUNG

- Die unstrukturierte Textdarstellung repräsentiert den Text als ungeordnete Menge von Wörtern, auch bekannt als **"Bag-of-Words" (BoW)**.
    - Betrachten wir die zwei Sätze:

      Satz 1: "Das Auto ist blau."
      Satz 2: "Das Auto ist rot."

    - Im BoW-Modell würde jeder dieser Sätze in ein Vokabular aller eindeutigen Wörter aufgeteilt, und dann würde die Häufigkeit jedes Wortes in jedem Satz gezählt. In diesem Fall wäre das Vokabular: {Das, Auto, ist, blau, rot}.

      Satz 1 würde dann repräsentiert als: {Das: 1, Auto: 1, ist: 1, blau: 1, rot: 0}
      Satz 2 würde repräsentiert als: {Das: 1, Auto: 1, ist: 1, blau: 0, rot: 1}

- Diese Darstellung ignoriert die Syntax, Semantik und Pragmatik des Textes und ist dennoch in der Lage, gute Ergebnisse in der IR-Leistung zu erzielen.
- Die BoW-Darstellung ist die am häufigsten verwendete Darstellung im Information Retrieval aufgrund ihrer Einfachheit und Geschwindigkeit.

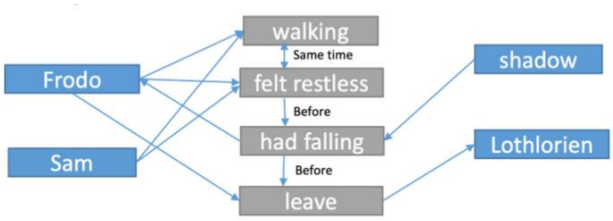
SCHWACH STRUKTURIERTE DARSTELLUNG

- Die schwach strukturierte Textdarstellung berücksichtigt bestimmte Gruppen von Wörtern, z.B. **Bag-of-Nouns** oder benannte Entitäten (Personen bzw. Entitäten), während andere Wörter entweder reduziert oder komplett ignoriert werden.
    - Bag of nouns
    - {(Evening, 1), (shadow, 1), (twilight, 1), … }
    - Bag of named entities
    - {(Lothlorien, 1), (Frodo, 2), …}
- Dies erfordert zusätzliche NLP-Tools wie einen POS-Tagger oder einen Named Entity Recognizer (NER) zur Identifizierung von benannten Entitäten. Diese Darstellung reduziert die Größe des Vokabulars und kann zu einer besseren IR-Leistung führen.

STRUKTURIERTE DARSTELLUNG

- Strukturierte Textdarstellungen nutzen semantische Beziehungen zwischen Begriffen und Konzepten und stellen sie als Knoten und Kanten in Graphen dar, auch bekannt als Wissensgraphen.
- Diese Darstellung erfordert anspruchsvolle Informationsextraktions- und NLP-Tools, um die semantischen Beziehungen zwischen Begriffen und Konzepten zu erkennen.
- Obwohl sie eine viel detailliertere Darstellung des Textes ermöglicht, ist sie **weniger gebräuchlich im Information Retrieval**, da sie aufgrund ihrer Komplexität und Ressourcenintensität weniger praktisch und effizient ist.

Graph-Struktur



TEXTVORBEREITUNG

Die Text-Preprocessing ist ein wichtiger Schritt bei der Vorbereitung von Texten für den Suchprozess und umfasst Schritte wie die Extraktion von reinem Textinhalt, die Spracherkennung, die Tokenisierung, die morphologische Normalisierung und die Entfernung von Stoppwörtern, um die Größe des Vokabulars zu reduzieren und Rauschen zu entfernen. Nach der Vorverarbeitung ist der Text bereit, indiziert zu werden, was in kommenden Vorlesungen diskutiert wird.

STOPWORDS

Stopwords sind semantisch arme Wörter, wie Artikel, Präpositionen, Konjunktionen und Pronomen, die oft aus Texten entfernt werden, um die Anzahl der Terme im Index zu reduzieren. Die Entfernung von Stopwörtern kann dazu beitragen, den Index zu verkleinern und die Effizienz von IR-Systemen zu verbessern. Es gibt Listen von Stopwörtern für verschiedene Sprachen, die in der IR-Praxis verwendet werden.

STEMMING

Stemming ist ein Prozess, bei dem verschiedene Formen desselben Wortes auf ihre grammatikalische Wurzel reduziert werden, um eine höhere Übereinstimmung zwischen Abfrage und Dokumenten zu erreichen. Es werden häufig regelbasierte Algorithmen wie der Porter-Algorithmus verwendet, der Phasen der Suffix-

Entfernung verwendet, um Wörter zu reduzieren. Das Stemming kann in IR-Systemen nützlich sein, da es die Anzahl der Terme reduziert und somit das Volumen des Index reduziert.

PORTER-ALGORITHMUS

Der Porter-Algorithmus ist ein häufig verwendeter Stemming-Algorithmus, der auf regelbasierten Algorithmen und Phasen der Suffix-Entfernung basiert. Der Algorithmus entfernt sukzessive Suffixe (-ing, -heit, …) von einem Wort, bis die Stammform erreicht ist. Der Porter-Algorithmus wurde für die englische Sprache entwickelt, aber ähnliche Algorithmen wurden für andere Sprachen entwickelt.

TOKENISIERUNG

Tokenisierung ist der Prozess, bei dem ein Text in eine Sequenz von Tokens oder Termen aufgeteilt wird. Es bekommt jedes Wort einen Token zugewiesen. Tokenisierung kann manuell oder automatisch durchgeführt werden. Die Tokenisierung kann für IR-Systeme von entscheidender Bedeutung sein, da die Ergebnisqualität von Abfragen von der Qualität der Tokenisierung abhängen kann.

NORMALISIERUNG / LEMMATISIERUNG

Die Normalisierung ist ein Prozess der Standardisierung von Wörtern. Es gibt verschiedene Arten der Normalisierung, z.B. Inflektionsbedingte Normalisierung (Lemmatisierung), die alle verschiedenen Formen eines Wortes in ihre Basisform reduziert. (Häuser→ Haus) Eine andere Art der Normalisierung ist die Derivationsbedingte Normalisierung, die alle abgeleiteten Formen (Zerstörung → zerstören) eines Wortes auf das Originalwort reduziert. Die Normalisierung kann helfen, das Volumen des Indexes zu reduzieren und die Ergebnisqualität von Abfragen zu verbessern.

BOOLEAN RETRIEVAL

DEFINITION

Boolean Retrieval ist ein Modell des Information Retrieval (IR), bei dem eine Abfrage als Boolesche Ausdrücke formuliert wird, die aus logischen Operatoren wie **AND**, **OR** und **NOT** ($\wedge$, $\vee$, $\neg$) bestehen. Das Modell geht davon aus, dass Dokumente als Menge von Wörtern oder Begriffen dargestellt werden können und dass eine Anfrage als Menge von Bedingungen oder Einschränkungen formuliert werden kann, die diese Wörter oder Begriffe enthalten oder nicht enthalten müssen.

- Google suche: «Frodo gave Sam the sword» → $"Frodo"\ AND\ "gave"\ AND\ "Sam"\ AND\ "sword"$

BESCHREIBUNG

Die grundlegende Idee von Boolean Retrieval ist es, Dokumente zu finden, die einer bestimmten Anfrage entsprechen, indem man die Mengen von Dokumenten, die den Bedingungen in der Abfrage entsprechen, mithilfe von Booleschen Operatoren kombiniert. Zum Beispiel kann eine Anfrage nach Dokumenten suchen, die die Wörter "Information" und "Retrieval" enthalten und nicht das Wort "Web". Die Boolesche Operation AND wird verwendet, um Dokumente zu finden, die beide Wörter enthalten, während NOT verwendet wird, um Dokumente auszuschließen, die das Wort "Web" enthalten.

Das Boolean Retrieval-Modell ist einfach und schnell, aber es hat einige Einschränkungen.

- schwierig, komplexe Abfragen zu formulieren, die mehrere Bedingungen gleichzeitig erfüllen
- keine Möglichkeit von Bedingungen (IF)
- Keine Relevanz der Dokumente, die die Bedingungen der Abfrage erfüllen, da alle Dokumente, die die Bedingungen erfüllen, gleichbehandelt werden.

INVERTED INDEX

Der Inverted Index ist eine Datenstruktur, die häufig in Suchmaschinen verwendet wird, um schnell zu bestimmen, welche Dokumente einen bestimmten Suchbegriff oder eine Phrase enthalten. Der Index ist "invertiert", weil er den umgekehrten Ansatz verfolgt, im Vergleich zu einer typischen Liste von Dokumenten und ihren enthaltenen Wörtern. Statt für jedes Dokument eine Liste der Wörter zu führen, die es enthält, wird für jedes Wort eine Liste der Dokumente erstellt, in denen es vorkommt. Die Einträge in der Liste verweisen dann auf die Stelle im Dokument, an der das Wort gefunden wurde.

Der Inverted Index ist eine effiziente Möglichkeit, große Mengen an Dokumenten zu durchsuchen, da er es ermöglicht, die Suche auf eine kleinere Menge von Dokumenten zu beschränken, die die Suchbegriffe enthalten. Der Index wird normalerweise durch Crawlen und Indizieren von Dokumenten aufgebaut, die dann in einem Suchindex gespeichert werden.

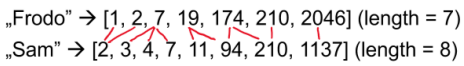
BEISPIEL

- Inverted_index = ["$FRODO$": [1, 3, 7, 33, 85], "SAM": [1, 7, 43, 69]]
    - Das Wort Frodo ist in den Dokumenten 1, 3, 7, 33, 85 zu finden.

POSTING LISTS

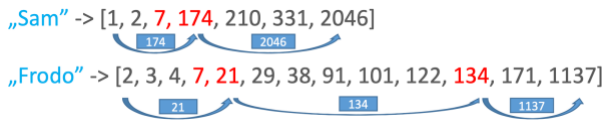
Eine Posting List ist eine Liste von Dokumenten, die ein bestimmtes Wort enthalten, z.B. "$FRODO$": [1, 3, 7, 33, 85]. Wenn nach diesem Wort gesucht werden soll, weiss man genau in welchem Dokument gesucht werden muss.

- Wenn eine Posting Liste *sortiert* ist, ist die Komplexität linear.
    - maximal: $O(x + y)$



    - maximal: $O(7 + 8)$: da Frodo 5 und Sam 4 Einträge haben
    - In diesem Beispiel: Komplexität = 11 (11 rote Linien)
- Ist die Posting Liste *nicht sortiert*:
    - $O(x * y)$
    - $O(20) = O(5 * 4)$
    - Alle Elemente der Posting Listen müssen abgeglichen werden

SKIP-POINTER



Skip Pointer steigern die Effizienz bei der Suche in Posting Listen. Mit Hilfe von SP werden Elemente in der Liste übersprungen und geprüft ob das Element <= dem Element der Liste 1 ist. Falls dies nicht der Fall ist, wird der Skip Pointer ausgelassen. Falls ja, werden die Elemente übersprungen.

Dabei stellt sich die Frage, wie viele Skip Pointer sollen in der Datenstruktur enthalten sein?

- Faustregel: $\sqrt{L}$ wo bei L die Länge der Posting List darstellt
- Einfach zu unterhalten in einer read-only Struktur, da sich die Länge der Posting List nicht ändert.
- Nicht empfehlenswert, wenn die Liste häufig geupdatet wird

RANKED RETRIEVAL

Ranked Retrieval (auch bekannt als Relevanzbewertung oder gewichtete Suche) ist ein Ansatz in der Informationsrückgewinnung, bei dem Suchergebnisse basierend auf ihrer Relevanz oder Bedeutung für den Benutzer sortiert werden. Bei Ranked Retrieval werden den Dokumenten in der Posting-Liste Gewichte oder Relevanzwerte zugewiesen, die sich aus verschiedenen Faktoren wie der Häufigkeit und Position der Suchbegriffe im Dokument, der Länge des Dokuments und der Popularität des Dokuments ergeben können. Die Gewichtungsfaktoren werden normalerweise durch statistische Modelle und Algorithmen berechnet, die auf dem Korpus oder der Sammlung von Dokumenten basieren.

- Durch Ranked Retrieval werden die für den Benutzer relevantesten Themen absteigend sortiert angezeigt. (Vermeidung von Overload)

RANKING PRINCIPLES

RP1: Occurrence of search term
RP2: Total occurrence of search terms
RP3: Rarity of the occurring search terms
RP4: Distance of the occurring search terms
RP5: Position of the occurring search terms

- RP1: Häufigkeit bzw. Anzahl des Suchbegriffs im Dokument
- RP2: Häufigkeit des Suchbegriffs im Verhältnis zu allen Worten im Dokument (TF) $\frac{count\ term}{total\ count\ term}$
- RP3: Seltenheit des Suchbegriffs im Dokument (IDF) $\frac{count\ of\ documents}{count\ of\ documents\ with\ term}$
- RP4: Die Distanz der Suchterme zueinander (je näher desto besser)
- RP5: Position des Suchterms (Titel höhere Relevanz)

VECTOR SPACE MODEL

Jedes Dokument wird als Vektor dargestellt, wobei die Einträge die Worte und die Häufigkeit des Auftretens darstellen. In Dokument 1 kommen Auge und Zahn je zwei Mal vor.

D1	“Auge um Auge, Zahn um Zahn”	[2,2,0,0,0,0,0,0]
D2	“aus den Augen, aus dem Sinn”	[1,0,1,0,0,0,0,0]
D3	“das Auge isst mit”	[1,0,0,1,0,0,0,0]
D4	“Schönheit liegt im Auge des Betrachters”	[1,0,0,0,1,1,1,0]
D5	“Schönheit vor Alter”	[0,0,0,0,1,0,0,1]

Wenn nun ein Query formuliert wird, wir aus diesem wieder ein Vektor erstellt.

Q1	Alter Augen	Q1	[1,0,0,0,0,0,0,1]

Nun wird die Cosinus-Ähnlichkeit von Q_1 und den Dokumenten D_k genutzt, um die Ähnlichkeit des Dokuments zu dem Query zu berechnen. Je kleiner der Winkel zwischen den zwei Vektoren umso ähnlicher sind sie.

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} (a_i)^2} \cdot \sqrt{\sum_{i=1}^{n} (b_i)^2}}$$

TF-IDF

- TF-IDF ist ein Ranking-Algorithmus, der häufig in der Information Retrieval und der Textanalyse verwendet wird, um die Relevanz von Dokumenten für eine bestimmte Suchanfrage zu bewerten. Es ist eine Kombination aus zwei Komponenten: Term Frequency (TF) und Inverse Document Frequency (IDF).
- Die Term Frequency (TF) ist ein Maß dafür, wie oft ein bestimmter Suchbegriff in einem Dokument vorkommt. Es wird berechnet, indem die Anzahl der Vorkommen des Suchbegriffs im Dokument durch die Gesamtzahl der Wörter im Dokument dividiert wird. Eine höhere TF zeigt an, dass der Suchbegriff in dem Dokument wichtiger ist.
- Die Inverse Document Frequency (IDF) ist ein Maß dafür, wie selten ein bestimmter Suchbegriff in der gesamten Sammlung von Dokumenten vorkommt. Es wird berechnet, indem die Gesamtzahl der Dokumente in der Sammlung durch die Anzahl der Dokumente dividiert wird, die den Suchbegriff enthalten. Eine höhere IDF zeigt an, dass der Suchbegriff in der Sammlung seltener vorkommt und daher wichtiger ist.
- Die TF-IDF-Bewertung eines Dokuments wird durch Multiplikation der TF- und IDF-Werte aller Suchbegriffe im Dokument berechnet. Dies führt dazu, dass Dokumente, die viele Vorkommen wichtiger Suchbegriffe enthalten, höhere TF-IDF-Werte und somit eine höhere Relevanzbewertung für die Suchanfrage erhalten.

$$TF = \frac{count\ term}{total\ count\ term}$$

$$IDF = \log\left(\frac{count\ of\ documents}{count\ of\ documents\ with\ term}\right)$$

$$TF * IDF = TFIDF\ Score$$

DATA PROCESSING

RELATIONALES MODELL

Das relationale Modell ist ein Datenmodell, das Daten in Tabellen oder Relationen organisiert. Jede Tabelle besteht aus Spalten mit eindeutigen Namen und Datentypen, während jede Zeile einen Datensatz darstellt. Beziehungen zwischen Tabellen können durch Verknüpfungen von Schlüsseln definiert werden, um komplexe Abfragen und Datensuche zu ermöglichen. Obwohl es auch andere Datenmodelle gibt, wie beispielsweise das dokumentorientierte Modell oder das graphenorientierte Modell, bietet das relationale Modell immer noch viele Vorteile, die es zu einer beliebten Wahl für die Verwaltung von Daten machen.

Vorteile Relationales Modell:

1. **Einfache Struktur**: Das relationale Modell ist einfach und intuitiv, da es auf einer tabellarischen Struktur basiert, die den meisten Menschen vertraut ist.
2. **Flexibilität**: Das relationale Modell ist flexibel und kann für verschiedene Anwendungen angepasst werden, da es eine Vielzahl von Datenstrukturen unterstützt, einschließlich einfacher Tabellen und komplexer Beziehungen.
3. **Einfache Abfrage**: Das relationale Modell bietet eine einfache Abfragesprache (SQL), mit der Daten schnell und einfach abgefragt werden können.
4. **Skalierbarkeit**: Relationale Datenbanken können auf große Datenmengen skalieren und bieten die Möglichkeit, Daten effizient zu organisieren und zu verwalten.
5. **Sicherheit**: Das relationale Modell bietet Mechanismen zur Überwachung und Sicherung von Daten, um unbefugten Zugriff und Datenverlust zu vermeiden.

RECORD LINKAGE

Record Linkage, auch als Datenverknüpfung oder Deduplizierung bezeichnet, ist ein Prozess der Identifizierung und Zusammenführung von Datensätzen aus verschiedenen Datenquellen, die sich auf dieselben realen Objekte oder Ereignisse beziehen. Das Ziel der Record Linkage besteht darin, genaue, einheitliche und konsistente Datensätze zu erstellen, die eine umfassende Sicht auf die realen Objekte oder Ereignisse bieten.

Im Prozess der Record Linkage werden Datensätze aus verschiedenen Quellen basierend auf bestimmten Attributen oder Merkmalen wie Name, Adresse oder Geburtsdatum miteinander verglichen, um ähnliche Datensätze zu identifizieren. Dies kann durch den Vergleich von Ähnlichkeiten oder durch die Verwendung von Regeln und Wahrscheinlichkeiten erfolgen.

Ein Beispiel für die Anwendung von Record Linkage ist die Zusammenführung von Kundendaten aus verschiedenen Systemen in einem Unternehmen. Durch die Zusammenführung von Daten aus verschiedenen Quellen können Unternehmen eine einheitliche und konsistente Sicht auf ihre Kunden erhalten, was zu fundierten Entscheidungen und besseren Geschäftsergebnissen führen kann.

LEVENSTEIN DISTANCE

Um Daten oder Tabellen zusammenzuführen, kann die Levenstein Distanz verwendet werden um die Ähnlichkeit zweier Worte zu vergleichen.

Beispiel:

- WORDS vs. SWORD → Distanz = 2

1. Erstellen Sie eine Matrix mit den Abmessungen (m+1) x (n+1), wobei m und n die Längen der beiden zu vergleichenden Zeichenketten sind. Initialisieren Sie die erste Zeile und die erste Spalte mit ihren Indizes, beginnend bei 0 und inkrementieren Sie um 1. Wenn eine Zelle leer ist, füllen Sie sie mit dem entsprechenden Zeichen aus der Zielzeichenkette.

Beispiel: Wenn die beiden Zeichenketten "WORDS" und "SWORD" lauten, würde die Matrix so aussehen:

		W	O	R	D	S
	0	1	2	3	4	5
S	1	1	2	3	3	4
W	2	1	2	3	4	4
O	3	2	1	2	3	4
R	4	3	2	1	2	3
D	5	4	3	2	1	2

Buchstaben unterschiedlich

+1	+1
+1	

Buchstaben gleich

0	+1
+1	

2. Lösen Sie Teilprobleme:

- Wenn die Zeichen in der Zeile und Spalte übereinstimmen, ist der Wert der gleiche wie in der letzten Zelle (diagonalen)
- Wenn die Zeichen nicht übereinstimmen, erhöht sich der Wert der letzten Zelle um 1 und es wird ein Feld heruntergerutscht

BIG DATA

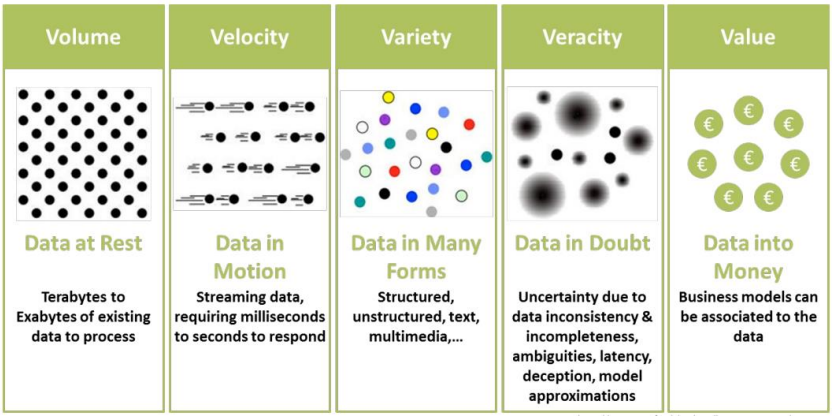
Traditionelle Datenbanksysteme sind nicht geeignet für Big Data: Traditionelle relationale Datenbanken (RDBMS) sind nicht für die Verarbeitung des Umfangs, der Geschwindigkeit und der Vielfalt von Big Data konzipiert. RDBMS haben feste Schemata, die es schwierig machen, unstrukturierte Daten zu speichern und zu verwalten. Darüber hinaus wurden RDBMS nicht für verteilte Systeme konzipiert, und ihre Leistung nimmt ab, je mehr Daten verarbeitet werden müssen.

- Ungefähr 80% der Daten in einem Unternehmen sind unstrukturiert: Unstrukturierte Daten wie Bilder, Videos und Social-Media-Beiträge sind aufgrund ihres starren Schemas schwer in RDBMS zu speichern und zu analysieren.
- Neue Lösung: NoSQL (Not Only SQL): NoSQL-Datenbanken sind für die Verarbeitung des Umfangs, der Geschwindigkeit und der Vielfalt von Big Data konzipiert. Sie bieten ein flexibles Schema, das die Speicherung und Verarbeitung von unstrukturierten Daten ermöglicht, und sie verwenden eine Shared-Nothing-Architektur, die eine horizontale Skalierbarkeit ermöglicht.

BIG DATA V'S

Die Big Data V's, auch bekannt als die Big Data Dimensionen, sind fünf grundlegende Eigenschaften von Big Data, die helfen, die Herausforderungen und Möglichkeiten im Zusammenhang mit der Analyse großer Datenmengen zu verstehen. Nicht alle V's müssen erfüllt sein, damit eine Datenansammlung als Big Data gilt. Die fünf Big Data V's sind:

1. **Volume** (Volumen): Dies bezieht sich auf die enorme Menge an Daten, die von verschiedenen Quellen generiert werden. Unternehmen und Organisationen müssen in der Lage sein, diese Daten in großem Maßstab zu speichern und zu verwalten.
2. **Velocity** (Geschwindigkeit): Big Data wird in Echtzeit generiert und muss in Echtzeit verarbeitet werden. Unternehmen und Organisationen müssen in der Lage sein, Daten in schneller Geschwindigkeit zu sammeln, zu verarbeiten und zu analysieren, um wertvolle Einblicke in Echtzeit zu gewinnen.
3. **Variety** (Varietät): Big Data kann in verschiedenen Formaten und von verschiedenen Quellen wie strukturierten, unstrukturierten und halbstrukturierten Daten vorliegen. Es ist wichtig, die Vielfalt der Daten zu verstehen und sie auf sinnvolle Weise zu integrieren und zu verarbeiten.
4. **Veracity** (Wahrhaftigkeit): Big Data kann oft ungenau oder unvollständig sein, und es ist wichtig, die Qualität der Daten zu verstehen und sicherzustellen, dass sie zuverlässig sind.
5. **Value** (Wert): Big Data bietet Potenzial für wertvolle Erkenntnisse, die Unternehmen und Organisationen helfen können, bessere Entscheidungen zu treffen und ihre Geschäftsprozesse zu verbessern. Es ist wichtig, den Wert der Daten zu verstehen und sicherzustellen, dass die Analyse der Daten einen tatsächlichen Nutzen bringt.



SKALIERBARKEIT

Skalierbarkeit bezieht sich auf die Fähigkeit eines Systems, seine Leistungsfähigkeit zu erhöhen oder zu verringern, um die Anforderungen einer wachsenden oder schrumpfenden Benutzerzahl oder einer zunehmenden oder abnehmenden Datenmenge zu erfüllen. Ein skalierbares System sollte in der Lage sein, mit wachsender Belastung oder größerer Komplexität umzugehen, ohne dabei an Leistung oder Verfügbarkeit zu verlieren.

ARTEN VON SKALIERBARKEIT

Horizontale Skalierbarkeit bedeutet, dass ein System durch Hinzufügen weiterer gleichartiger Ressourcen, wie z.B. Server oder Rechenzentren, skalierbar ist. Dies wird auch als "**Scaling-Out**" bezeichnet.

Vertikale Skalierbarkeit bedeutet, dass ein System durch Hinzufügen zusätzlicher Ressourcen auf derselben Ebene, wie z.B. durch Hinzufügen von mehr RAM oder CPU-Leistung, skalierbar ist. Dies wird auch als "**Scaling-Up**" bezeichnet. Wird für Big Data eingesetzt.

Scaling Up verwendet Cluster von Rechenmaschinen. Ein Cluster ist eine Sammlung von miteinander verbundenen Computern oder Servern, die auf der Shared-Nothing-Architektur basieren. Jeder Knoten innerhalb des Clusters hat seine eigene CPU, seinen eigenen Arbeitsspeicher und seine eigenen Festplatten und läuft unter seinem eigenen Betriebssystem. Die Knoten interagieren miteinander durch das Senden von Nachrichten. Das Verteilen von Daten, Abfragen, Berechnungen und Arbeitslasten auf die Knoten innerhalb des Clusters ermöglicht eine höhere Skalierbarkeit. Es kann jedoch auch sinnvoll sein, einen einzelnen Knoten zu verwenden, insbesondere für Anwendungen wie Graphdatenbanken, bei denen es schwierig ist, die Daten zu verteilen. Horizontal skalierbare Systeme, wie Cluster, eröffnen jedoch viele Möglichkeiten und sind für Big Data notwendig.

SHARDING / PARTITIONIERUNG

Sharding bezieht sich auf die Technik der horizontalen (und vertikale) Partitionierung von Daten in eine Datenbank, um die Verarbeitung von Daten in einem verteilten System zu erleichtern und zu beschleunigen. Das bedeutet, dass die Datenbank in mehrere kleineren und unabhängigen Datenbanken aufgeteilt wird, die als Shards bezeichnet werden. Jeder Shard enthält einen Teil der Daten, und jede Abfrage wird auf allen relevanten Shards ausgeführt, um die Ergebnisse zu sammeln und zurückzugeben.

Das Sharding wird verwendet, um Datenbanken skalierbar zu machen, indem es die **Last** auf **mehrere Server** verteilt. Dadurch wird die **Verarbeitungszeit** von Abfragen **verringert** und es können höhere Anforderungen an die Leistung der Datenbank erfüllt werden. Ein weiterer Vorteil von Sharding ist, dass es die **Ausfallsicherheit** erhöht, da ein Ausfall eines einzelnen Shards keine Auswirkungen auf die gesamte Datenbank hat.

SHARDING METHODEN

Sharding ist eine Methode zur Datenverteilung, bei der große Datenbanken in kleinere, schnellere und leichter zu handhabende Teile unterteilt werden, die als "Shards" bezeichnet werden. Jeder Shard enthält eine Teilmenge der Gesamtdaten und funktioniert als eigenständige Datenbank.

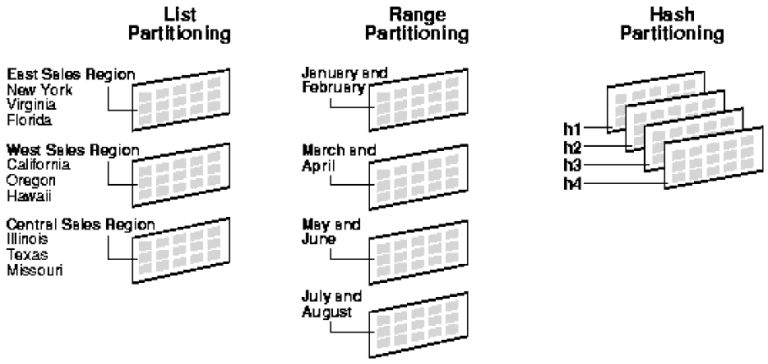
Es gibt verschiedene Strategien, um zu bestimmen, wie Daten in einem sharding System verteilt werden. Hier sind einige gängige Sharding-Strategien:

1. Range-based Sharding: Bei dieser Methode werden Daten anhand eines festgelegten Bereichs verteilt. Zum Beispiel könnten Kunden in einer Datenbank anhand ihrer Kundennummern ge-sharded werden, so dass Kunden 1-1000 auf einem Shard, Kunden 1001-2000 auf einem anderen Shard usw. liegen. Der Hauptvorteil dieser Methode ist ihre Einfachheit, aber sie kann zu ungleichmäßiger Lastverteilung führen, wenn die Daten nicht gleichmäßig über die Bereiche verteilt sind.

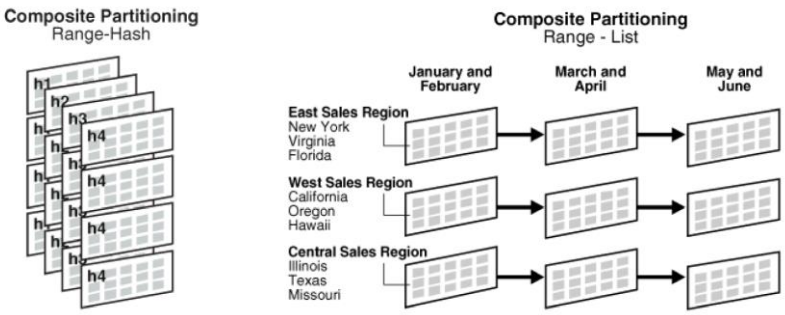
2. Hash-based Sharding: Bei dieser Methode wird eine Hashfunktion auf den Shardschlüssel angewendet, um den Shard zu bestimmen, auf dem die Daten gespeichert werden. Dies führt im Allgemeinen zu einer gleichmäßigeren Verteilung der Daten, kann aber zu Schwierigkeiten bei der Durchführung von Bereichsanfragen führen (z. B. finde alle Kunden mit Kundennummern zwischen 1001 und 2000), da die gehashten Werte nicht notwendigerweise in der gleichen Reihenfolge liegen wie die Originalwerte.

3. List-based Sharding: Bei dieser Methode wird eine explizite **Liste von Werten oder Kategorien** verwendet, um Daten auf Shards zu verteilen. Zum Beispiel könnten Kunden auf Shards basierend auf ihrem Wohnsitzstaat verteilt werden, wobei jeder Staat einen eigenen Shard hat. Diese Methode kann sinnvoll sein, wenn die Daten natürliche Gruppen bilden, kann aber ebenfalls zu ungleichmäßiger Lastverteilung führen.

4. Composite Sharding: Auch bekannt als multi-level sharding, beinhaltet diese Methode die Kombination von zwei oder mehr Sharding-Strategien. Beispielsweise könnte eine Datenbank zunächst auf der Grundlage von Ranges ge-sharded werden, und dann könnte innerhalb jedes dieser Ranges eine Hash-basierte Sharding-Strategie angewendet werden. Composite Sharding kann dazu beitragen, die Stärken der einzelnen Sharding-Strategien zu nutzen und ihre Schwächen zu minimieren.

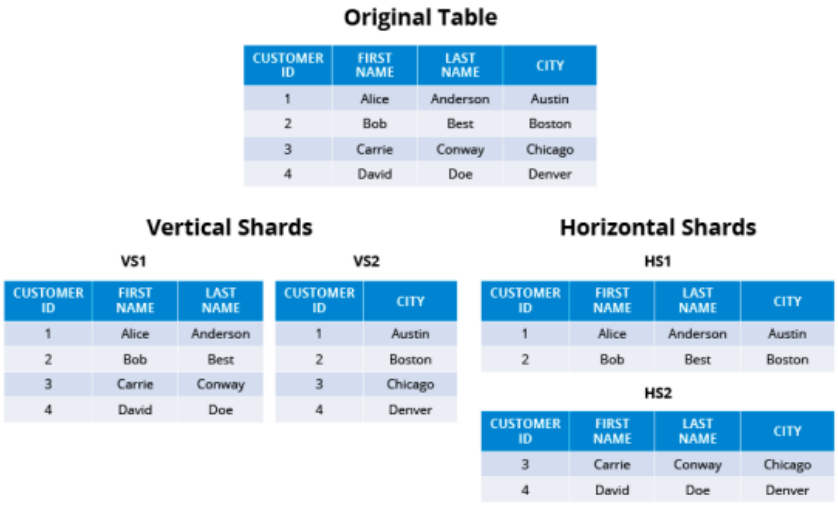


Composit Sharding:



ZIELE

- eine einheitliche Datenverteilung erreichen
- ausgewogene Arbeitslast erreichen (Lese- und Schreibanfragen)
- respektieren physische Standorte
- z. B. verschiedene Rechenzentren für Benutzer auf der ganzen Welt



SCHWIERIGKEITEN

- Entscheidung, während Schreibvorgängen, auf welchem Shard die Daten gespeichert werden sollen
- Entscheidung während Lesevorgängen, auf welchem Shard vorhandene Daten abgerufen werden sollen
- Suche nach Shards basierend auf Suchkriterien wie Schlüssel oder ID
- Struktur des Clusters kann sich ändern, z.B. durch Hinzufügen oder Entfernen von Knoten
- Cluster-Konfiguration kann veraltet oder unvollständig sein
- Ausfall von einzelnen Knoten möglich
- Netzwerknachrichten können nicht zugestellt werden

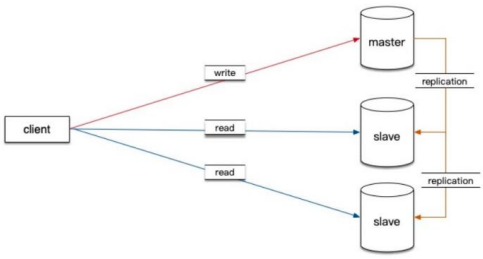
REPLICATION

Replication (Replikation) bezieht sich auf die Technik der Erstellung und Aufrechterhaltung von Kopien von Daten in einer oder mehreren Datenbanken. Das Ziel der Replikation besteht darin, sicherzustellen, dass die Daten in verschiedenen Standorten und Systemen synchronisiert werden, um eine höhere Verfügbarkeit, Ausfallsicherheit und Leistung zu gewährleisten.

Es gibt zwei Arten von Replikation: **Master-Slave-Replikation** und **Peer-to-Peer**-Replikation.

Master-Slave-Replikation

- Bei der Master-Slave-Replikation gibt es einen zentralen Datenbank-Server (der Master), der Änderungen an den Daten vornimmt, und eine oder mehrere **Slave-Datenbanken**, die **Kopien der Daten vom Master-Server** halten.
- Der Master-Server ist der Einzige, der **Schreibzugriffe a**uf die Datenbank ausführt, während die Slave-Server nur Lesezugriffe auf die Daten ausführen können. Sollte der Master Server ausfallen, muss dieser ersetzt werden.
- Der Master kann aus Gründen der Performance oder Ausfällen ein Bottleneck darstellen.



Peer-to-Peer-Replikation

- Peer-to-Peer (P2P) ist eine Art von Netzwerk, bei dem alle Teilnehmer gleichberechtigt sind und direkt miteinander kommunizieren können.
- Alle Server halten die selben Daten
- Im Gegensatz zu Master-Slave-Architekturen, können alle Knoten (Server) lesen und schreiben.
- Kein Bottleneck

P2P-Netzwerke werden oft in verteilten Systemen eingesetzt, um Daten oder Ressourcen gemeinsam zu nutzen. Ein bekanntes Beispiel hierfür sind Filesharing-Netzwerke, bei denen Benutzer Dateien direkt miteinander teilen, anstatt diese über einen zentralen Server herunterzuladen.

P2P-Netzwerke haben den Vorteil, dass sie widerstandsfähiger gegen Ausfälle sind, da kein einzelner Server ausfällt, der das gesamte Netzwerk beeinträchtigt. Auch die Skalierbarkeit kann verbessert werden, da neue Knoten einfach hinzugefügt werden können, um die Last auf das Netzwerk zu verteilen. Allerdings kann die dezentralisierte Natur von P2P-Netzwerken auch Herausforderungen mit sich bringen, wie z.B. Schwierigkeiten bei der Kontrolle von Daten und der Verhinderung von Missbrauch. Die Replikation ist besonders nützlich in Systemen, die eine hohe Verfügbarkeit erfordern, wie z.B. in der Finanz- und E-Commerce-Branche, sowie in verteilten Systemen, die in verschiedenen geografischen Regionen eingesetzt werden. Wenn ein Server ausfällt, können die anderen Server weiterhin auf die Daten zugreifen und die Arbeitsbelastung übernehmen.

Es gibt jedoch auch Herausforderungen im Zusammenhang mit der Replikation, wie z.B. die Notwendigkeit der Synchronisierung der Daten zwischen den Servern, um sicherzustellen, dass sie konsistent bleiben. Wenn Änderungen auf verschiedenen Servern gleichzeitig vorgenommen werden, müssen Konflikte behandelt werden, um sicherzustellen, dass die Daten in allen Servern synchron bleiben. Zudem können die Kosten und der Aufwand für die Einrichtung und Wartung von Replikationsservern hoch sein.

CONSISTENT HASHING

Consistent Hashing (Konsistentes Hashing) ist eine Technik, die bei der Verteilung von Daten in einem verteilten System verwendet wird. Es wird verwendet, um die Zuordnung von Daten zu Servern effizienter und skalierbarer zu machen. Einer der kompliziertesten Fälle bei der Replikation ist die Peer-to-Peer-Replikation mit Sharding. Hierbei können Knoten hinzukommen oder entfernt werden, jedoch muss eine Replication der Daten sichergestellt werden.

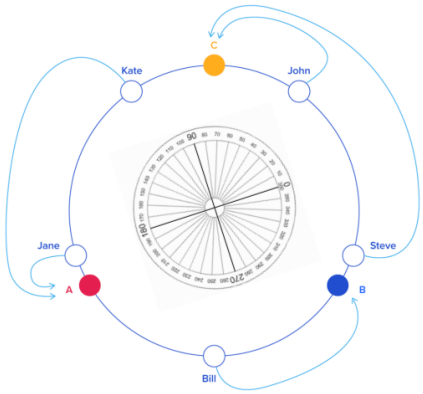
Um diese Herausforderungen zu lösen, wird oft das Prinzip des "consistent hashing" eingesetzt. Dabei wird jeder Knoten und jedes Objekt in einem verteilten System einem Platz auf einem abstrakten Kreis oder Hash-Ring zugewiesen, unabhängig von der Anzahl der Knoten oder Objekte. Dies ermöglicht es, dass Knoten und

Objekte skaliert werden können, ohne das gesamte System zu beeinträchtigen. Nur k/N Schlüssel müssen neu zugewiesen werden, wenn k die Anzahl der Schlüssel und N die Anzahl der Server ist.

Durch die Verwendung von consistent hashing können **P2P-Replikation** und **Sharding** erfolgreich zusammen verwendet werden, um eine effektive und skalierbare Datenverteilung zu erreichen.

Die Idee hinter konsistentem Hashing besteht darin, dass **jeder Knoten** im System durch einen **Hashwert** identifiziert wird. Die zu verteilenden **Daten** werden auch **gehasht**, um einen Wert im gleichen Bereich wie die Knoten zu erhalten. Dann wird jeder Wert einem Knoten zugeordnet, indem er dem nächstgrößeren Knotenhash zugeordnet wird, beginnend mit dem Hash des ersten Knotens und dann kreisförmig durch alle Knoten im Uhrzeigersinn.

KEY	HASH	ANGLE (DEG)	LABEL	SERVER
"john"	1632929716	58.7	"C"	C
"kate"	3421831276	123.1	"A"	A
"jane"	5000648311	180	"A"	A
"bill"	7594873884	273.4	"B"	B
"steve"	9786437450	352.3	"C"	C



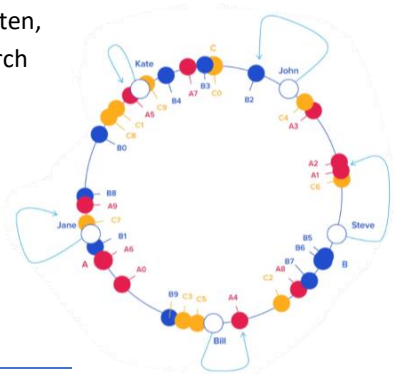
$$Winkel = \frac{hash}{\max(hash)} * 360°$$

Durch diese Methode wird gewährleistet, dass die meisten Daten einem Knoten zugeordnet werden, der ihnen am nächsten liegt, ohne dass die Zuordnung von Daten geändert werden muss, wenn neue Knoten hinzugefügt oder entfernt werden. Darüber hinaus wird durch diese Methode das Risiko von Hotspots (überlasteten Knoten) verringert, da die Last auf mehrere Knoten verteilt wird.

Eine Herausforderung im Zusammenhang mit konsistentem Hashing besteht darin, dass das System möglicherweise neu konfiguriert werden muss, um das Hinzufügen oder Entfernen von Knoten zu ermöglichen, insbesondere wenn es um große Mengen von Daten geht.

GLEICHMÄSSIGE VERTEILUNG VON SERVERN

Um die Daten gleichmässige auf die Server zu verteilen, können virtuelle Knoten, sogenannte Labels eingefügt werden. Die Gewichtung jedes Servers kann durch die Anzahl virtueller Knoten festgelegt werden.



Beispiel:

- 3 Server: A, B, C
- Server A ist doppelt so stark wie B & C
  A bekommt 10 Labels, B und C bekommen dabei nur 5 Labels

ENTFERNEN EINES SERVERS

Bei Ausfall eines Servers, müssen nur die Keys die jenem Server zugewiesen waren verschoben werden.

NOSQL

NoSQL-Lösungen bieten eine effiziente Methode zum Speichern und Zugreifen auf Daten. Die Dominanz von relationalen Systemen scheint zu Ende zu gehen, während NoSQL-Systeme anfangen, relationale Datenbanksysteme (RDBMS) herauszufordern. Der Name NoSQL ist etwas unglücklich gewählt, da Abfragesprachen immer noch stark an SQL erinnern. Ein besserer Name wäre "nicht-relational". Es gibt eine große Sammlung verschiedener Arten von Systemen.

NoSQL (nicht nur SQL) konzentriert sich auf horizontale Skalierbarkeit (Shared-Nothing-Architektur) und unterstützt nur einen Teil der traditionellen RDBMS. Die Daten werden als Schlüssel-Wert-Paare anstelle von Tabellen gespeichert.

Benutzer bevorzugen NoSQL-Datenbanken aufgrund ihres flexiblen Datenmodells. Sie benötigen kein vorher festgelegtes Schema und keine Datenbereinigung, kein ETL, kein Laden. Sie bieten entspanntere Konsistenzmodelle und tauschen Konsistenz gegen Verfügbarkeit ein, was zu Eventual Consistency führt. Sie haben auch niedrige Anfangskosten für Software und ermöglichen einen schnelleren Erkenntnisgewinn aus der Datenerfassung.

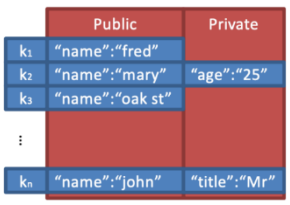
Die sechs Schlüsseleigenschaften von NoSQL-Datenbanken sind:

1. Fähigkeit, einfache Operationen **horizontal** über viele Server zu **skalieren**
2. Fähigkeit, Daten zu **replizieren** und über viele Server zu **verteilen** (Partition)
3. Einfache Call-Level-Schnittstelle oder Protokoll (im Gegensatz zu einer SQL-Bindung)
4. **Schwächeres Nebenläufigkeitsmodell** als ACID-Transaktionen der meisten relationalen (SQL) Datenbanksysteme
5. Effiziente Nutzung von verteilten Indizes und RAM zur Datenspeicherung
6. Fähigkeit, dynamisch neue Attribute zu Datensätzen hinzuzufügen

NOSQL DATENMODELLE

NoSQL-Datenmodelle können in vier Kategorien eingeteilt werden: Key/Value, Dokument, Spaltenfamilie und Graph.

1. **Key-Value-Datenmodell**: (Dict) (Hash-Tabelle) In diesem Modell werden Werte (Daten) auf der Basis von programmiererdefinierten Schlüsseln gespeichert. Das System ist unabhängig von der Struktur (Semantik) des Werts. Abfragen werden in Bezug auf Schlüssel ausgedrückt und Indizes werden über Schlüssel definiert. Einige Systeme unterstützen sekundäre Indizes über (Teile) des Werts. Ein Beispiel hierfür ist Redis.
2. **Dokumentdatenmodell**: In diesem Modell werden Dokumente (Daten) auf der Basis von programmiererdefinierten Schlüsseln gespeichert. Das System ist sich der (beliebigen) Dokumentstruktur bewusst und unterstützt Listen, Zeiger und verschachtelte Dokumente. Abfragen werden in Bezug auf Schlüssel (oder Attribute, falls ein Index existiert) ausgedrückt. Ein Beispiel hierfür ist MongoDB.
3. **Column-Family-Datenmodell**: In diesem Modell werden "Dokumente" auf der Basis einer Spaltenfamilie und eines Schlüssels gespeichert. Das System ist sich der (beliebigen) Struktur der Spaltenfamilie bewusst und verwendet die Spaltenfamilieninformationen, um Daten zu replizieren und zu verteilen. Abfragen werden basierend auf Schlüssel und Spaltenfamilie ausgedrückt.



4. **Graphdatenmodell**: In diesem Modell werden Daten in Form von Knoten und (getypten) Kanten gespeichert. Sowohl Knoten als auch Kanten können (beliebige) Attribute haben. Abfragen werden basierend auf System-IDs ausgedrückt (falls keine Indizes existieren). Sekundärindizes für Knoten und Kanten werden unterstützt.

Jedes Datenmodell hat seine eigenen Stärken und Schwächen und ist für bestimmte Anwendungsfälle geeignet. Zum Beispiel bietet das Dokumentdatenmodell Schema-Flexibilität und eine bessere Leistung aufgrund der Lokalität, während das relationale Datenmodell eine bessere Unterstützung für Joins und viele-zu-eins- sowie viele-zu-viele-Beziehungen bietet. Die Wahl des richtigen Datenmodells hängt von den Anforderungen der Anwendung und den Beziehungen zwischen den Datenobjekten ab.

KONSISTENZMODELLE

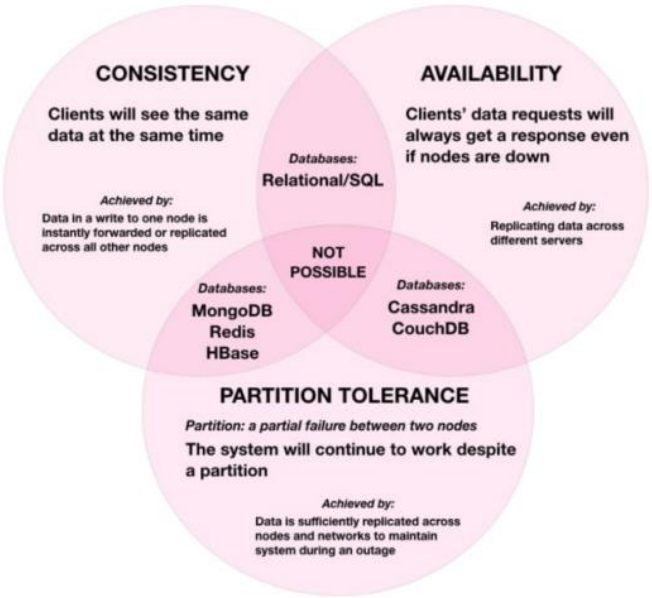
Das **CAP-Theorem**, auch bekannt als Brewers Theorem, stellt fest, dass ein verteiltes Datenbanksystem nur zwei der drei folgenden Eigenschaften gleichzeitig erfüllen kann:

Konsistenz (C), **Verfügbarkeit** (A) und **Ausfalltoleranz** (P).

1. **Konsistenz** (C): (Bank/ Business) Dies bezieht sich auf die Garantie, dass alle Knoten in einem verteilten System **zu jeder Zeit die gleichen Daten anzeigen**. Wenn Daten an einem Ort aktualisiert werden, muss diese Aktualisierung sofort auf allen anderen Knoten repliziert werden. (ACID)
    a. A – Atomarität (Alles oder nichts)
    b. C – Konsistenz (keine Inkonsistenzen in den Daten)
    c. I – Isolation (parallele Transaktionen beeinflussen sich nicht)
    d. D – Dauerhaftigkeit (Sobald Transaktion abgeschlossen, wird sie dauerhaft gespeichert)

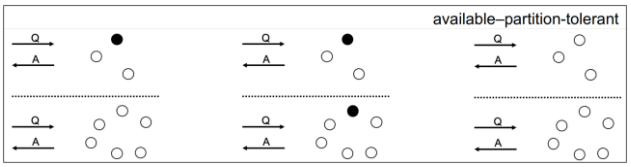
2. **Verfügbarkeit** (A): Dies bezieht sich auf die Garantie, dass alle Anfragen an das System **immer eine Antwort erhalten**, entweder mit den angeforderten Daten oder mit einer Bestätigung der erfolgreich durchgeführten Aktualisierung.

3. **Ausfalltoleranz / Partitionstoleranz** (P): Dies bezieht sich auf die Fähigkeit des Systems, **weiterhin zu funktionieren** und Anfragen zu bedienen, auch wenn Teile des Systems (ein oder mehrere Knoten) ausfallen oder die Kommunikation zwischen den Knoten unterbrochen ist.

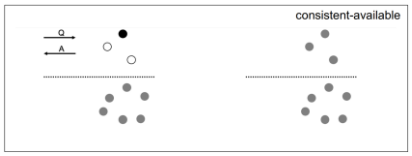


Es gibt verschiedene NoSQL-Datenbanken, die unterschiedliche Paare dieser Eigenschaften erfüllen. Beispiele sind:

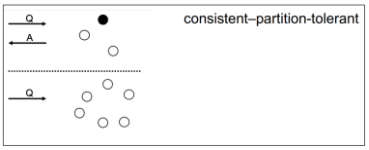
- **Verfügbar und Ausfalltolerant** (A, P): Dynamo, Riak, Voldemort, SimpleDB, CouchDB und Cassandra.
    - Bei Ausfall wird neuer Master wird definiert da ausfalltolerant
    - Deswegen immer verfügbar und Ausfalltolerant



- **Konsistent und Verfügbar** (C, A): Traditionelle relationale Datenbanksysteme (RDBMS), GreenPlum.
    - Darf immer Antworten geben, da Nodes ausgeschalten, wenn Verbindung zu Master lost. → Deswegen konsistent und gibt immer Antwort (wenn Sys nicht down)



- **Konsistent und Ausfalltolerant** (C, P): MemCacheDB, Redis, Scalaris, MongoDB, BigTable, HBase, HyperTable und VoltDB.
    - Bei Ausfall von Master:
        - Keine Antworten zugelassen, da Konsistenz gewährleistet sein muss



NOSQL CONSISTENCY MODELS

1. **Starke Konsistenz**: Nach dem Commit eines Updates gibt jeder nachfolgende Zugriff den aktualisierten Wert zurück.
2. **Schwache Konsistenz**: Das System garantiert nicht, dass nachfolgende Zugriffe den aktualisierten Wert zurückgeben. Eine Reihe von Bedingungen müssen erfüllt sein, bevor der aktualisierte Wert zurückgegeben wird. (Instagram)
    - **Eventuelle Konsistenz**: Ist eine spezifische Form von «Weak Consistency». Bei eventualer Konsistenz gibt es die Garantie, dass bei Beendigung von Updates alle Kopien eines Datenobjekts schließlich den gleichen Wert aufweisen werden.

KONSISTENZPROBLEME

- Schreibkonflikte (zwei Benutzer wollen denselben Datensatz aktualisieren),
- Lese-Konsistenz (ein Benutzer liest, während ein anderer schreibt)
- Replikations-Konsistenz (Sicherstellung, dass derselbe Datenpunkt den gleichen Wert hat, wenn er von verschiedenen Replikaten gelesen wird)

DOKUMENTDATENBANKEN

Dokumentenspeicher sind eine Art von NoSQL-Datenbank, die Daten in Dokumenten speichert und abruft. Diese Dokumente können in Formaten wie XML, JSON usw. vorliegen. Dokumente in diesen Speichern sind selbstbeschreibend und bestehen aus hierarchischen Baumdatenstrukturen. Sie können aus Karten, Sammlungen, Skalarwerten, verschachtelten Dokumenten und mehr bestehen.

Obwohl die Dokumente in einer Sammlung ähnlich sein sollten, kann das Schema der Dokumente unterschiedlich sein, was eine hohe Flexibilität ermöglicht. Dokumentendatenbanken speichern Dokumente im Wertteil des Key-Value-Speichers, und diese Werte sind untersuchbar.

Einige typische Anwendungsfälle für Dokumentenspeicher sind:

1. **Ereignisprotokollierung**: Viele verschiedene Anwendungen möchten Ereignisse protokollieren, und die Art der erfassten Daten ändert sich ständig. Ereignisse können nach dem Namen der Anwendung oder der Art des Ereignisses verteilt werden.
2. **Content-Management-Systeme und Blogging-Plattformen:** Hier werden Dokumentenspeicher für die Verwaltung von Benutzerkommentaren, Benutzerregistrierungen, Profilen, weborientierten Dokumenten usw. verwendet.
3. **Webanalytik oder Echtzeitanalytik**: Teile des Dokuments können aktualisiert werden, und neue Metriken können leicht hinzugefügt werden, ohne Schemaänderungen vornehmen zu müssen.
4. **E-Commerce-Anwendungen**: Dokumentenspeicher bieten ein flexibles Schema für Produkte und Bestellungen und ermöglichen es, Datenmodelle ohne teure Datenmigration weiterzuentwickeln.

Ungeeignete Anwendungsfälle könnten beinhalten:

1. **Komplexe Transaktionen, die sich über verschiedene Operationen erstrecken**
2. **Atomare Operationen über mehrere Dokumente hinweg**
3. **Abfragen gegen eine sich ständig ändernde Aggregatstruktur**

MONGODB

MongoDB ist eine populäre Dokumentendatenbank, die erstmals 2009 veröffentlicht wurde. Sie ist in C++ geschrieben und seit 2018 unter der Server Side Public License Open-Source. Sie unterstützt mehrere Plattformen und verwendet JSON-Dokumente mit dynamischen Schemata.

Einige Hauptmerkmale von MongoDB sind:

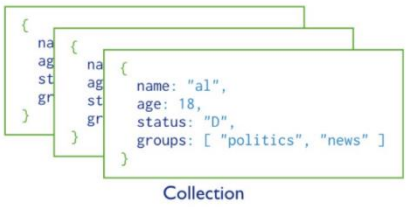
- **Hohe Performance**: MongoDB unterstützt Indizes, die die Abfrageleistung verbessern.
- **Hohe Verfügbarkeit**: Durch Replikation, Eventual Consistency und automatisches Failover kann MongoDB einen kontinuierlichen Betrieb gewährleisten.
- **Automatisches Scaling**: Durch automatisches Sharding über das Cluster hinweg kann MongoDB bei zunehmendem Datenaufkommen skaliert werden.
- **MapReduce-Unterstützung**: MongoDB kann MapReduce-Operationen für datenintensive Berechnungen durchführen.

In MongoDB hat jede Instanz mehrere Datenbanken, und jede Datenbank kann mehrere Sammlungen haben. Beim Speichern eines Dokuments müssen Sie eine Datenbank und eine Sammlung auswählen.

Oracle	MongoDB
database instance	MongoDB instance
schema	database
table	collection
row	document
rowid	_id
join	DBRef

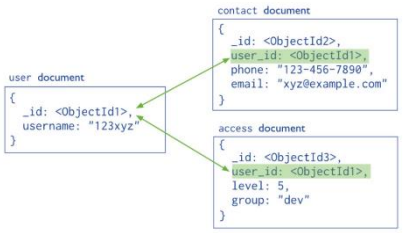
MongoDB verwendet JSON, speichert es aber als BSON, eine binäre Darstellung von JSON. Dokumente haben eine maximale Größe von 16 MB (in BSON), um nicht zu viel RAM zu verwenden. Für größere Dateien zerteilt das GridFS-Tool die Dateien in Fragmente.

Es gibt bestimmte Einschränkungen für Feldnamen in MongoDB. Der Feldname _id ist für die Verwendung als Primärschlüssel reserviert. Er ist einzigartig in der Sammlung und unveränderlich. Feldnamen dürfen nicht mit dem $-Zeichen beginnen (reserviert für Operatoren) und dürfen kein .-Zeichen enthalten (reserviert für den Zugriff auf Felder).



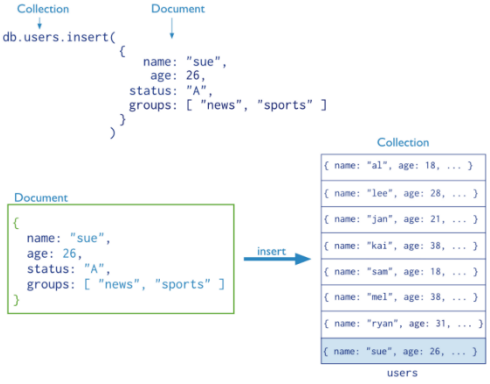
Dokumente in MongoDB haben ein flexibles Schema, und Sammlungen erzwingen keine Datenstruktur. In der Praxis sind die Dokumente jedoch ähnlich. Ein Schlüsselentscheidungspunkt ist die Wahl zwischen Referenzen und eingebetteten Dokumenten, abhängig von den Anforderungen der Anwendung, den Leistungsmerkmalen der Datenbankengine und den Datenabrufmustern.

Die Datenmodellierung in MongoDB kann entweder durch Referenzen (Verknüpfungen von einem Dokument zu einem anderen) oder durch eingebettete Daten (zusammenhängende Daten in einer einzigen Dokumentstruktur) erfolgen. Jeder Ansatz hat seine Vor- und Nachteile und ist abhängig von den spezifischen Anforderungen der Anwendung und den Beziehungen zwischen den Daten.



DATA MODIFIKATION

- CREATE
  - db.collection.**save**({type:'book', item: 'notebook', qty: 49})
- UPDATE
  - db.collection.**update**({type:'book'}, $inc: {qty : -1}, {multi: true})
  - db.collection.**save**({_id: 10, type: 'misc', item: 'placard'}
- DELETE
  - db.collection.**remove**({type: 'food'})



QUERY

Abfragen

- db.collection.**find**({age: { $gt: 18}}).sort({age:1})
  - > 18, sortiert aufsteigend (-1 = absteigend)
- db.collection.**find**({age: { $in: ['food', 'snack']}})
  - in Liste
- db.collection.**find**({type: 'food', price: { $lt: 100}})
  - Food < 100
- collection.find({'grades': {'$elemMatch': {'score': {'$gt': 80, '$lt': 100}}}}, {'name': 1})
  - 'name': 1 bedeutet dass nur der Name angezeigt werden soll (-1 = alles ausser Name)
  - '$elemMatch' → es müssen alle Elemente in einem Array die Bedingung erfüllen
- collection.find({'name': {'$regex': 'sugar', '$options': 'i'}, 'cuisine': 'Bakery'}, {'name': 1})
  - '$options': 'i' → ignoriert Gross- und Kleinschreibung

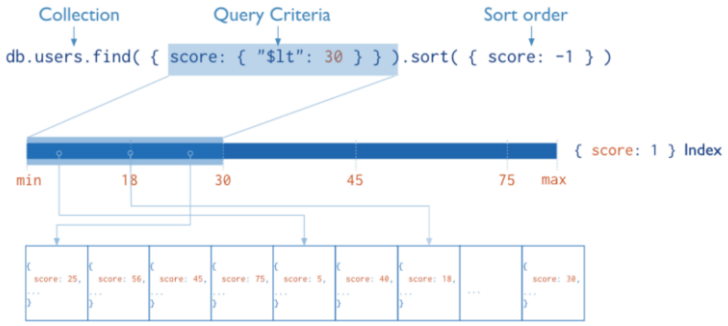
Logische Operatoren

- db.collection.**find**({ **$or**: [{ qty: {$gt: 100}}, {price: {$lt 9.95}}]}}
    - findet Food entweder Anzahl > 100 oder Preis < 9.95
    - $or, $and, $ne (not equal), $in, $nin (not in), $gt (greater), $lt (less)

INDIZES

Indizes in MongoDB dienen dazu, Abfragen zu beschleunigen und die Leistung von bestimmten Operationen zu optimieren. Ohne Indizes muss MongoDB jedes Dokument in einer Sammlung sequentiell durchsuchen, um die Dokumente auszuwählen, die der Abfrageanweisung entsprechen. Mit Indizes kann ein Teil des Datensatzes der Sammlung in einer leicht zu durchsuchenden Form gespeichert werden.

Ein Index ist eine Liste mit IDs, die ein gewisses Attribut aufweisen. In diesem Beispiel auf 'Score'



Es gibt verschiedene Arten von Indizes in MongoDB:

1. **Standard _id Index**: Dieser Primärschlüssel-Index existiert standardmäßig. Jedes Dokument erhält automatisch einen eindeutigen _id Index, wenn dieser nicht spezifiziert wird.
2. **Einfeld-Index**: Dies sind vom Benutzer definierte Indizes auf einem einzelnen Feld eines Dokuments.
3. **Zusammengesetzter Index**: Dies sind vom Benutzer definierte Indizes auf mehreren Feldern.
4. **Mehrschlüssel-Index**: Dieser Index wird verwendet, um den Inhalt in Arrays zu indizieren und erstellt für jedes Element des Arrays einen separaten Indexeintrag.
5. **Geografisches Feld**: Es gibt zwei Arten von geografischen Indizes. 2D-Indizes verwenden planare Geometrie, um Ergebnisse zurückzugeben, und sind für Daten gedacht, die Punkte auf einer zweidimensionalen Ebene repräsentieren. 2Sphere-Indizes verwenden sphärische (erdähnliche) Geometrie und sind für Daten gedacht, die Längen- und Breitengrade repräsentieren.
6. **Textindizes**: Diese Indizes dienen zur Suche nach String-Inhalten in einer Sammlung.
7. **Hash-Indizes**: Diese Indizes indizieren den Hash-Wert eines Feldes und unterstützen nur Gleichheitsabfragen, nicht Bereichsabfragen.

GRAPH-DATENBANKEN

Graph-Datenbanken sind Datenbanken, die speziell entwickelt wurden, um Entitäten und die Beziehungen zwischen diesen Entitäten zu speichern. In einer Graph-Datenbank repräsentieren Knoten (Nodes) Objekte oder Entitäten und haben Eigenschaften wie zum Beispiel einen Namen. Kanten (Edges) stellen Beziehungen zwischen den Knoten dar und haben sowohl eine Richtung als auch Typen oder Labels (wie z.B. "mag", "Freund" etc.). Die Organisation der Knoten erfolgt über Beziehungen, was das Auffinden von interessanten Mustern und die Ausführung von Graphentraversal-Algorithmen ermöglicht.

Im Vergleich zu relationalen Datenbanksystemen (RDBMS) bieten Graph-Datenbanken einige Vorteile, wenn es um die Arbeit mit graphenähnlichen Strukturen geht. In einem RDBMS ist eine graphenähnliche Struktur normalerweise auf einen einzigen Beziehungstyp beschränkt. Die Hinzufügung einer weiteren Beziehung bedeutet oft Änderungen am Schema, Datenverschiebung und so weiter. In Graph-Datenbanken können Beziehungen jedoch dynamisch erstellt oder gelöscht werden, es gibt keine Begrenzung hinsichtlich der Anzahl und Art von Beziehungen.

Im RDBMS modellieren wir den Graphen im Voraus basierend auf der Traversierung, die wir durchführen möchten. Wenn sich die Traversierung ändert, müssen die Daten geändert werden, und es werden normalerweise viele Join-Operationen benötigt. In Graph-Datenbanken hingegen werden Beziehungen nicht zur Abfragezeit berechnet, sondern persistiert. Dies verschiebt den Großteil der Arbeit des Navigierens durch den Graphen auf die Einfügungen und macht die Abfragen so schnell wie möglich.

Ein einfaches Beispiel für die Verwendung einer Graph-Datenbank wäre ein **soziales Netzwerk**, in dem Personen Freundschaften mit anderen Personen haben können. In diesem Szenario könnten Personen als Knoten und Freundschaften als Kanten repräsentiert werden.

RDBMS VS. KEY-VALUE VS. GRAPH DATABASES

RDBMS

- Ideal für strukturierte Daten und komplexe Transaktionen
- Nutzt vordefinierte Schemas
- Starke Konsistenz
- Schwierigkeiten bei der horizontalen Skalierung

Key-Value-Datenbanken

- Schnelle und flexible Datenabfrage durch einfache Schlüssel-Wert-Paare
- Schemalos, einfach zu ändern und zu skalieren
- Begrenzte Abfragefähigkeiten, keine Beziehungen zwischen den Daten

Graph-Datenbanken

- Optimal für komplexe Beziehungen und Verbindungen zwischen Daten
- Flexibles Schema, unterstützt komplexe Abfragen und Analysen
- Weniger geeignet für datenintensive Anwendungen ohne komplexe Verbindungen

PRO / CON

Geeignet für Graphdatenbanken

- Vernetzte Daten (z. B. soziale Netzwerke, Beziehungen zwischen Benutzern)
- Empfehlungssysteme (z. B. Produktempfehlungen basierend auf Benutzerinteraktionen)
- Routing und Standortbasierte Dienste (z. B. Berechnung von kürzesten Pfaden, GPS-Routenplanung)
- Wissensgraphen (z. B. Erfassung und Abfrage von Wissen in einer bestimmten Domäne)
- Betrugsanalyse (z. B. Erkennung verdächtiger Muster und Verbindungen)
- Netzwerkanalyse (z. B. Analyse von Beziehungen in komplexen Netzwerken)

Nicht geeignet für Graphdatenbanken:

- Große Mengen strukturierter Daten ohne komplexe Beziehungen
- Einfache Datenstrukturen mit wenigen Beziehungen

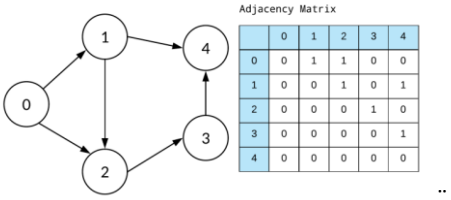
- Transaktionsintensive Anwendungen mit hoher Konsistenzanforderung

Daten: Eine Menge von Entitäten und ihren Beziehungen

- Grundlegende Operationen: Nachbarn eines Knotens finden, überprüfen, ob zwei Knoten durch eine Kante verbunden sind, Aktualisierung der Graphenstruktur
- Effiziente Durchführung von Graphenoperationen erforderlich
- Graph G = (V, E) wird in der Regel wie folgt modelliert:
- Menge von Knoten (Vertices) V
- Menge von Kanten E
- n = |V|, m = |E|

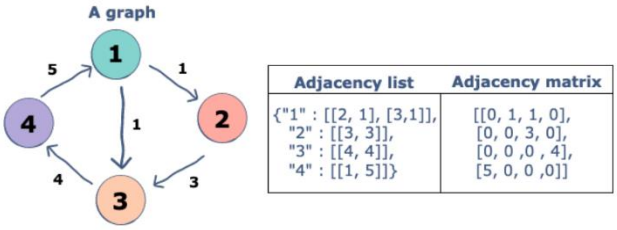
Adjazenzmatrix-Datenstruktur:

- Zweidimensionales Array a mit n x n-Booleschen Werten
- Indizes des Arrays = Knotenidentifikatoren des Graphen
- Der Boolesche Wert Aij der beiden Indizes gibt an, ob die beiden Knoten verbunden sind
- Varianten: Gerichtete Graphen, gewichtete Graphen
- **Vorteile**: Hinzufügen/Entfernen von Kanten, Überprüfung der Verbindung von zwei Knoten
- **Nachteile**: Quadratischer Speicherbedarf im Verhältnis zu n, teures Hinzufügen von Knoten, lineare Laufzeit beim Abrufen aller Nachbarknoten



Adjazenzliste-Datenstruktur:

- Eine Sammlung von Listen, wobei jede Liste die Nachbarn eines Knotens enthält
- Ein Vektor von n Zeigern auf Adjazenzlisten
- Bei ungerichteten Graphen verbindet eine Kante die Knoten i und j => die Liste der Nachbarn von i enthält den Knoten j und vice versa
- Oft komprimiert durch Ausnutzung von Regelmäßigkeiten in Graphen
- **Vorteile**: Abrufen der Nachbarn eines Knotens, günstiges Hinzufügen von Knoten, kompaktere Darstellung von dünn besetzten Matrizen
- **Nachteile**: Überprüfung, ob eine Kante zwischen zwei Knoten besteht, Optimierung durch sortierte Listen möglich (logarithmische Suche und Einfügung)



NEO4J

Neo4j ist eine führende Open-Source-Graphendatenbank, die seit 2007 verfügbar ist. Sie ist in Java geschrieben und plattformübergreifend einsetzbar. Neo4j speichert Daten in Knoten, die durch gerichtete, typisierte Beziehungen miteinander verbunden sind und sowohl Knoten als auch Beziehungen können Eigenschaften haben. Die Daten werden in einem nativen, auf der Festplatte basierenden Speichermodell gespeichert. Neo4j zeichnet sich durch eine intuitive Datenmodellierung, volle ACID-Transaktionen, Skalierbarkeit für große Datenmengen, eine leistungsfähige und menschenlesbare Abfragesprache namens **Cypher** sowie einen effizienten Traversierungsfunktionalität aus. Es kann über eine REST-Schnittstelle oder eine objektorientierte Java-API angesprochen werden. Neo4j ist für hochvernetzte Daten optimiert und bietet eine effiziente Modellierung von Beziehungen. Es eignet sich nicht nur für graphische Anwendungen, sondern auch für die Verwaltung komplexer Datenstrukturen in verschiedenen Anwendungsbereichen.

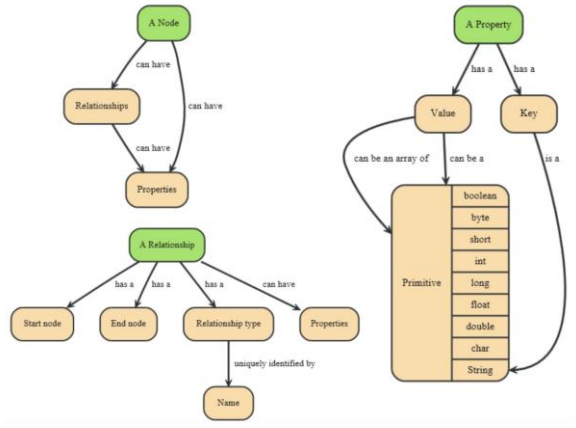
DATENMODELL

Neo4js Datenmodell besteht aus zwei grundlegenden Einheiten: Knoten (Nodes) und Beziehungen (Relationships). Sowohl Knoten als auch Beziehungen können Eigenschaften enthalten, die als Schlüssel-Wert-Paare repräsentiert werden. Die Schlüssel sind Zeichenketten, und die Werte können primitive Typen oder Arrays eines einzigen primitiven Typs sein, wie Zeichenketten, Ganzzahlen oder Ganzzahlen-Arrays.

In Neo4j ist null kein gültiger Eigenschaftswert. Stattdessen kann null durch das Fehlen eines Schlüssels für eine bestimmte Eigenschaft dargestellt werden.

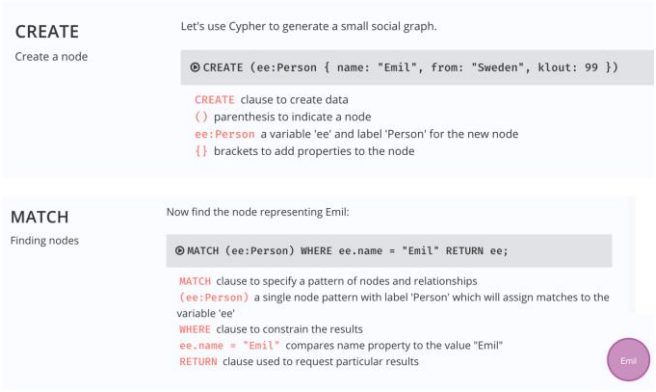
Beziehungen in Neo4j sind gerichtet, das heißt, sie haben eine eingehende und eine ausgehende Kante. Sie können in beide Richtungen gleich gut traversiert werden, sodass es nicht erforderlich ist, beide Richtungen einer Beziehung hinzuzufügen, um die Leistung zu verbessern. Die Richtung von Beziehungen kann von Anwendungen ignoriert werden, wenn sie nicht benötigt wird.

Jede Beziehung in Neo4j hat einen Startknoten und einen Endknoten. Darüber hinaus können Beziehungen rekursiv sein, was bedeutet, dass sie einen Knoten mit sich selbst verbinden können.



Neo4j verwendet die Sprache Cypher als Graphabfragesprache. Cypher ermöglicht das Abfragen und Aktualisieren von Daten in Neo4j. Die Sprache befindet sich noch in der Entwicklung, daher können Syntaxänderungen auftreten.Cypher ist eine deklarative Abfragesprache, bei der wir beschreiben, was wir möchten, und nicht wie wir es erhalten. Im Gegensatz zu traditionellen Traversalsprachen ist es nicht notwendig, Traversals ausdrücklich anzugeben.Eine der Stärken von Cypher ist seine Lesbarkeit. Die Syntax ist von SQL und SPARQL inspiriert und bietet eine intuitiv verständliche Art, Graphabfragen auszudrücken. Es wird

allgemein empfohlen, Cypher immer dann zu verwenden, wenn es möglich ist. Die Dokumentation von Neo4j enthält weitere Informationen zur Verwendung von Cypher.



TRANSAKTIONSMANAGEMENT

Neo4j bietet eine Transaktionsverwaltung mit Unterstützung für ACID-Eigenschaften. Alle Schreiboperationen, die auf dem Graphen durchgeführt werden, müssen in einer Transaktion ausgeführt werden, wobei jede Abfrage als Transaktion betrachtet wird. Es können auch verschachtelte Transaktionen verwendet werden, wobei das Rollback einer verschachtelten Transaktion zum Rollback der gesamten Transaktion führt. Der Ablauf einer Transaktion umfasst das Beginnen der Transaktion, das Ausführen von Schreiboperationen auf dem Graphen, das Markieren der Transaktion als erfolgreich oder nicht erfolgreich und das Beenden der Transaktion. Dabei werden der Speicher und die Sperrungen freigegeben.

Leseoperationen greifen auf den zuletzt bestätigten Wert zu, blockieren jedoch nicht und nehmen keine Sperrungen vor. Nicht-wiederholbare Lesevorgänge können auftreten, wenn eine Zeile zweimal abgerufen wird und sich die Werte zwischen den Lesevorgängen unterscheiden. Es besteht die Möglichkeit, Lese-Sperrungen explizit zu setzen, um ein höheres Maß an Isolation zu erreichen. Alle Änderungen, die in einer Transaktion durchgeführt werden, werden im Speicher gehalten. Bei sehr großen Aktualisierungen müssen diese aufgeteilt werden.

Die Standard-Sperrmechanismen von Neo4j umfassen das Setzen von Schreibsperren auf Knoten, Beziehungen und deren Eigenschaften, wenn diese hinzugefügt, geändert oder entfernt werden. Beim Erstellen oder Löschen eines Knotens wird eine Schreibsperre auf den betreffenden Knoten gesetzt, während beim Erstellen oder Löschen einer Beziehung eine Schreibsperre auf die Beziehung selbst und die beteiligten Knoten gesetzt wird. Deadlocks können auftreten, werden jedoch erkannt und führen zu einer Ausnahmebehandlung.

Insgesamt bietet Neo4j eine robuste Transaktionsverwaltung mit Unterstützung für ACID-Eigenschaften, die die Konsistenz und Integrität der Daten gewährleistet.

INDEXING

Neo4j bietet die Möglichkeit, Indizes zu erstellen, um den Zugriff auf bestimmte Entitäten (Knoten oder Beziehungen) zu beschleunigen. Ein Index hat einen eindeutigen, vom Benutzer festgelegten Namen und kann beliebig viele Schlüssel-Wert-Paare mit beliebig vielen Entitäten verknüpfen. Es ist möglich, einen Knoten oder eine Beziehung mit mehreren Schlüssel-Wert-Paaren zu indizieren, die denselben Schlüssel haben. Wenn ein neuer Wert für einen bereits indizierten Schlüssel festgelegt wird, muss der alte Wert gelöscht werden, um Kollisionen zu vermeiden. Standardmäßig gibt es einen automatischen Index für Knoten und einen für Beziehungen. Dieser Index kann verwendet werden, um die Eigenschaftswerte von Knoten und Beziehungen zu indizieren. Die automatische Indexierung ist standardmäßig deaktiviert, kann jedoch für bestimmte Eigenschaften von Knoten und Beziehungen aktiviert werden. Der Index kann wie jeder andere Index abgefragt werden, um effiziente Suchvorgänge durchzuführen. Indizes in Neo4j dienen dazu, den Zugriff auf bestimmte

Daten zu beschleunigen und die Effizienz von Suchoperationen zu verbessern. Sie bieten eine Möglichkeit, gezielte Abfragen auf Basis von Schlüssel-Wert-Paaren durchzuführen und so schnell auf relevante Daten zuzugreifen.

VERFÜGBARKEIT

Neo4j bietet die Möglichkeit der Hochverfügbarkeit, um eine fehlertolerante Datenbankarchitektur zu ermöglichen. Dabei können mehrere Neo4j-Slave-Datenbanken als exakte Replikate einer einzelnen Neo4j-Master-Datenbank konfiguriert werden. Dies ermöglicht eine horizontale Skalierung für eine vorwiegend lesende Architektur, bei der das System eine größere Lesebelastung bewältigen kann als eine einzelne Neo4j-Datenbankinstanz.

Transaktionen bleiben weiterhin atomar, konsistent und dauerhaft, werden jedoch schließlich auf andere Slaves propagiert. Der Übergang von einem einzelnen Rechner zu einem mehrere Rechner umfassenden Betrieb ist einfach und erfordert keine Änderungen an bestehenden Anwendungen. Es genügt, den Wechsel von der Verwendung der "GraphDatabaseFactory" zur "HighlyAvailableGraphDatabaseFactory" vorzunehmen, da beide die gleiche Schnittstelle implementieren.

In einer hochverfügbaren Neo4j-Umgebung gibt es immer einen Master und null oder mehr Slaves. Schreiboperationen werden auf dem Master durchgeführt und schließlich an die Slaves propagiert. Alle anderen ACID-Eigenschaften bleiben unverändert. Wenn eine Schreiboperation auf einem Slave durchgeführt wird, erfolgt eine sofortige Synchronisation mit dem Master. Der Slave muss stets auf dem aktuellen Stand des Masters sein, daher muss die Operation auf beiden durchgeführt werden.

Im Falle eines Ausfalls wird ein Slave von den anderen Knoten im Cluster erkannt. Wenn der Master ausfällt, wird ein Slave als neuer Master gewählt. Im Wiederherstellungsfall synchronisiert sich ein Slave mit dem Cluster, während der alte Master zu einem Slave wird. Dieser Mechanismus gewährleistet eine kontinuierliche Verfügbarkeit und Ausfallsicherheit des Neo4j-Systems.

CYPHER

`CREATE` clause to create data
`()` parenthesis to indicate a node
`ee:Person` a variable 'ee' and label 'Person' for the new node
`{}` brackets to add properties to the node

`MATCH` clause to specify a pattern of nodes and relationships
`(ee:Person)` a single node pattern with label 'Person' which will assign matches to variable 'ee'
`WHERE` clause to constrain the results
`ee.name = "Emil"` compares name property to the value "Emil"
`RETURN` clause used to request particular results

```
CREATE (:Movie {title: 'Forrest Gump', released: 1994})

MATCH (m:Movie)
WHERE m.title = 'Forrest Gump'
SET m:OlderMovie,
m.released = 1994,
m.lengthInMinutes = 142

MATCH (p:Person)-[:ACTED_IN]->(m:Movie)<-[:RATED]-(c:Person)
WITH p, m, COLLECT(c) as co
WHERE p.name = 'Tom Cruise' AND SIZE(co) > 100
OPTIONAL MATCH (p2:Person)-[:WATCHED]->(m)
RETURN m.title, SIZE(co) AS rewiew_count, SIZE(p2) AS watched
```