7.5 Anwendung von B+-Bäumen in NO-SQL-Datenbankystemen

- **Defizite relationaler DBMS**
  - Performance in vielen Anwendungen nicht akzeptabel.
  - Potenzial der Verteilung wird nicht ausgenutzt.
  - Restriktives Datenmodell
  - Komplexe und überfrachtete Systemarchitektur
- **Seit Anfang 2000 Entwicklung von NO-SQL-Systemen.**
  - Einfachere Systeme mit einer einfachen Benutzerschnittstelle
    - Aufgabe von unnötigen Ballast
  - Keine Beschränkung auf relationales Datenmodell
  - Performance durch gute Skalierbarkeit.

Einordnung der Systeme

- **Key-Value Stores**
  - Einschränkung auf Maps, die Keys auf Werte abbilden.
- **Document Stores**
  - Keine Tabellen wie bei relationalen Systemen, sondern unstrukturierte oder semi-strukturierte Textdokumente.
- **Search Engines**
  - Spezielle Variante von Document Stores, die eine Suchfunktionalität wie bei Suchmaschinen bieten.
- **Graph DBMS**
- **Time Series DBMS**
  - Spezialsysteme für Zeitserien
- **RDF Stores**
- **…**

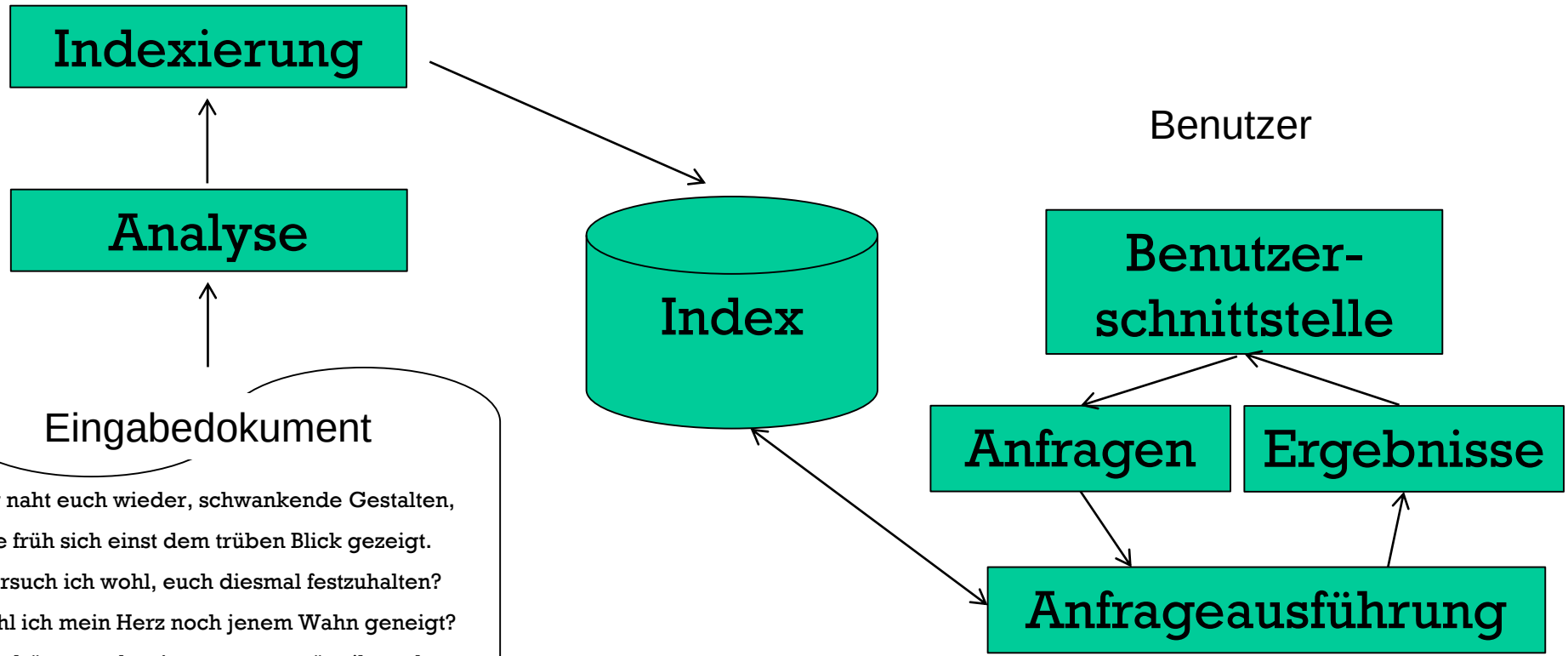
https://db-engines.com/de/ranking

7.5.1 Search-Engine Lucene

- Apache Lucene
  - Java-Bibliothek mit Funktionalität für eine Search-Engine
  - Entwicklung seit 2001, aktuelle Version 8.0.0 von 2019

- Verwendung von Lucene in Systemen
  - Apache Solr
  - ElasticSearch
    - Suchmaschine zur Verwaltung von Dokumenten
    - Hochverfügbarkeit und Lastverteilung
      - Verteilung der Daten in einem Rechnernetzwerk
    - Direkte Unterstützung von JSON

Prinzipielle Vorgehensweise



Dokumente und Felder

- Ein **Dokument** besteht aus einem oder mehreren Feldern.
- Jedes **Feld** hat einen Namen und einen Wert.
  - Der Wert entspricht einer Folge von Wörtern (**Termen**).
    - Zwei gleiche Terme in verschiedenen Feldern werden als unterschiedlich angesehen!
  - Beispiel (in Java):

```
Document doc = new Document();
doc.add(new TextField("title", "Faust", Field.Store.YES));
doc.add(new TextField("body", "Ihr naht …", Field.Store.YES));
w.addDocument(doc);
```

- Ein **Index** wird über die Menge von Termen angelegt.

Invertierter Index

- **Analyse** zerlegt die Eingabe in Terme
  - Optional werden Stoppwörter entfernt.
  - Normierung der Terme
- Aufbau eines Index über Terme einer Dokumentenkollektion
  - **Zu einem Term wird die Liste der Dokumente verwaltet**, in denen die Terme vorkommen.
    - Optional: Position des Terms im Dokument und die Häufigkeit des Auftretens.
- **Beispiel (in Java):**

```
StandardAnalyzer analyzer = new StandardAnalyzer();
Directory index = new RAMDirectory();
IndexWriterConfig config = new IndexWriterConfig(analyzer);
IndexWriter w = new IndexWriter(index, config);
addDoc(w, "Faust", "Ihr naht …");
```

Segmente

- **Segmente**
  - Lucene teilt einen großen Index in mehrere Segmente auf.
    - Jedes Segment ist ein eigener Index.
  - Neue Dokumente werden in eigenes Segment eingefügt.
  - Verschmelzen von Segmenten bei Bedarf zu größeren Segmenten.

- **Dadurch ergeben sich Performance-Vorteile beim Einfügen.**
  - Diese Vorgehensweise wird auch für gewöhnliche B+-Bäume angewendet. Man spricht dann von **Log-Structured-Merge Bäumen (LSM trees)**.

    https://en.wikipedia.org/wiki/Log-structured_merge-tree

Implementierung

- Suchanfragen nach Termen durch Index
  - Verwaltung der Terme durch einen B+-Baum
    - Für jeden Term gibt es eine Liste mit der Kennung des Dokuments
  - Unterstützung von AND, OR und NOT durch Mengenoperationen
- **Beispiel**

Dokumente:	Termlisten:	Anfragen:
d1: "A C B A"	A: d1,d3	A and B = {d1,d3} ∩ {d1,d2,d3}
d2: "B D D"	B: d1,d2,d3	= {d1,d3}
d3: "A B B E"	C: d1	
	D: d2	
	E: d3	

Anfragen

- **Formulierung einer Anfrage durch den Benutzer**
  - Gegeben: Menge von Suchtermen
  - Gesucht: Alle Dokumente mit diesen Suchtermen.
    - Ggf. Dokumente mit „vielen“ dieser Terme.

- **Problem**
  - Bei der Suche qualifizieren sich zu viele Dokumente für die Anfrage.
  - Bei der Suche gibt es kein Dokument mit allen Suchtermen.
    - Approximative Ergebnisse sollen dann zurückgeliefert werden.

Vektormodell

- **Motivation**
  - Statt viele Ergebnisse auszugeben, sollen nur die k relevantesten berechnet werden.
    - Bei Bedarf können weitere Ergebnisse nachgeladen werden.
- **Verwendung des Vektormodells**
  - Jedes Dokument und Anfrage wird auf einen Vektor abgebildet.
    - Jeder Term repräsentiert eine Dimension des Vektors.

Dokumente:	Vektoren:
d1: "A C B A"	d1:[2 1 1 0 0]
d2: "B D D"	d2:[0 1 0 2 0]
d3: "A B B E"	d3:[1 2 0 0 1]
	A B C D E

Ranking

- **Ähnlichkeitsmaß für zwei Vektoren d1 und d2 der Dimension n.**
  - $d1 = (d1_1, d1_2, \ldots, d1_n)$, $d2 = (d2_1, d2_2, \ldots, d2_n)$

$$\mathrm{sim}(d1, d2) = \frac{d1 \cdot d2}{\|d1\| \cdot \|d2\|} = \frac{\sum_{i=1}^{n} d1_i \cdot d2_i}{\sqrt{\sum_{i=1}^{n} {d1_i}^2} \cdot \sqrt{\sum_{i=1}^{n} {d2_i}^2}}$$

- **Beispiel: sim(d1,d3)**
  - d1= (2,1, 1, 0, 0), d3 = (1, 2, 0, 0, 1)
  - ||d1|| =
  - ||d3|| =
  - sim (d1,d3) =

Anfragesprache (1)

- **Definition der Anfragesprache**
  - https://lucene.apache.org/core/2_9_4/queryparsersyntax.html
- **Keyword matching**
  - Suche nach "foo bar" in dem Feld title
    title:"foo bar“
  - Suche nach "foo bar" in dem Feld title AND "quick fox" in dem Feld body OR "fox" in dem Feld title.
    (title:"foo bar" AND body:"quick fox") OR title:fox
  - Suche nach "foo" AND NOT "bar" in dem Feld title
    title:foo -title:bar

Anfragesprache (2)

- **Bereichssuche**
  - Suche nach einen Datum in einem Zeitintervall (im Feld mod_date)
    mod_date:[20160101 TO 20170701]

- **Wildcard Suche**
  - Suche nach einem Wort mit dem Anfang "foo" in dem Feld title

    title:foo*

- **Proximity matching**
  - Suche nach "foo" AND "bar" mit maximal 4 anderen Wörteren dazwischen.
    "foo bar"~4

7.5.2 ElasticSearch

- ElasticSearch ist ein hoch-skalierbares System für die Volltextsuche und Analyse.
  - Skalierbarkeit bedeutet, dass es verteilt in einem Rechnernetzwerk arbeitet und den Verteilungsgrad je nach Bedarf dynamisch anpasst.
  - Aktueller Release: 7.1.1 unter Apache 2.0 Lizenz
- Merkmale
  - REST-basierte Such- und Analysefunktionalität
  - Verwendung von JSON als Datenmodell
  - Auslieferung im sogenannten ELK-Stack
    - ELK = Elasticsearch/Logstash/Kibana)
    - Datenverarbeitung, Analyse, Visualisierung …
  - Verwendung von Lucene als Search Engine

Shards

- **Elastic kann einen Index über Knoten verteilen.**
  - Ein Shard ist ein lokaler Teil des Index auf einem Knoten, der mit Lucene verwaltet wird.
- **Vorteile von Shards**
  - Horizontalle Aufteilung des Index (die lokal in Speicher passen).
  - Verteilte Anfrageverarbeitung zur Beschleunigung von Anfragen.
- **Verfügbarkeit und Replikation**
  - Ausfall eines Shards soll nicht Ausfall des gesamten systems verursachen. ➔ Replikation von Shards
  - Replikate können die parallele Verarbeitung verbessern.

Verteilung der Dokumente

- **Für die Verteilung der Dokument wird ein ausgezeichneter Typ routing verwendet.**
  - Per Default wird routing auf das Id-Feld des Dokuments gesetzt.

- **Formel für die Verteilung**

  shard = hash(routing) % number_of_primary_shards

- **Konsequenz**
  - Die Anzahl der Shards kann zur Laufzeit nicht mehr verändert werden.

Suchen in Elastic

- **Anfragen in Elastic mit URI-Requests !**
- **Beispiel**
  - Mit der Anfrage "get /_search" werden alle Dokumente in der Datenbank geliefert.
  - Als Schema für die Ausgabe wird JSON benutzt.

```
{
"hits" : {
        "total" :       14,
        "hits" : [
         {
        "_index":   "us",
        "_type":    "tweet",
        "_id":      "7",
        "_score":   1,
        "_source": {
                                "date":    "2014-09-17",
                                "name":    "John Smith",
                                 "tweet":   "The Query DSL is really powerful and flexible",
                                 "user_id": 2

        }

        ...
```

Gesamtanzahl aller Ergebnisse

Typ zur Indexgenerierung

Score des Ergebnisses

Das erste Ergebnis

Suchen in Elastic (2)

```
            ],
            "max_score" :   1
        },
        "took" :          4,
        "_shards" : {
            "failed" :      0,
            "successful" :  10,
            "total" :       10
        },
        "timed_out" :     false
}
```

Weiter hinten im Ergebnis gibt es noch Metainformationen zum Ergebnis:

- Größter Score
- Beanspruchte Zeit
- Anzahl der Shards
- Wurde die Anfrage durch ein Timeout beendet.
  - Die Timeout-Schranke kann durch den Benutzer gesetzt werden.

Anfragen über mehrere Indexe und Typen

- **Solange Benutzer nicht die Anfrage einschränken, wird die Suche über alle Indexe und Typen gestellt.**
- **Beispiele von Anfragen**
  - get /idx/_search
    - Alle Dokumente in dem Index idx werden gesucht.
  - get /idx1,idx2/_search
    - Suche über zwei Indexe idx1 und idx2
  - get /i*/_search
    - Suche in allen Indexen, die mit dem Buchstaben i beginnen.
  - get /idx/user/_search
    - Suche in Index idx nach allen Dokumenten vom Typ user.
  - get /_all/user,tweet/_search
    - Suche Dokumente Typen user und tweet in allen Indexen

Inhaltsbasierte Anfragen

- **Selektionsanfragen können unter Verwendung eines Anfragestrings abgesetzt werden.**
- **Beispiele:**
  - get /_all/tweet/_search?q=tweet:dbs1
    - Suche in allen Indexen nach Dokumenten vom Typ tweet, bei dem das Wort dbs1 vorkommt.
  - Leider werden Anfragen durch die notwendige Percent-Kodierung ziemlich unleserlich.
    - Suche wir nach +name:seeger +tweet:dbs1 lautet die Anfrage:

    get /_search?q=%2Bname%3Aseeger+%2Btweet%3Adbs1

Das besondere Feld _all

- **Im Wesentlichen können in Elastic durch JSON die Felder im Dokument beschrieben werden.**
  - Diese Felder können wie in relationalen System ganz unterschiedliche Typen haben.
  - Zusätzlich kann der Typ Text verwendet werden.
- **Zusätzlich unterstützt Elastic noch ein Feld _all.**
  - _all entspricht der textuelle Verknüpfung aller Felder.
    - Man kann dies bei der Definition explizit verbieten.
  - Die Anfrage ohne Feldangabe werden immer als Freitextsuche auf dem Feld _all ausgeführt.
  - Beispiel
    - get/_search?q=kemper+neumann+2016

Exakte Suche oder Volltextsuche?

- **Elastic unterstützt in seiner Anfragesprache beides.**
- **Beispiel**
  - get /_search?q=2014 # 12 results
    - Liefert alle Dokumente, die in _all den Term 2014 enthalten.
  - get /_search?q=date:2014 # 0 results !
    - Liefert alle Dokumente, die im Feld date den Wert 2014 haben.
    - Dies ist nicht der Fall, da alle Werte im Format yyyy-mm-dd vorliegen.

Mapping

- **Elastic bietet die Möglichkeit wie in relationalen Systeme die Daten über Felder zu strukturieren.**
  - Die Definition dieser Felder nennt man auch Mapping.
  - Ein Mapping kann mit dem Befehl

    get /idx/_mapping/tweet

    angefragt werden.
- **Es gibt noch Möglichkeiten für die Definition von Mappings.**
  - Siehe https://www.elastic.co/guide/en/elasticsearch
  - Insbesondere können Mappings dynamisch um weitere Felder erweitert werden.
  - Verschachtelte Felder definiert werden.

Zusammenfassung

- **Elastic bietet einen Dokumenten-basierten Zugang für das Datenmanagement.**
  - Flexible Schemadefinition
  - Verwendung invertierter Indexe verteilt über Knoten
  - Neuartige Anfragesprachen mit URI-Requests.
    - Alternative: Verwendung von JSON (Query DSL)

- **Offene Fragen**
  - Wie kann bei Änderungen die Konsistenz sichergestellt werden?
  - Wie kann man Daten anfragen, die nicht über einen Index zugreifbar sind?