Big Data: Neue Entwicklungen im Datenbankbereich

- **Semantic Web: RDF**
- **Datenströme**
- **Information Retrieval**
- **Graph Exploration**
- **Map Reduce: Massiv parallele Verarbeitung**
- **Peer to Peer Informationssysteme**
- **Key Value Stores / NoSQL**
- **Multi-Tenancy/Cloud-Datenbanken**

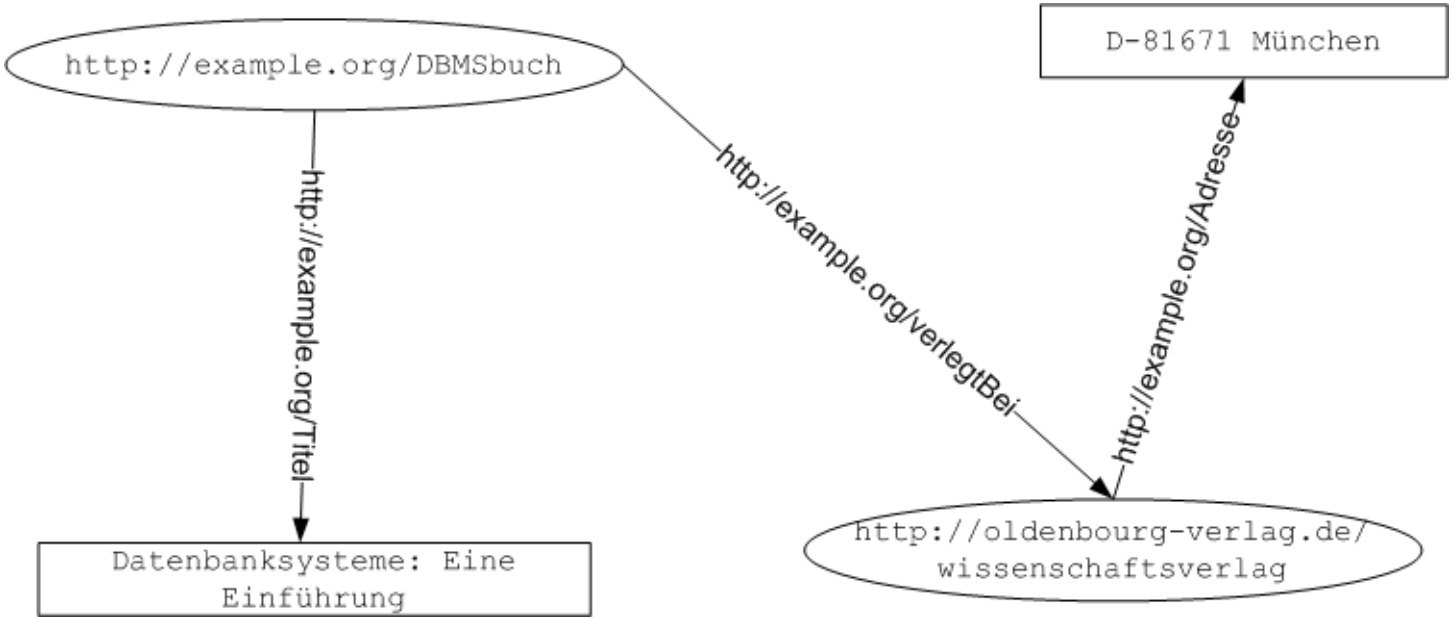
Semantic Web: Resource Description Framework (RDF)

Triple-Datenmodell

- (Subjekt, Prädikat, Objekt)

Meist graphische Visualisierung
- Subjekte und Objekte sind Knoten
- Prädikate sind gerichtete Kanten
  - Von Subjekt-Knoten nach Objekt Knoten

Beispiel-RDF-Graph



- Das Buch mit der Kennung ⟨`http://example.org/DBMSbuch`⟩ wird ⟨`http://example.org/verlegtBei`⟩ dem Verlag ⟨`http://oldenbourg-verlag.de/wissenschaftsverlag`⟩.
- Der ⟨`http://oldenbourg-verlag.de/wissenschaftsverlag`⟩ hat als ⟨`http://example.org/Adresse`⟩ das Literal "`D-81671 München`".
- Das ⟨`http://example.org/DBMSbuch`⟩ hat als ⟨`http://example.org/Titel`⟩ das Literal "`Datenbanksysteme: Eine Einführung`".

Somit wird klar, dass das erste Element des Tripels das Subjekt einer Aussage, das zweite das Prädikat und das dritte Element das Objekt darstellt.

Es gibt mehrere leicht unterschiedliche, textuelle RDF-Tripeldarstellungen, die man unter den Namen *N3*, *N-Triples* oder *Turtle* findet. In unserem Fall sieht die Beispieldatenbank in der *Turtle*-Notation wie folgt aus:

```
<http://example.org/DBMSbuch> <http://example.org/verlegtBei>
                    <http://oldenbourg-verlag.de/wissenschaftsverlag>.
<http://oldenbourg-verlag.de/wissenschaftsverlag>
                    <http://example.org/Adresse> "D-81671 München".
<http://example.org/DBMSbuch> <http://example.org/Titel>
                    "Datenbanksysteme: Eine Einführung".
```

Kurzform

```
@prefix ex: <http://example.org>.
@prefix ol: <http://oldenbourg-verlag.de>.

ex:DBMSbuch ex:verlegtBei ol:wissenschaftsverlag.
ol:wissenschaftsverlag ex:Adresse "D-81671 München".
ex:DBMSbuch ex:Titel "Datenbanksysteme: Eine Einführung".
```

```
ex:DBMSbuch ex:verlegtBei ol:wissenschaftsverlag;
            ex:Titel "Datenbanksysteme: Eine Einführung".
ol:wissenschaftsverlag ex:Adresse "D-81671 München".
```

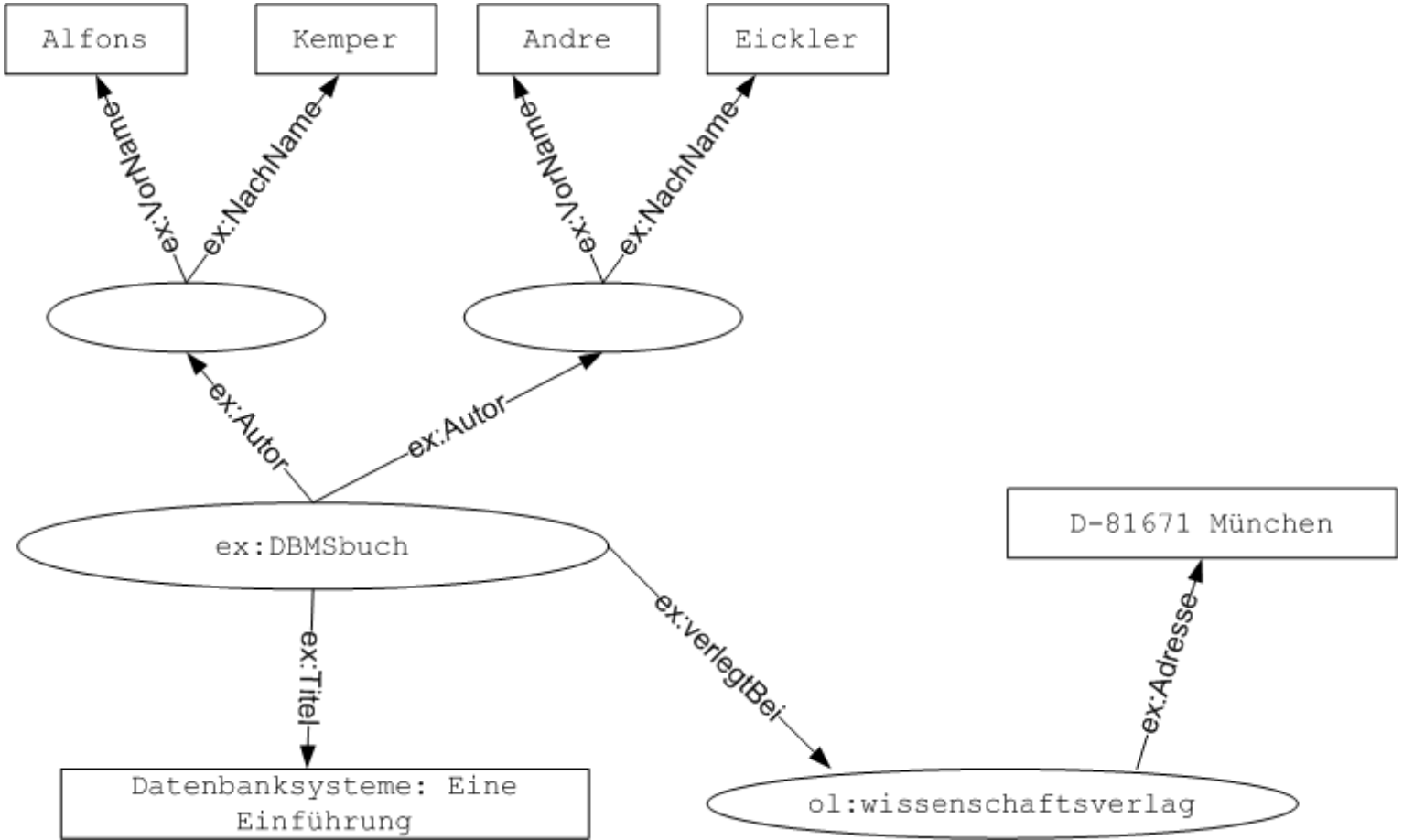
Man beachte, dass man jetzt ein Semikolon verwendet, um anzugeben, dass das vorherige Subjekt übernommen wird. Man kann mengenwertige Beziehungen auch noch weiter kompaktifizieren, wenn man Cluster gleicher Subjekte und Prädikate bildet. Beispielsweise:

```
ex:DBMSbuch ex:AutorNachName "Kemper" ,
                             "Eickler".
```

Namenlose Knoten

```
@prefix ex: <http://example.org>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

ex:DBMSbuch ex:Autor _:k.
_:k ex:NachName "Kemper"^^xsd:string.
_:k ex:VorName "Alfons"^^xsd:string.

ex:DBMSbuch ex:Autor _:e.
_:e ex:NachName "Eickler"^^xsd:string.
_:e ex:VorName "Andre"^^xsd:string.
```

Graph mit unbenannten Knoten



SPARQL: Die RDF Anfragesprache

```
SELECT ?Var1 ?Var2 … $VarN
WHERE {Muster1. Muster2. … MusterM. }
```

```
PREFIX ex: <http://www.example.org>

SELECT ?AutorenDesOldenbourgVerlags WHERE
{  ?buch ex:Autor ?a.
   ?a ex:NachName ?AutorenDesOldenbourgVerlags.
   ?buch ex:verlegtBei <http://oldenbourg-verlag.de/wissenschaftsverlag>.
}
```

SPARQL: Die RDF Anfragesprache

```
PREFIX ex: <http://www.example.org>

SELECT ?KempersBuecherTitel WHERE
{   ?KempersBuecher ex:Autor ?k.
    ?k ex:NachName "Kemper".
    ?KempersBuecher ex:Titel ?KempersBuecherTitel.
}
```

SPARQL: Die RDF Anfragesprache

Union

```
PREFIX ex: <http://www.example.org>

SELECT ?KempersOderEicklersBuecherTitel WHERE
{ {   ?KempersBuecher ex:Autor ?k.
      ?k ex:NachName "Kemper".
      ?KempersBuecher ex:Titel ?KempersOderEicklersBuecherTitel.
  } UNION
  {   ?EicklersBuecher ex:Autor ?k.
      ?k ex:NachName "Eickler".
      ?EicklersBuecher ex:Titel ?KempersOderEicklersBuecherTitel.
} }
```

SPARQL: Die RDF Anfragesprache

optional

```
PREFIX ex: <http://www.example.org>

SELECT ?KempersBuecherTitel ?KempersBuecherISBN WHERE
{   ?KempersBuecher ex:Autor ?k.
    ?k ex:NachName "Kemper".
    ?KempersBuecher ex:Titel ?KempersBuecherTitel.
    OPTIONAL {  ?KempersBuecher ex:hatISBN ?KempersBuecherISBN  }
}
```

SPARQL: Die RDF Anfragesprache

filter

```
PREFIX ex: <http://www.example.org>

SELECT ?KempersBuecherTitel ?auflagenNr WHERE
{   ?KempersBuecher ex:Autor ?k.
    ?k ex:NachName "Kemper".
    ?KempersBuecher ex:Titel ?KempersBuecherTitel.
    ?KempersBuecher ex:Auflage ?auflagenNr.
    FILTER ( ?auflagenNr > 7 )
}
```

SPARQL: Die RDF Anfragesprache

count-Aggregation

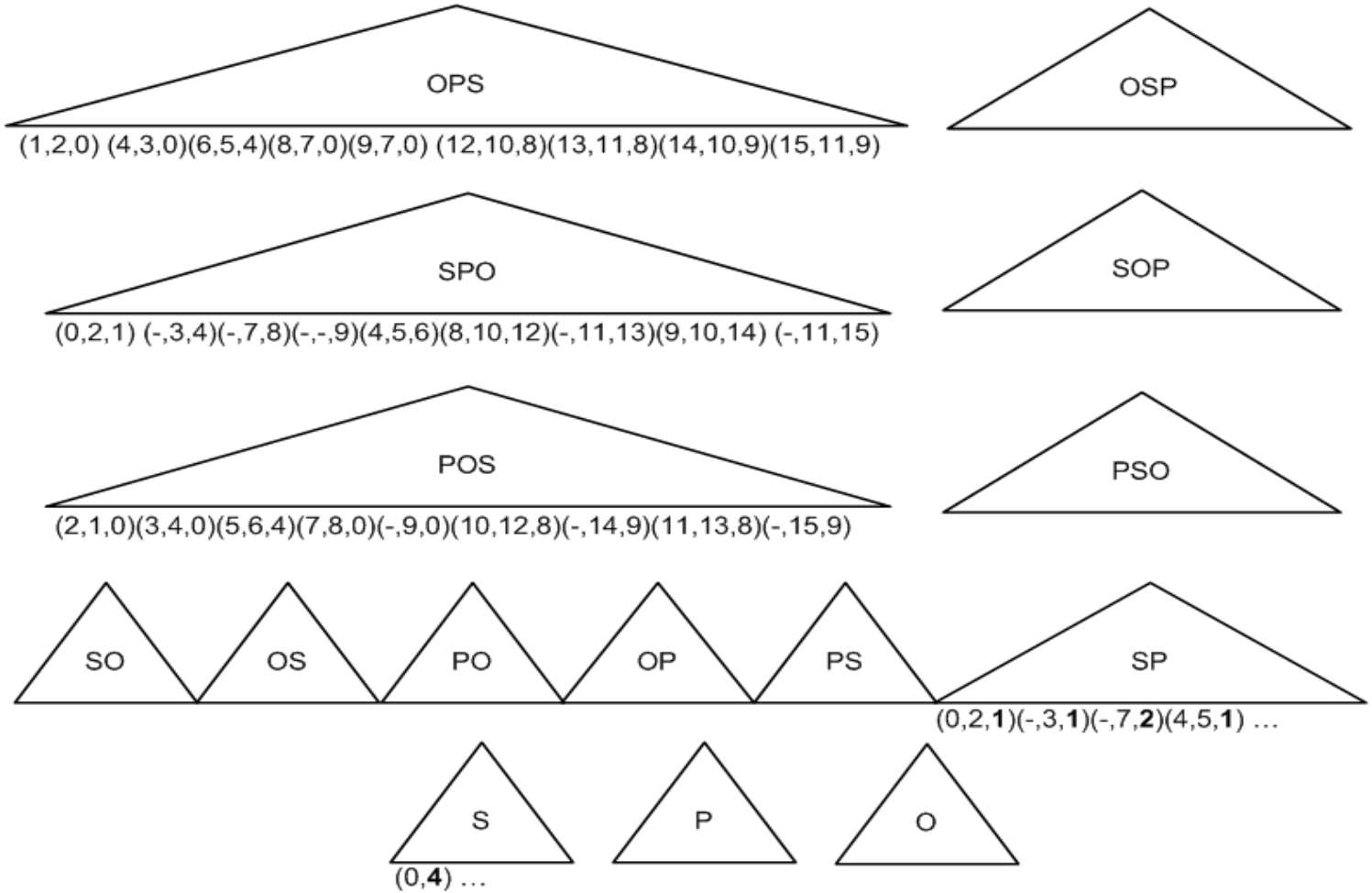
```
PREFIX ex: <http://www.example.org>

SELECT COUNT ?verlag WHERE
{
    ?buch ex:verlegtBei ?verlag.
}
```

Für SQL-affine Leser ist diese Anfrage etwas gewöhnungsbedürftig, da man ja in der Tat die Bücher zählen will. Die SPARQL-Formulierung zielt aber darauf ab, die Anzahl der Vorkommnisse des Musters **„?buch ex:verlegtBei ?verlag“** für jeden **?verlag** zu zählen.

Implementierung einer RDF-Datenbank: RDF-3X

Dictionary	
Literal/URI	Code
<http://www.example.org/DBMSbuch>	0
Datenbanksysteme: Eine Einführung	1
<http://www.example.org/Titel>	2
<http://www.example.org/verlegtBei>	3
<http://oldenbourg-verlag.de/wissenschaftsverlag>	4
<http://www.example.org/Adresse>	5
D-81671 München	6
<http://www.example.org/Autor>	7
<dummy1>	8
<dummy2>	9
<http://www.example.org/VorName>	10
<http://www.example.org/NachName>	11
Alfons	12
Kemper	13
Andre	14
Eickler	15

B-Bäume … so viele wie möglich



Kompressionstechnik: Dictionary und Präfix

Jedes Tripel (s,p,o) wird also genau 6 mal repliziert abgelegt -- allerdings in permutierter Subjekt/Prädikat/Objekt-Reihenfolge, nämlich

- (p,s,o), (s,p,o), (p,o,s), (o,s,p), (s,o,p) und (o,p,s).

Zusätzlich gibt es noch die sogenannten aggregierten Indexe, die die Anzahl der Vorkommen des jeweiligen Musters repräsentieren.

Zum Beispiel bedeutet der Eintrag (s,o,7), dass das Subjekt s siebenmal mit dem Objekt o in einer Beziehung steht -- mit beliebigem Prädikat.

Das Speichervolumen wird dadurch (dramatisch) reduziert, dass man in den Blättern der Bäume eine Präfix-Komprimierung durchführt. Z.B. wird in dem zweiten Eintrag des SPO-Baums das Subjekt 0 weggelassen, da es identisch zum ersten Eintrag ist. In dem vierten Eintrag kann sogar die Subjekt- und die Prädikat-Kennung weggelassen werden, da beide identisch zum dritten Eintrag sind

Kompressionstechnik: Dictionary und Präfix

Es werden aber nicht nur gleiche Präfixe weggelassen; zusätzlich wird auch anstatt des jeweiligen Codes nur die Differenz zum Code des Vorgänger-Tripels gespeichert. Der letzte Eintrag im SPO-Baum würde demnach als (-,1,1) gespeichert, da er in der ersten Komponente identisch zum Vorgänger-Tripel ist, in der zweiten und dritten Komponente ist die Differenz zum Vorgänger-Tripel jeweils 1.

Diese Kompression ist sehr effektiv, da die Tripel in den Blattknoten ja fortlaufend sortiert sind und sich deshalb immer nur geringfügig vom Vorgänger-Tripel unterscheiden.

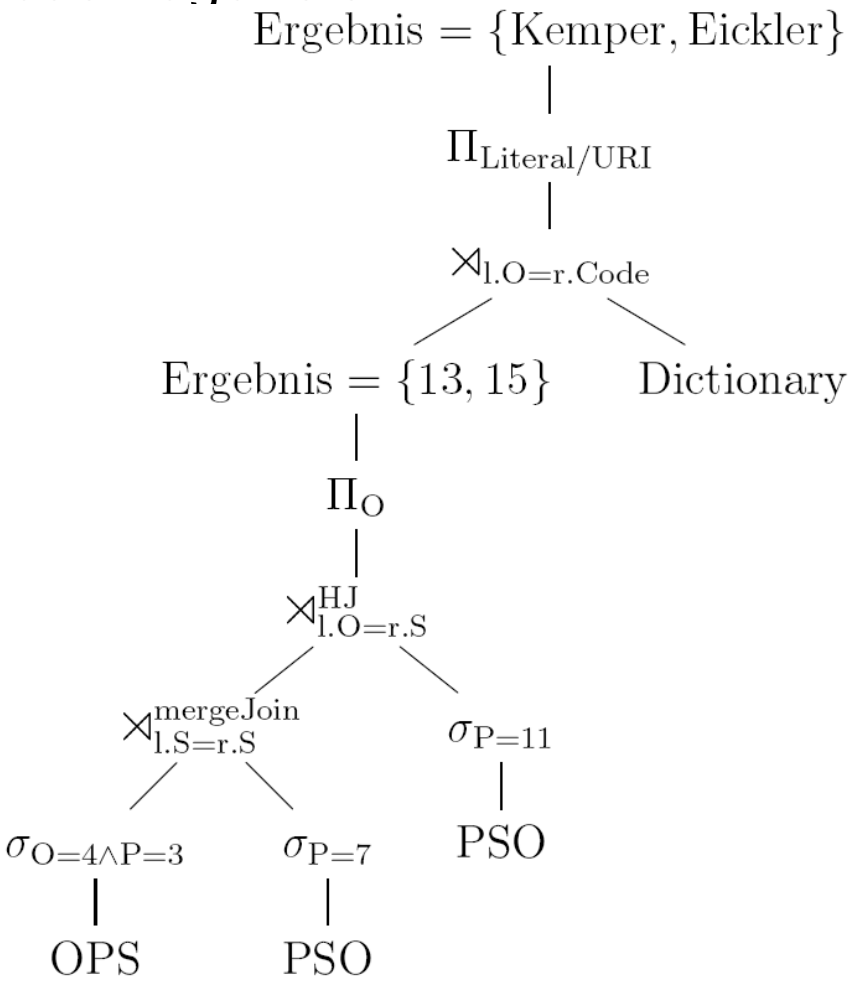
Als Anker für diese Differenz-Kompression wird auf jeder Blatt-Seite immer nur ein vollständiges Tripel, nämlich das Erste, gespeichert.

Anfrageauswertung

```
PREFIX ex: <http://www.example.org>

SELECT REDUCED ?AutorenDesOldenbourgVerlags WHERE
{ ?buch ex:Autor ?a.
  ?a ex:NachName ?AutorenDesOldenbourgVerlags.
  ?buch ex:verlegtBei <http://oldenbourg-verlag.de/wissenschaftsverlag>.
}
```


```
SELECT REDUCED ?AutorenDesOldenbourgVerlags WHERE
{   ?buch 7 ?a.
    ?a 11 ?AutorenDesOldenbourgVerlags.
    ?buch 3 4.
}
```

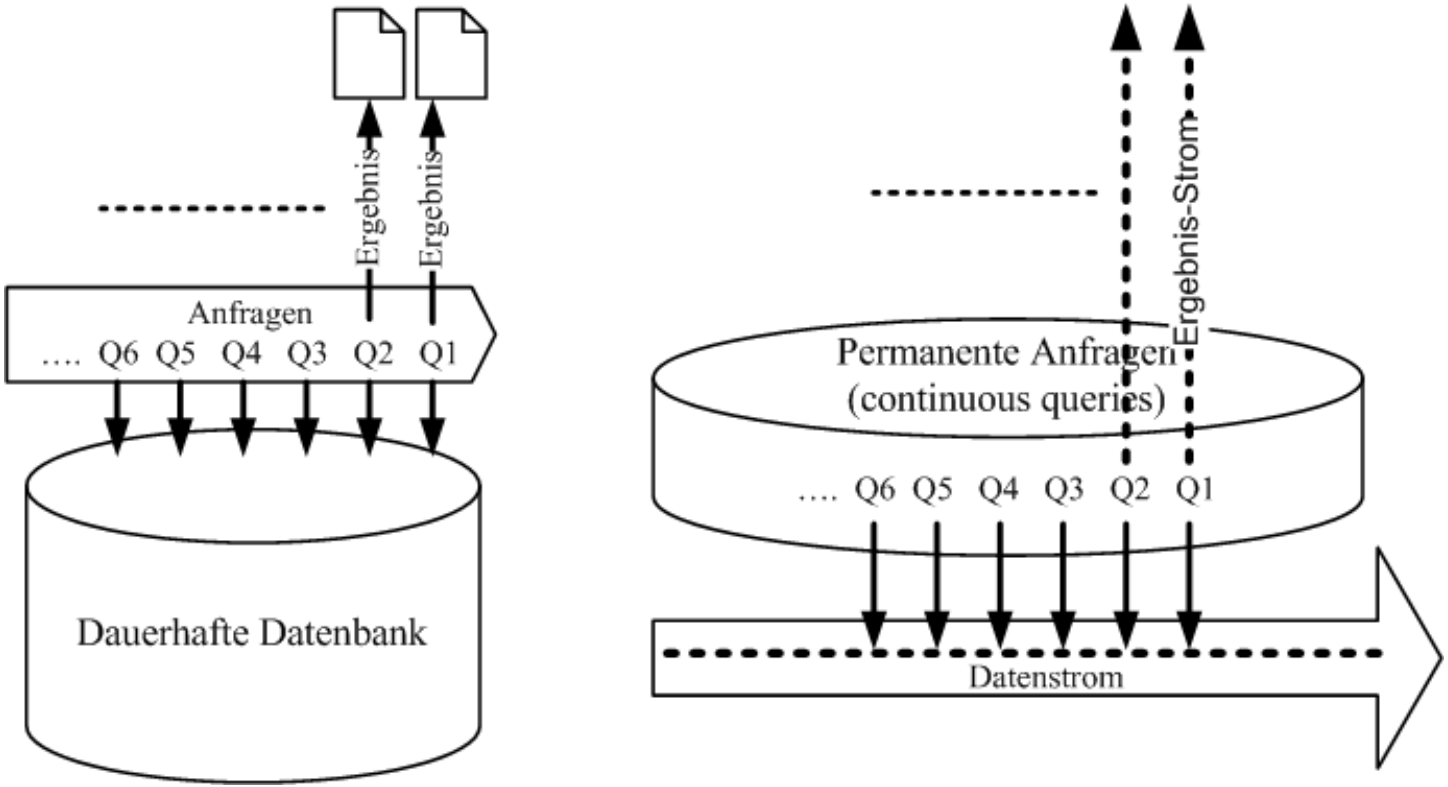
Merge-Joins … so weit das Auge reicht …



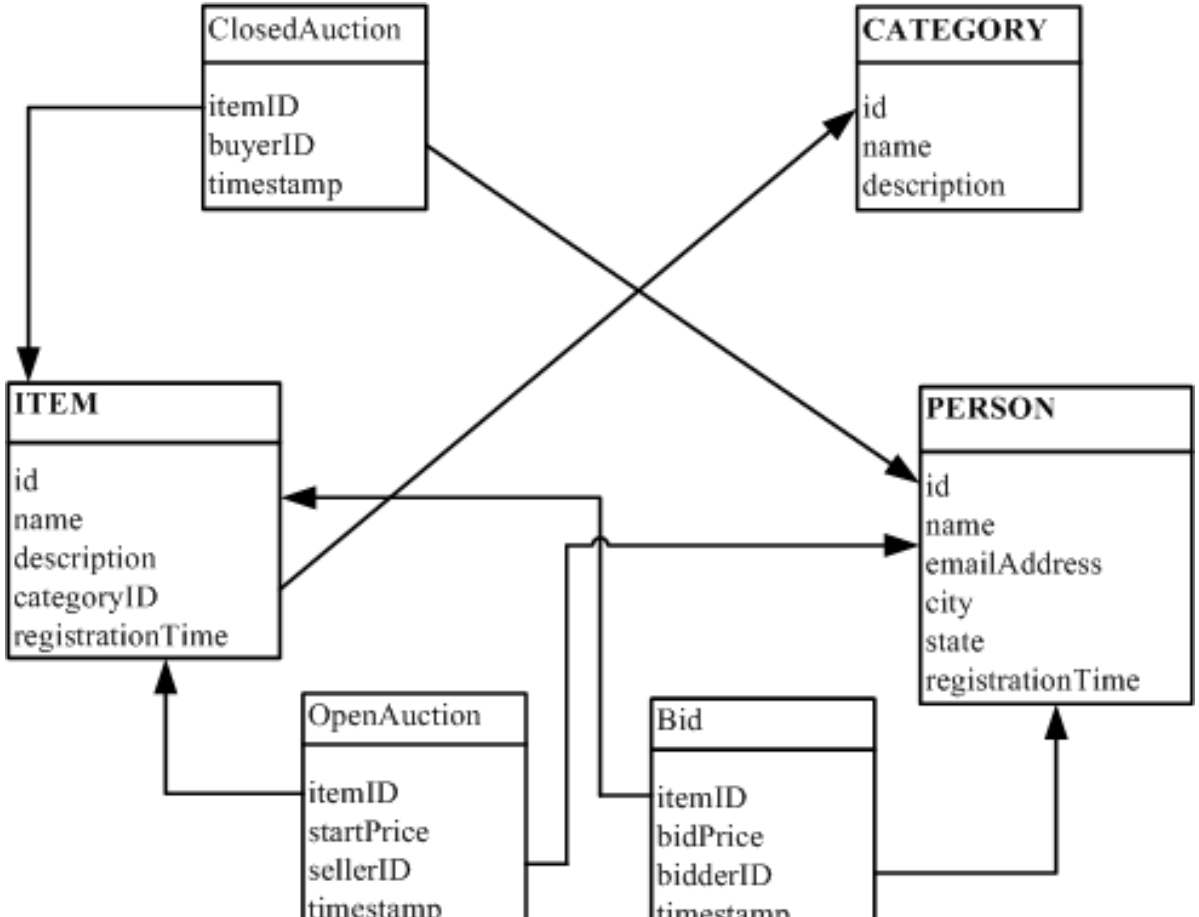
Datenströme

- RFID-Leser, die in der Nähe befindliche („vorbeikommende“) RFID-Tags (Radio Frequency IDentifiers) lesen. Diese Sensoren emittieren Tripel der Art (ReaderID, EPC-Code, TimeStamp), wobei der EPC-Code (electronic product code) eine eindeutige Warenkennzeichnung darstellt.
- Umweltsensoren, um Klimadaten zu ermitteln.
- Börsenticker emittieren die jüngsten Aktienkurse basierend auf den letzten Handelstransaktionen.
- Kameras und andere Sensoren liefern kontinuierliche Informationen über bewegliche Objekte wie z.B. Autos, Menschen, etc.
- Fast alle Menschen liefern über ihr Handy kontinuierlich Daten über ihr Bewegungsmuster. Manche Personen machen es den interessierten Unternehmen noch einfacher, ein präzises Bewegungsmuster via GPS-Daten zu erstellen.

Datenbank versus Datenstrom



Beispiel-Datenstrom



Datenstrom-Definition und einfache Anfrage

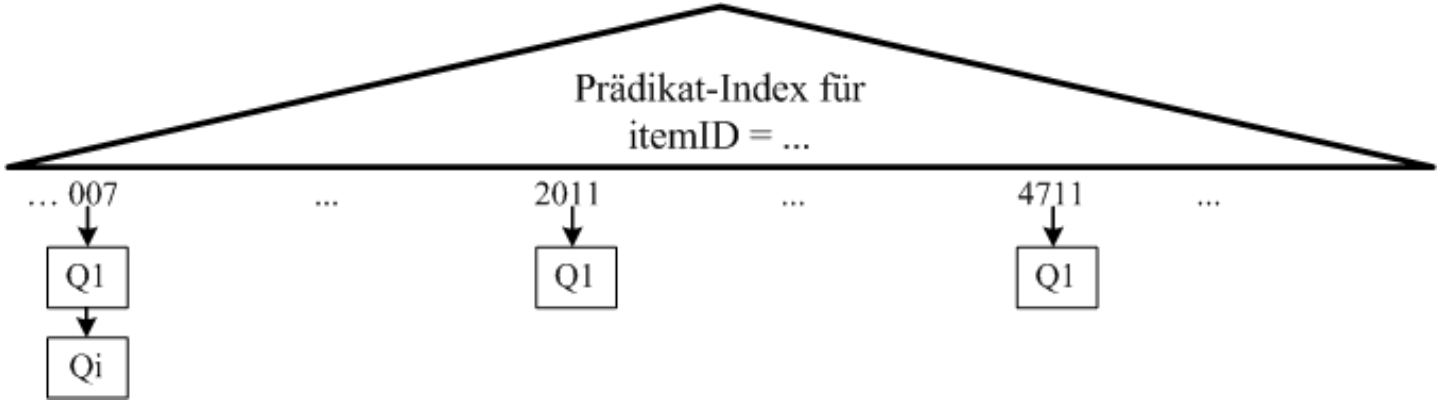
```
create stream OpenAuction(itemID int, sellerID int,
                                    startPrice real, time timestamp)
source establishConnection('port4711' , 'converter')
ordered by time;
```

```
select itemID, DollarToEuro(bidPrice), bidderID
from Bid
```

Subskriptions-Anfrage

Q1: **select** *
from Bid
where itemID = 4711 **or** itemID = 007 **or** itemID = 2011 **or** ...

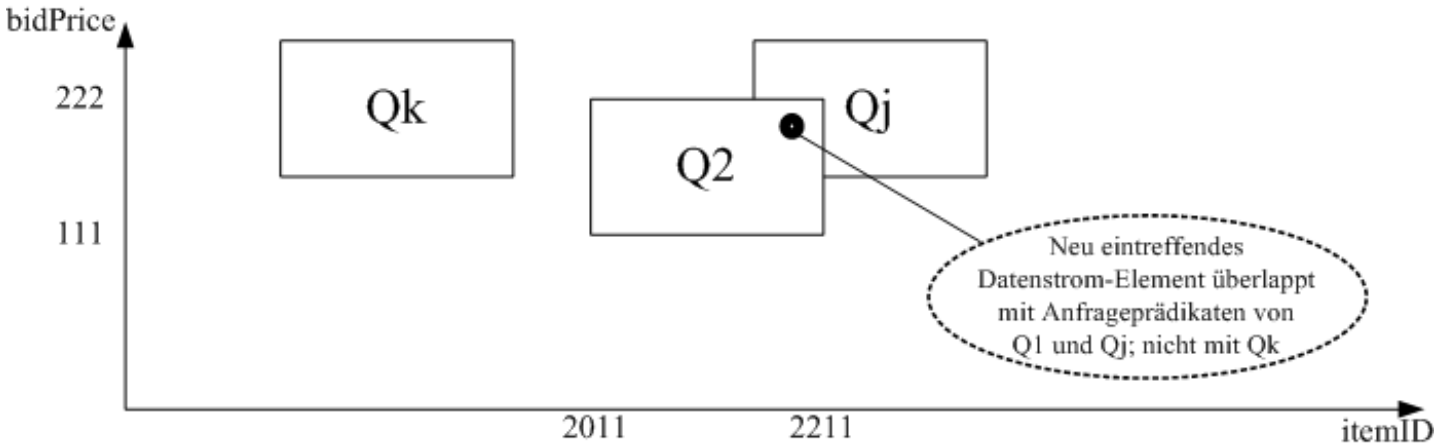
Prädikat-Index



Bereichsanfrage

Q2: **select** *
from Bid
where itemID **between** 2011 **and** 2211 **and**
bidPrice **between** 111 **and** 222

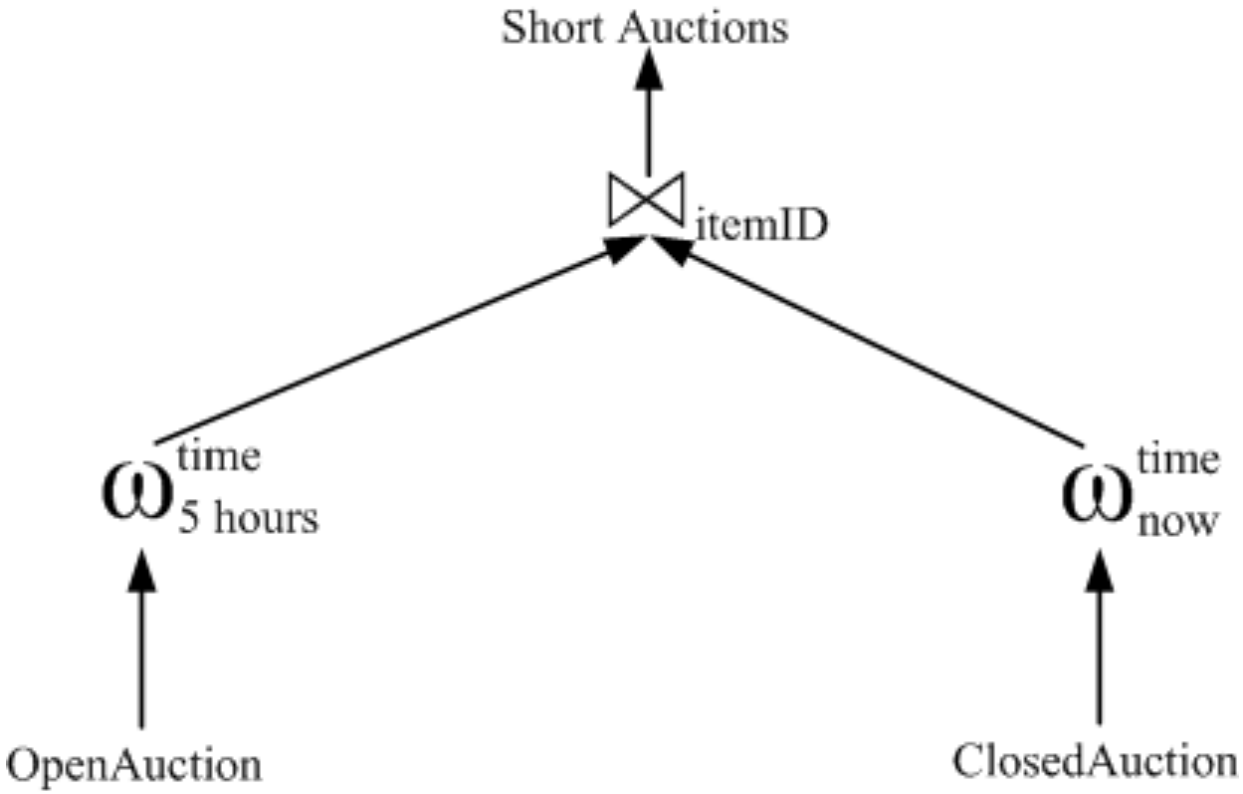
R-Baum-Index


bidPrice
222
111
Qk
Q2
Qj
Neu eintreffendes
Datenstrom-Element überlappt
mit Anfrageprädikaten von
Q1 und Qj; nicht mit Qk
2011
2211
itemID

Fenster-Anfrage

```
select O.*
from OpenAuction O window(range 5 hours),
     ClosedAuction c
where o.itemID = c.itemID
```

Auswertung: Short Auctions

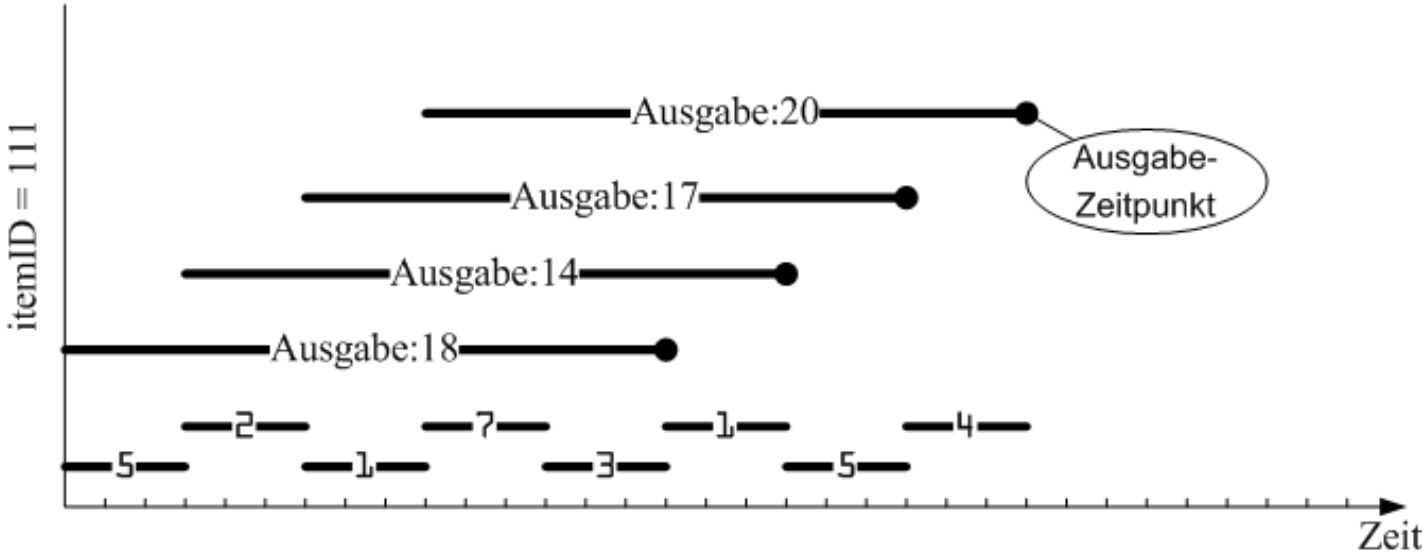


Sliding Windows

```
select itemID, count(*)
from Bid window(range 15 minutes slide 3 minutes)
group by itemID
```

```
select itemID, bidPrice
from Bid b1 window(range 10 minutes)
whereb1.bidPrice = (select max(b2.bidPrice)
                    from Bid b2 window(range 10 minutes))
```

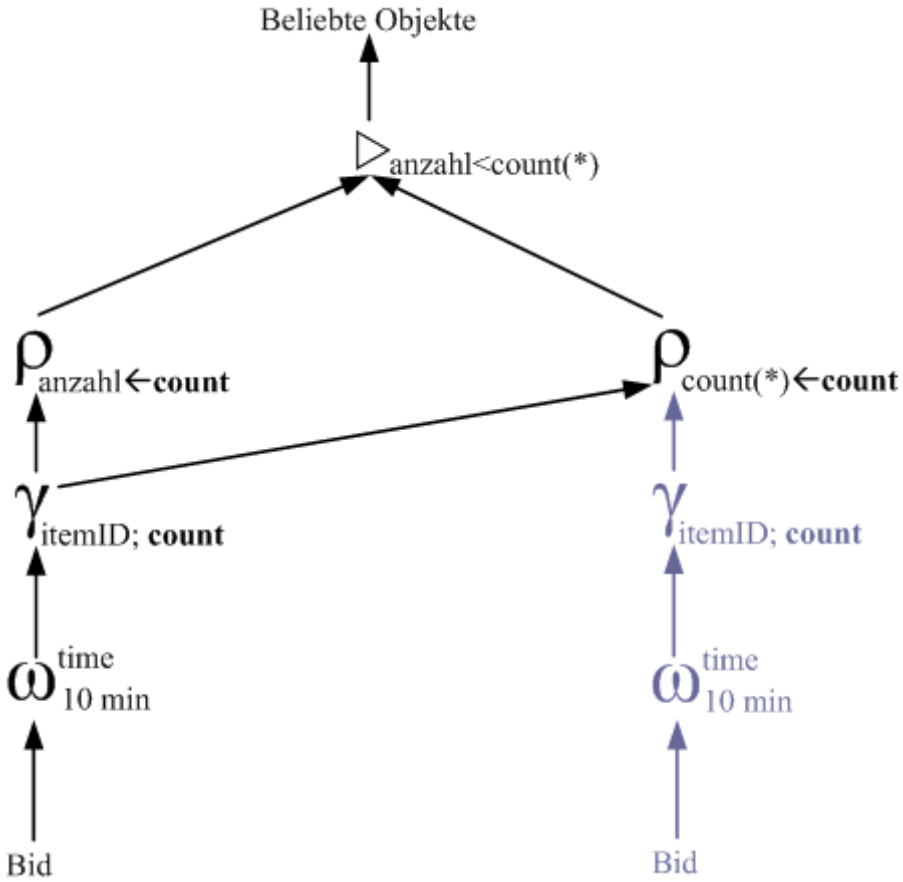
Überlappende Fenster



Hot Items-Anfrage

```
select itemID
from (select b1.itemID as itemID, count(*) as anzahl
      from Bid b1 window(range 10 minutes)
      group by b1.itemID)
where anzahl >= all (select count(*)
                     from Bid b2 window(range 10 minutes)
                     group by b2.itemID)
```

Auswertung: Hot Item



Information Retrieval

Informationsexplosion im Internet

Ranking von Dokumenten um relevante Information zu finden

Ähnlichkeit von Dokumenten (Dissertationen) zu erkennen

TF-IDF: Term Frequency – Inverse Document Frequency

$$TF_{ij} = f_{ij} / \sum_{i=1...|V|} f_{ij}$$

$$IDF_i = log(N/n_i)$$

$$rel(D_j, Q) = \sum_{i \in Q} TF_{ij} * IDF_i$$

Relevanz-Ranking am Beispiel

D_1	D_2	D_3
Nach dem Spiel ist vor dem Spiel.	Was wir ersinnen ist des Zufalls Spiel.	Der Ball ist rund und ein Spiel dauert neunzig Minuten.

TF_{ij}

Wort i	D_1	D_2	D_3
1: Ball	0	0	1/3
2: Minute	0	0	1/3
3: Spiel	2/2	1/2	1/3
4: Zufall	0	1/2	0

IDF_i

Wort i	N	n_i	N/n_i	$log(N/n_i)$
1: Ball	3	1	3	0,477121255
2: Minute	3	1	3	0,477121255
3: Spiel	3	3	1	0
4: Zufall	3	1	3	0,477121255

Anfrage $Q \equiv$ Ball$\wedge$Spiel ermittelt man für das Dokument D_3

$$rel(D_3, Q) = 1/3 * 0,477121255 + 1/3 * 0 = 0,1590404182$$

Relevanz-Ranking am Beispiel

D_1	D_2	D_3
Nach dem Spiel ist vor dem Spiel.	Was wir ersinnen ist des Zufalls Spiel.	Der Ball ist rund und ein Spiel dauert neunzig Minuten.

TF_{ij}

Wort i	D_1	D_2	D_3
1: Ball	0	0	1/3
2: Minute	0	0	1/3
3: Spiel	2/2	1/2	1/3
4: Zufall	0	1/2	0

IDF_i

Wort i	N	n_i	N/n_i	$log(N/n_i)$
1: Ball	3	1	3	0,477121255
2: Minute	3	1	3	0,477121255
3: Spiel	3	3	1	0
4: Zufall	3	1	3	0,477121255

Für die Anfrage $Q' \equiv \text{Ball} \wedge \text{Spiel} \wedge \text{Zufall}$ hat D_2 den höchsten Relevanzwert, nämlich

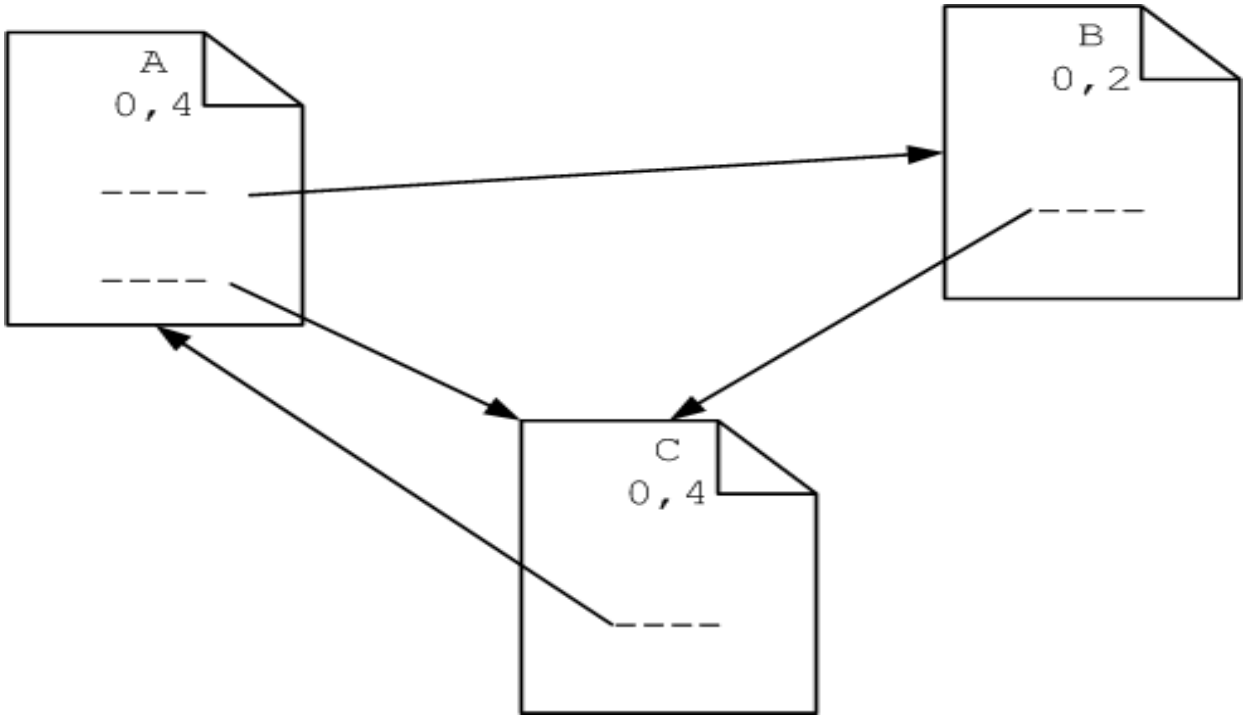
$$rel(D_2, Q') = 0 * 0,477121255 + 1/2 * 0 + 1/2 * 0,477121255 = 0,2385606274$$

Invertierte Indexierung

Begriff	$\rightarrow$	**Dokumente**		
Ball	$\rightarrow$	$(D_3, 1)$		
Minute	$\rightarrow$	$(D_3, 1)$		
Spiel	$\rightarrow$	$(D_1, 2)$	$(D_2, 1)$	$(D_3, 1)$
Zufall	$\rightarrow$	$(D_2, 1)$		

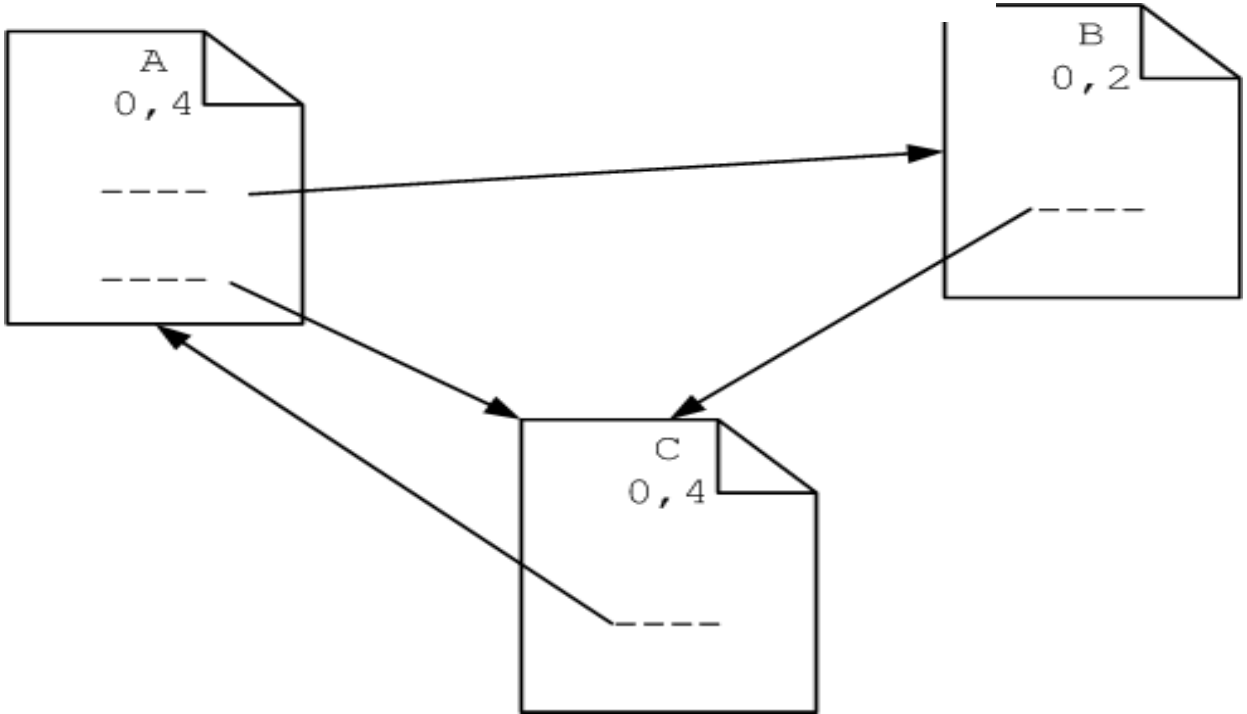
Page Rank: Grundidee

$$r(A) = \frac{\alpha}{N} + (1-\alpha)\left(\frac{r(B_1)}{|B_1|} + \cdots + \frac{r(B_n)}{|B_n|}\right)$$



Page Rank: Grundidee

$$
\begin{aligned}
r(A) &= r(C)/1 \\
r(B) &= r(A)/2 \\
r(C) &= r(A)/2 + r(B)
\end{aligned}
$$



Mathematisches Modell des PageRank

$$M_{ij} = \begin{cases} 1/|P_j| & \texttt{falls } P_j \texttt{ auf } P_i \texttt{ verweist} \\ 0 & \texttt{sonst} \end{cases}$$

$$M = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \end{pmatrix} \quad p_0 = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix}$$

Man berechnet dann iterativ die Vektoren

$$p_1 = M * p_0,\ p_2 = M * p_1 = M * (M * p_0) = M^2 * p_0, \ldots, p_i = M^i * p_0$$

Mathematisches Modell des PageRank: unser Beispiel

$$M = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \end{pmatrix} \qquad p_0 = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix}$$

$$p_1 = Mp_0 = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \end{pmatrix} * \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/6 \\ 1/2 \end{pmatrix}$$

$$p_2 = M * Mp_0 = M * p_1 = \begin{pmatrix} 1/2 \\ 1/6 \\ 1/3 \end{pmatrix}$$

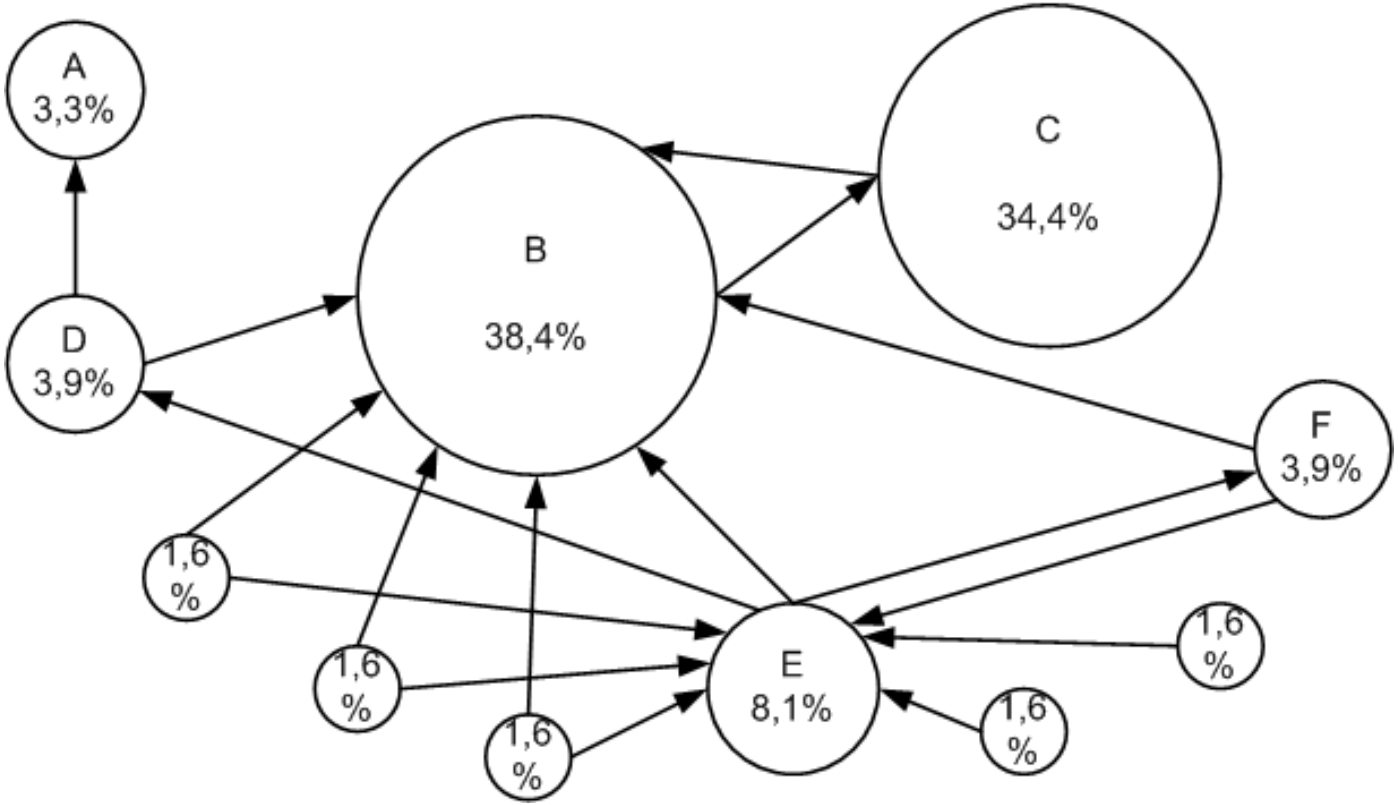
$$p_7 = M^7 * p_0 = \begin{pmatrix} 20/48 \\ 9/48 \\ 19/48 \end{pmatrix}$$

Konvergenz und Dämpfung

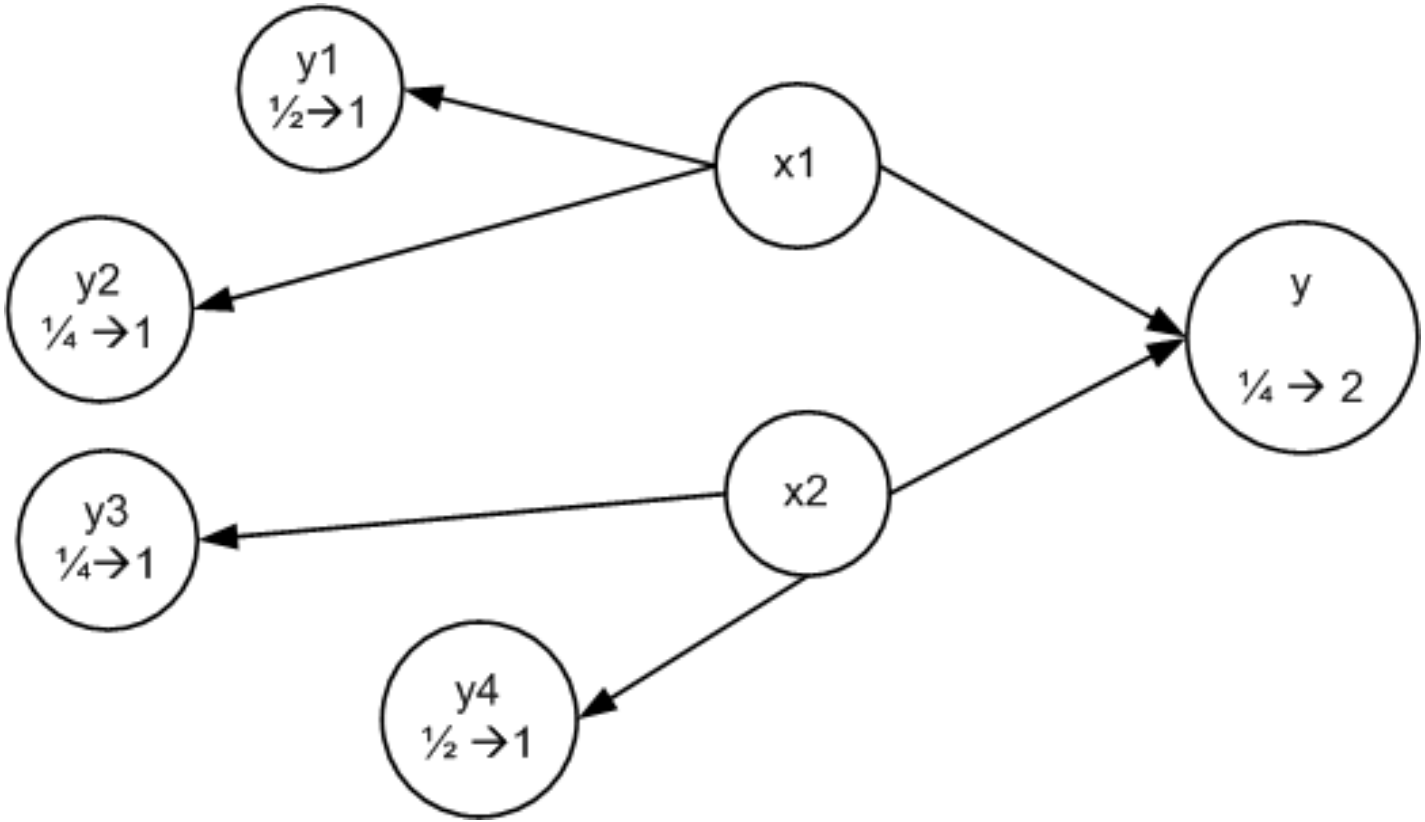
$$p_\infty = \lim_{n \to \infty} M^n p_0$$

$$p_i = (((1 - \alpha) * M) * p_{i-1}) + \begin{pmatrix} \frac{\alpha}{N} \\ \vdots \\ \frac{\alpha}{N} \end{pmatrix}$$

PageRank für größeren Graph [aus Wikipedia]



HITS-Algorithmus: Hubs und Autoritäten



HITS-Algorithmus: Hubs und Autoritäten

Dann wird der Hub-Wert einer Seite i wie folgt definiert:

$$h_i = \delta \sum_{j=1...N} A_{ij} a_j$$

Analog wird die Gewichtung der Autorität einer Seite i wie folgt errechnet:

$$a_i = \lambda \sum_{k=1...N} A^T_{ik} h_k$$

Eigenvektoren der Matrizen AA^T bzw. A^TA

$$\begin{aligned} h &= \delta\lambda AA^T h \\ a &= \delta\lambda A^T Aa \end{aligned}$$

Relationale HITS-Modellierung

A

von	nach
x1	y1
x1	y2
x2	y3
x2	y4
x1	y
x2	y
...	...

Aut

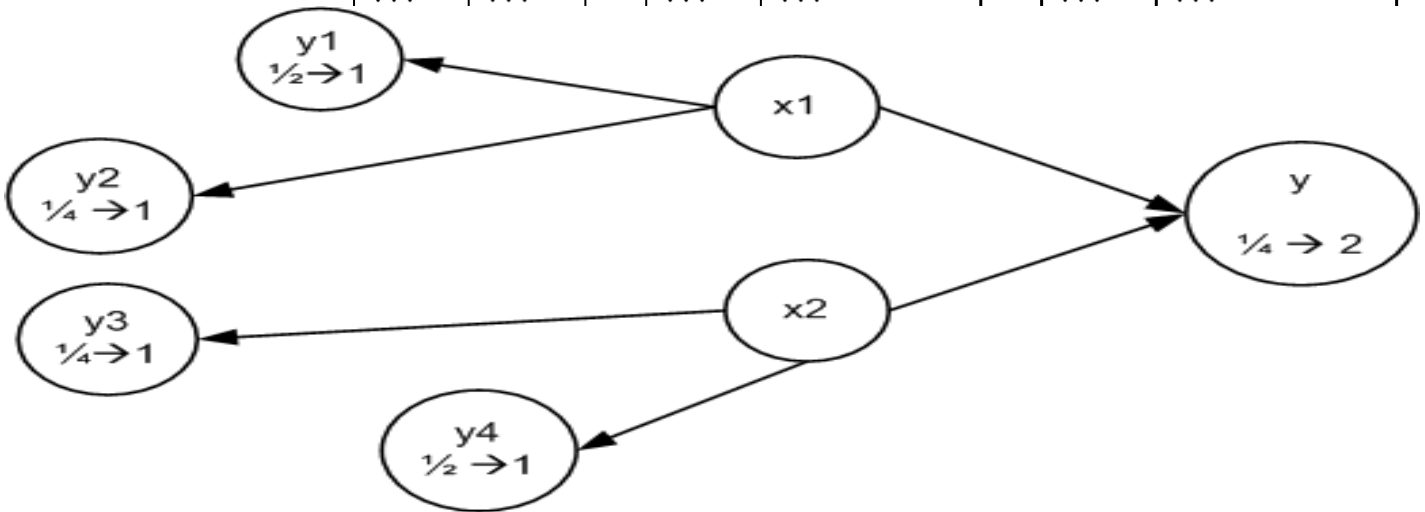
Seite	Wert
x1	...
x2	...
y1	1/2
y2	1/4
y3	1/4
y4	1/2
y	1/4
...	...

Aut2 (nach insert)

Seite	Wert
x1	...
x2	...
y1	1
y2	1
y3	1
y4	1
y	2
...	...

Aut2 (nach update)

Seite	Wert
x1	...
x2	...
y1	1/2
y2	1/2
y3	1/2
y4	1/2
y	1
...	...



Algorithmus

1. Berechne die Hub-Werte jeder Seite q indem man die Summe der Autoritätswerte aller Seiten r ermittelt, auf die q verweist.

2. Berechne die Autorität der Seite p durch Summierung der Hub-Werte der Seiten q, die auf p verweisen

3. Normalisiere die so erhaltenen Autoritätswerte indem man sie mit $\lambda = 1/max$ multipliziert, wobei max den Maximalwert aller gerade neu berechneten Autoritätswerte darstellt. Diese Normalisierung ist nötig, um die Werte nicht ins Unermessliche steigen zu lassen.

Algorithmus … in SQL

```
insert into Aut2 (
    select a1.nach, sum(Aut.Wert)
    from Aut, A a1, A a2
    where Aut.Seite = a2.nach and a1.von = a2.von
    group by a1.nach )
```

```
update Aut2
    set Wert = Wert / (select max(Wert) from Aut2)
```

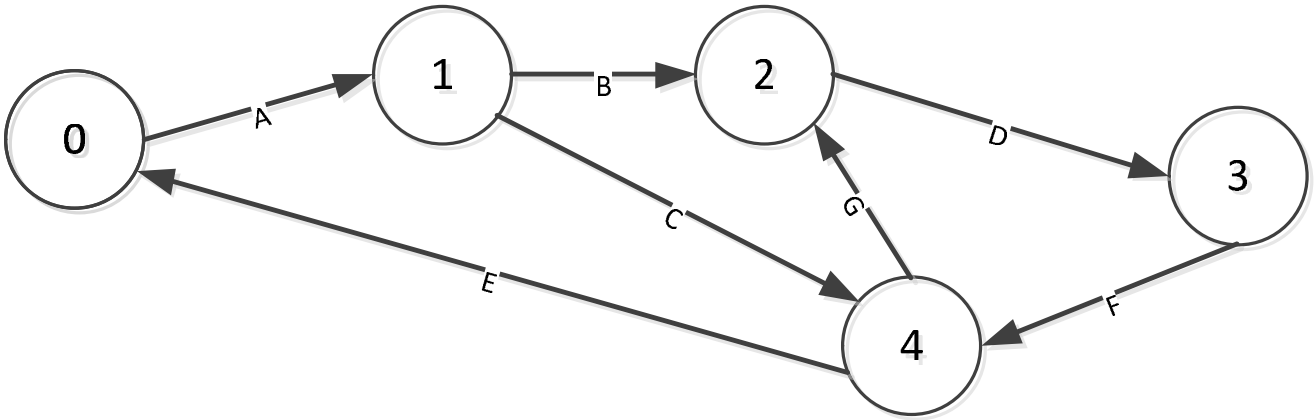
Graph Exploration

Analyse großer sozialer Netzwerke (Facebook, Twitter, Instagram, etc)

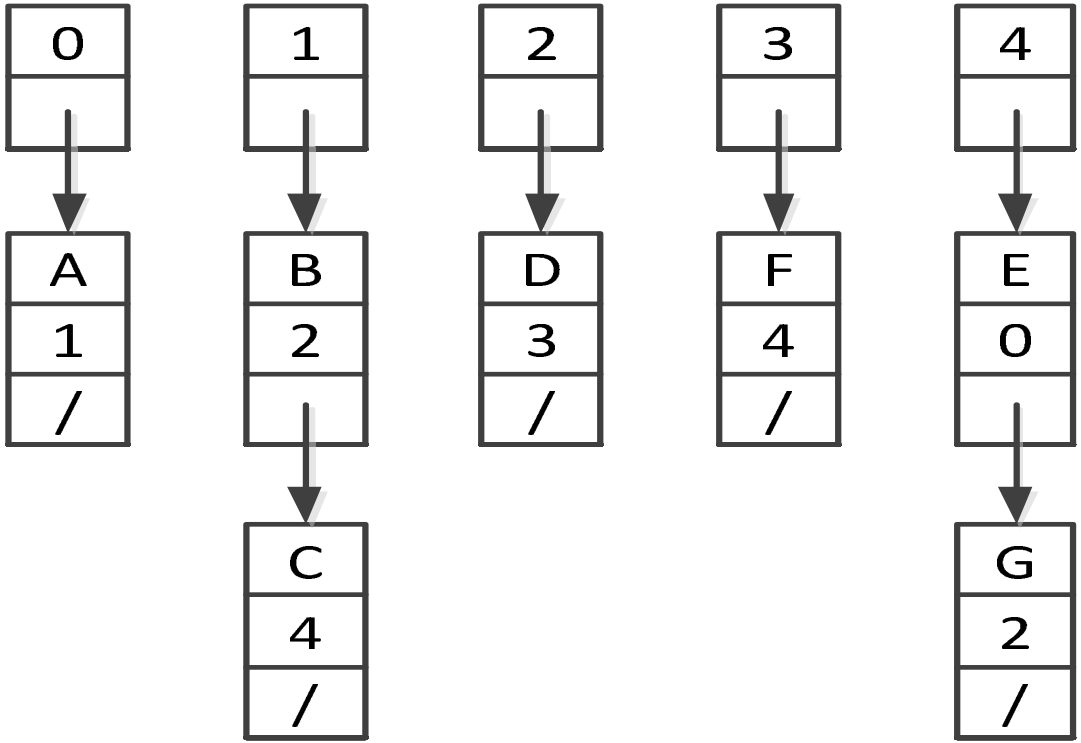
Analyse von Kommunikationsnetzen

…

Darstellung von Graphen



Adjazenzlisten-Darstellung

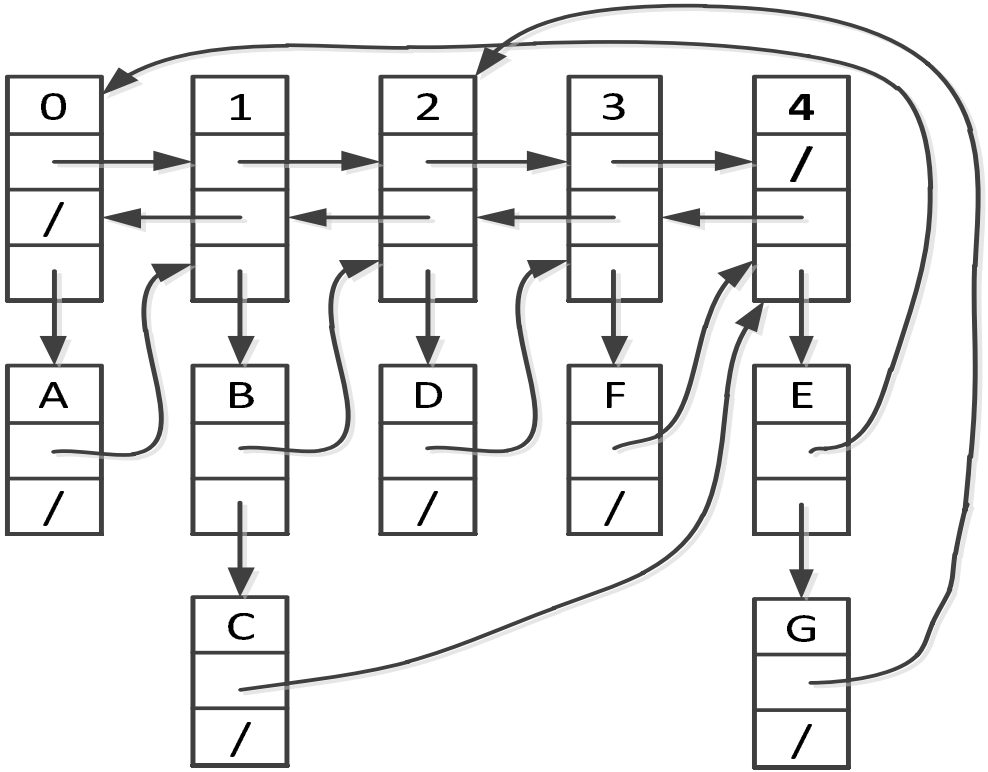


: Die Darstellung des Graphen als Adjazenzliste

Adjazenzmatrix

$$A = \begin{pmatrix} 0 & A & 0 & 0 & 0 \\ 0 & 0 & B & 0 & C \\ 0 & 0 & 0 & D & 0 \\ 0 & 0 & 0 & 0 & F \\ E & 0 & G & 0 & 0 \end{pmatrix}$$

Verzeigerte Adjazenzlisten-Struktur



Die vollständig verzeigerte Adjazenzlisten-Struktur

Compressed sparse rows (CSR)

$$Werte = \begin{pmatrix} A & B & C & D & F & E & G \\ (0,1) & (1,2) & (1,4) & (2,3) & (3,4) & (4,0) & (4,2) \end{pmatrix}$$

$$SpaltenIndex = \begin{pmatrix} 1 & 2 & 4 & 3 & 4 & 0 & 2 \\ (0) & (1 &) & (2) & (3) & (4 &) \end{pmatrix}$$

$$ZeilenPtr = \begin{pmatrix} 0 & 1 & 3 & 4 & 5 & 7 \\ (<0) & (0..0) & (0..1) & (0..2) & (0..3) & (0..4) \end{pmatrix}$$

$$A_{ij} = Wert[k] \iff SpaltenIndex[k] = j \quad \wedge \quad ZeilenPtr[i] \leq k < ZeilenPtr[i+1]$$

Rekonstruktion aus der CSR

$$A_{ij} = Wert[k] \iff SpaltenIndex[k] = j \quad \wedge \quad ZeilenPtr[i] \leq k < ZeilenPtr[i+1]$$

Die Rekonstruktion – beispielsweise der Zeile 1 der Matrix A – geschieht dann wie folgt: Die Einträge 1 bzw. 3 an den Positionen 1 und 2 des Vektors *ZeilenPtr* zeigen an, dass die Zeile 1 insgesamt $3 - 1 = 2$ nicht-Null-Einträge hat. Diese Adjazenzmatrizen-Werte findet man also an den Positionen 1 und 2 des *Werte*-Vektors, also B und C. Anhand des *SpaltenIndex*-Vektors ermittelt man dann noch, dass sie zu den Spalten 2 und 4 gehören, also ergibt sich $A_{12} = B$ und $A_{14} = C$. Für die automatische Rekonstruktion baut man sich natürlich einen entsprechenden Iterator, der die nicht-Null-Einträge der Adjazenzmatrix mitsamt ihren Zeile/Spalte-Indexpositionen der Reihe nach zurückgibt. Dies sei den Lesern als Übung empfohlen.

Zentralitätsmaße

aus den Sozialwissenschaften

charakterisiert ganze Graphen oder Teilstrukturen

Sinnhaftigkeit ist manchmal zweifelhaft

Verbindungszentralität

(degree centrality)

$$C_D(V) = degree(v)$$

$$C_D(G) = \sum_{i=1}^{|V|}[C_D(v^*) - C_D(v_i)]$$

Normierung mit dem Maximalwert für den sternförmigen Graphen

$$C_D(G^*) = \sum_{i=1}^{|V|}[C_D(v*) - C_D(v_i)] = (|V| - 2)(|V| - 1)$$

$$C'_D(G) = C_D(G)/[(|V| - 2)(|V| - 1)]$$

Nähe-Zentralität (closeness centrality)

$$C(v) = \frac{1}{\sum_{y \in V} d(y, v)}$$

$$H(v) = \sum_{v \neq y \in V} 1/d(y, v)$$

Hierbei definiert man $1/\infty$ als 0.

Nähe-Zentralität (closeness centrality)

stärkere Gewichtung der Nähe zweier Knoten:

$$D(v) = \sum_{v \neq y \in V} \frac{1}{2^{d(y,v)}}$$

Pfad-Zentralität (betweenness centrality)

Für einen Knoten v im Graph $G = (V, E)$ wird also dieser Wert wie folgt bestimmt:

1. Für jedes Knotenpaar (s, t) berechne deren kürzeste Pfade im Graphen.
2. Bestimme die Anzahl der kürzesten Pfade von s nach t als σ_{st}.
3. Für jedes Knotenpaar (s, t) bestimme die Anzahl der kürzesten Pfade, die durch den betrachteten Knoten v verlaufen. Dieser Wert sei als $\sigma_{st}(v)$ bezeichnet.

$$C_B(v) = \sum_{s \neq v \neq t \in V} (\sigma_{st}(v)/\sigma_{st})$$

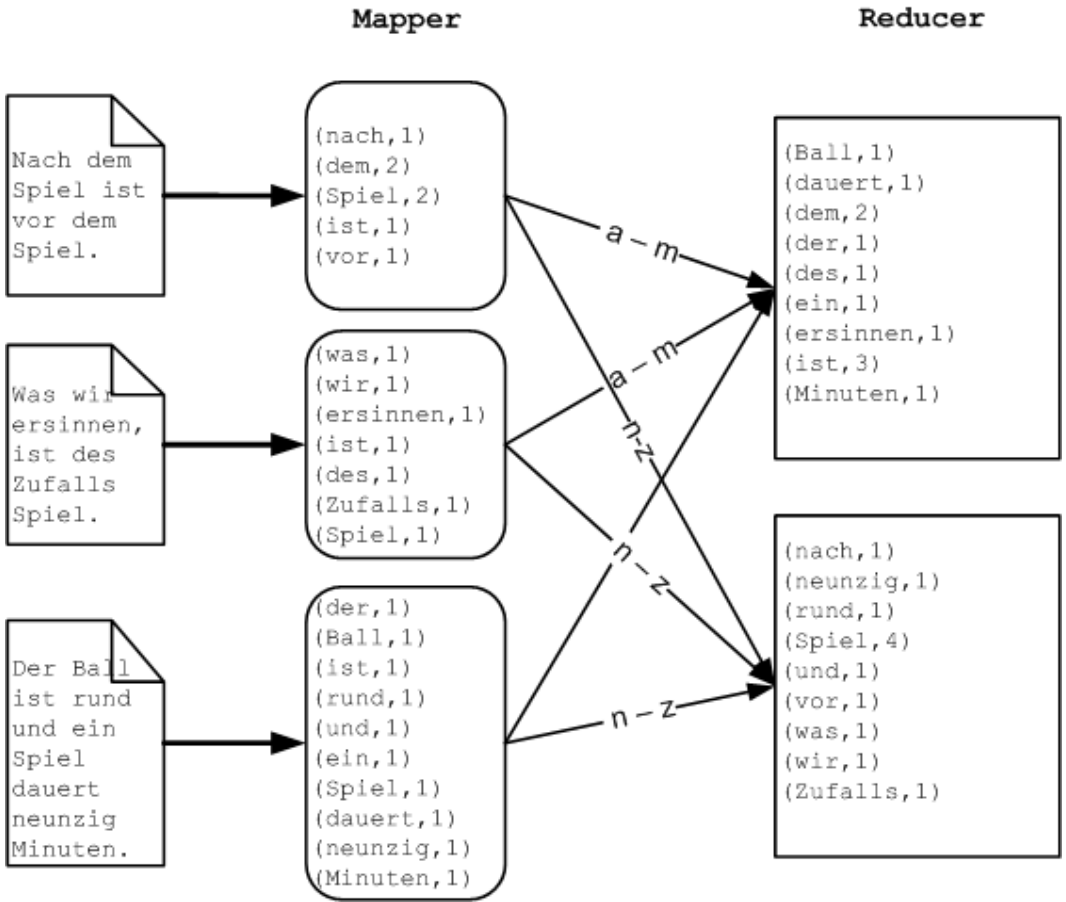
Pfad-Zentralität (betweenness centrality)

Normierung mit dem Wert für den zentralsten Knoten in einem sternförmigen Graphen:

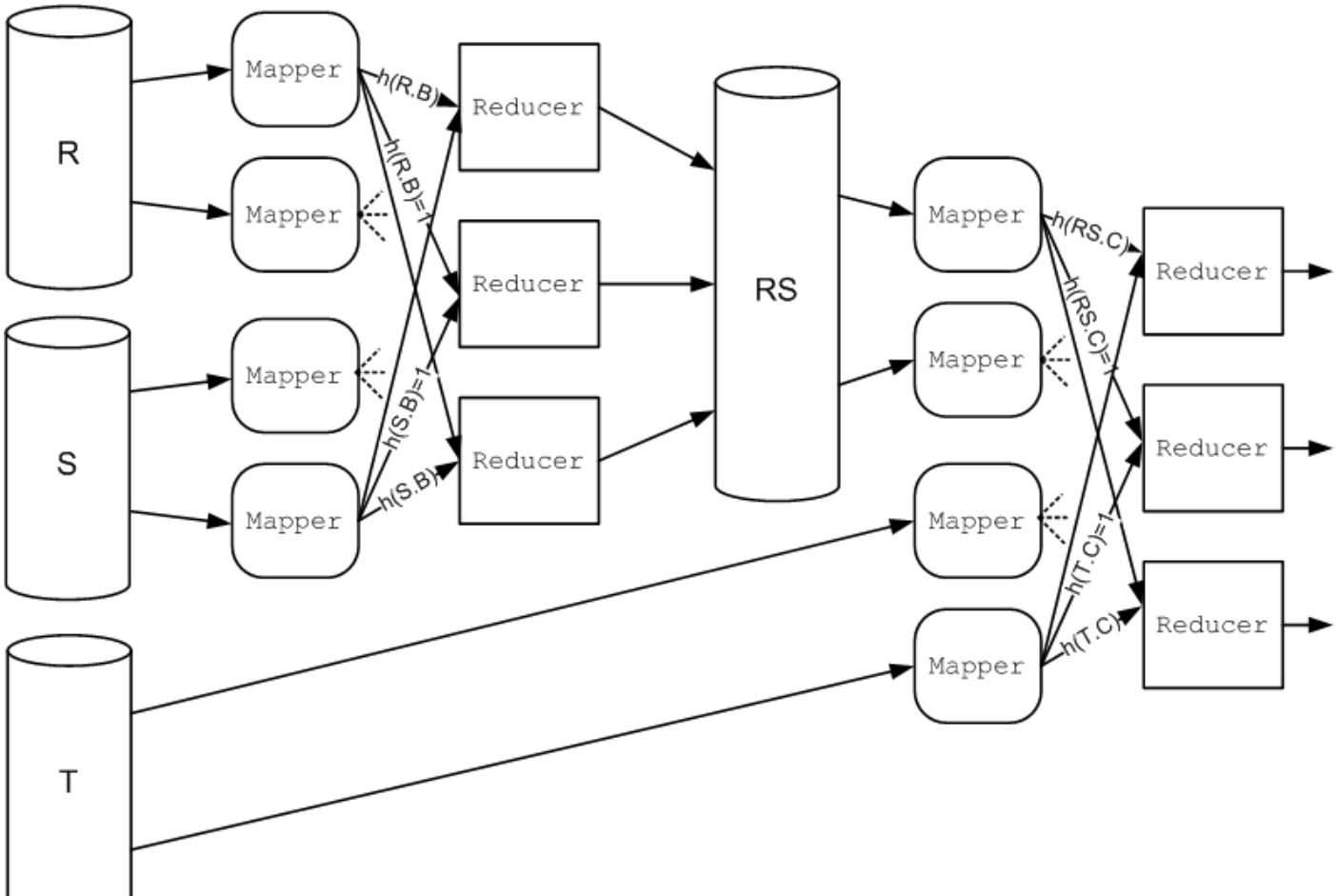
$$C_B(v) = \sum_{s \neq v \neq t \in V} (\sigma_{st}(v)/\sigma_{st})$$

$$C_B^*(v) = C_B(v)/[((n-1)(n-2))/2] = \frac{\sum_{s \neq v \neq t \in V} (\sigma_{st}(v)/\sigma_{st})}{[(n-1)(n-2)]/2}$$

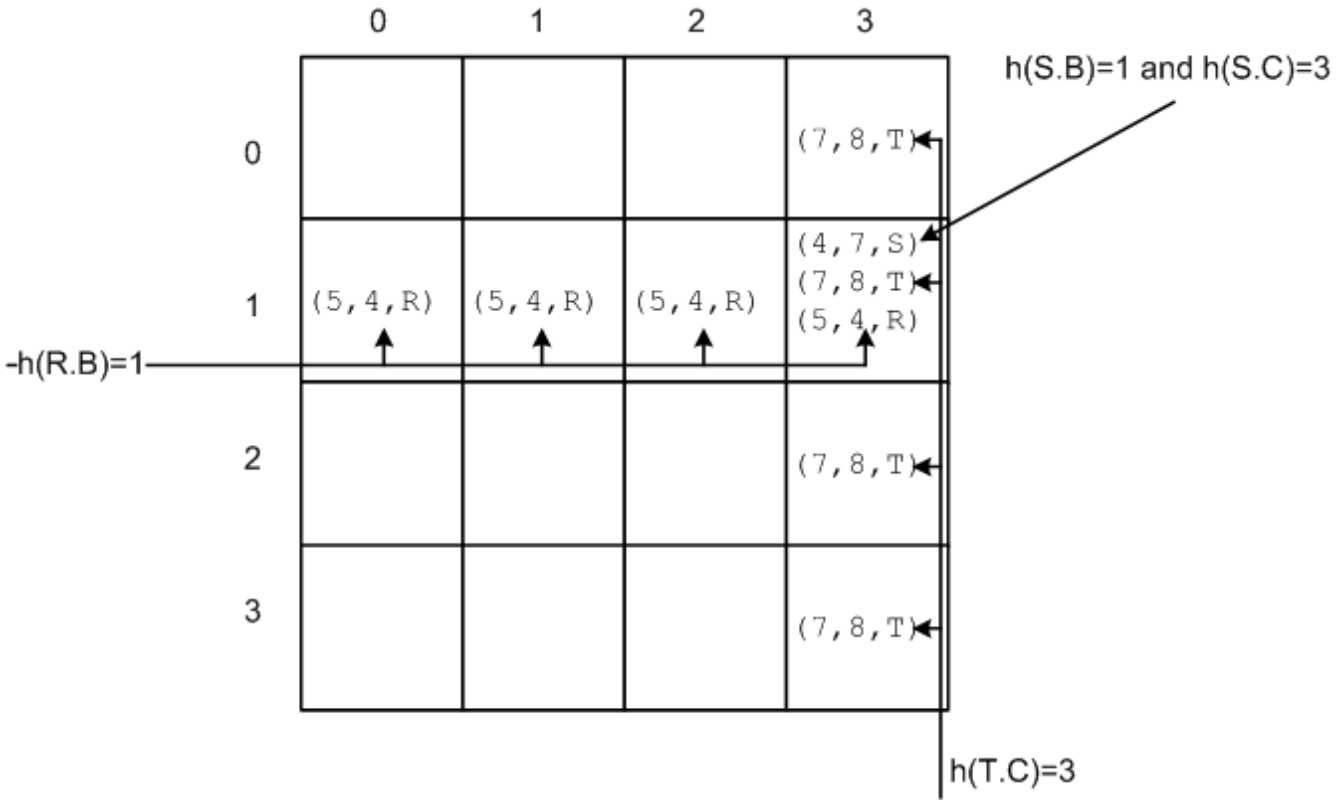
Map Reduce


Mapper
Reducer
Nach dem Spiel ist vor dem Spiel.
Was wir ersinnen, ist des Zufalls Spiel.
Der Ball ist rund und ein Spiel dauert neunzig Minuten.
(nach,1)
(dem,2)
(Spiel,2)
(ist,1)
(vor,1)
(was,1)
(wir,1)
(ersinnen,1)
(ist,1)
(des,1)
(Zufalls,1)
(Spiel,1)
(der,1)
(Ball,1)
(ist,1)
(rund,1)
(und,1)
(ein,1)
(Spiel,1)
(dauert,1)
(neunzig,1)
(Minuten,1)
a – m
a – m
n-z
n – z
n – z
(Ball,1)
(dauert,1)
(dem,2)
(der,1)
(des,1)
(ein,1)
(ersinnen,1)
(ist,3)
(Minuten,1)
(nach,1)
(neunzig,1)
(rund,1)
(Spiel,4)
(und,1)
(vor,1)
(was,1)
(wir,1)
(Zufalls,1)

Join mit Map Reduce



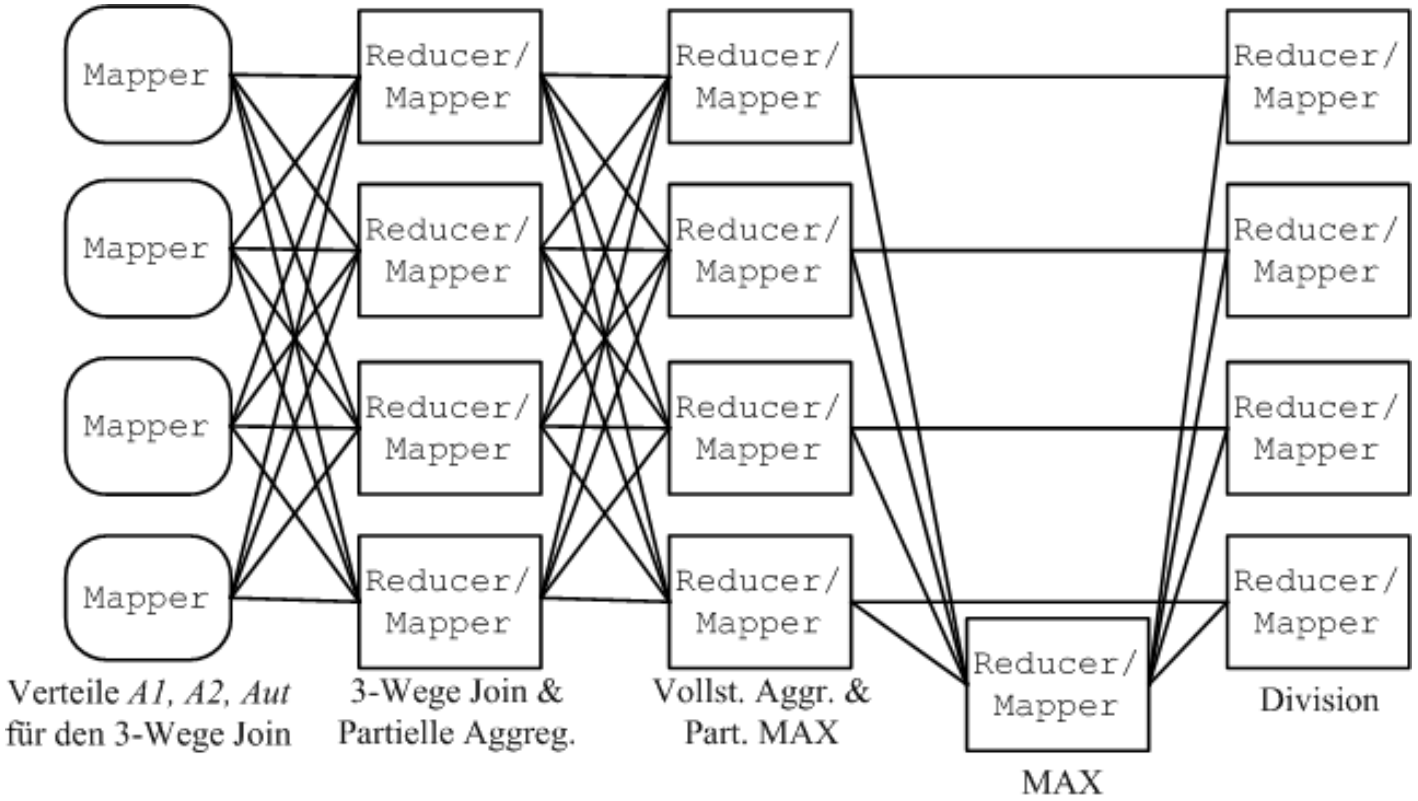
Verbesserung nach Ullman



Map Reduce Skripsprache: PigLatin

```
a1 = LOAD 'A' AS (a1von, a1nach);
a2 = LOAD 'A' AS (a2von, a2nach);
aut = LOAD 'Aut' AS (seite, wert);
j1 = JOIN a2 BY a2nach, aut BY seite;
j2 = JOIN a1 BY a1von, j1 BY a2von;
g1 = GROUP j2 BY a1nach;
aut2 = FOREACH g1 GENERATE group AS seite, SUM(j2.wert) AS wert;
g2 = GROUP aut2 ALL;
max = FOREACH g2 GENERATE MAX(aut2.wert) AS max;
c = CROSS aut2, max;
aut2Up = FOREACH c GENERATE seite, wert / max;
STORE aut2Up INTO 'Aut2';
```

Auswertung des HITS Algorithmus

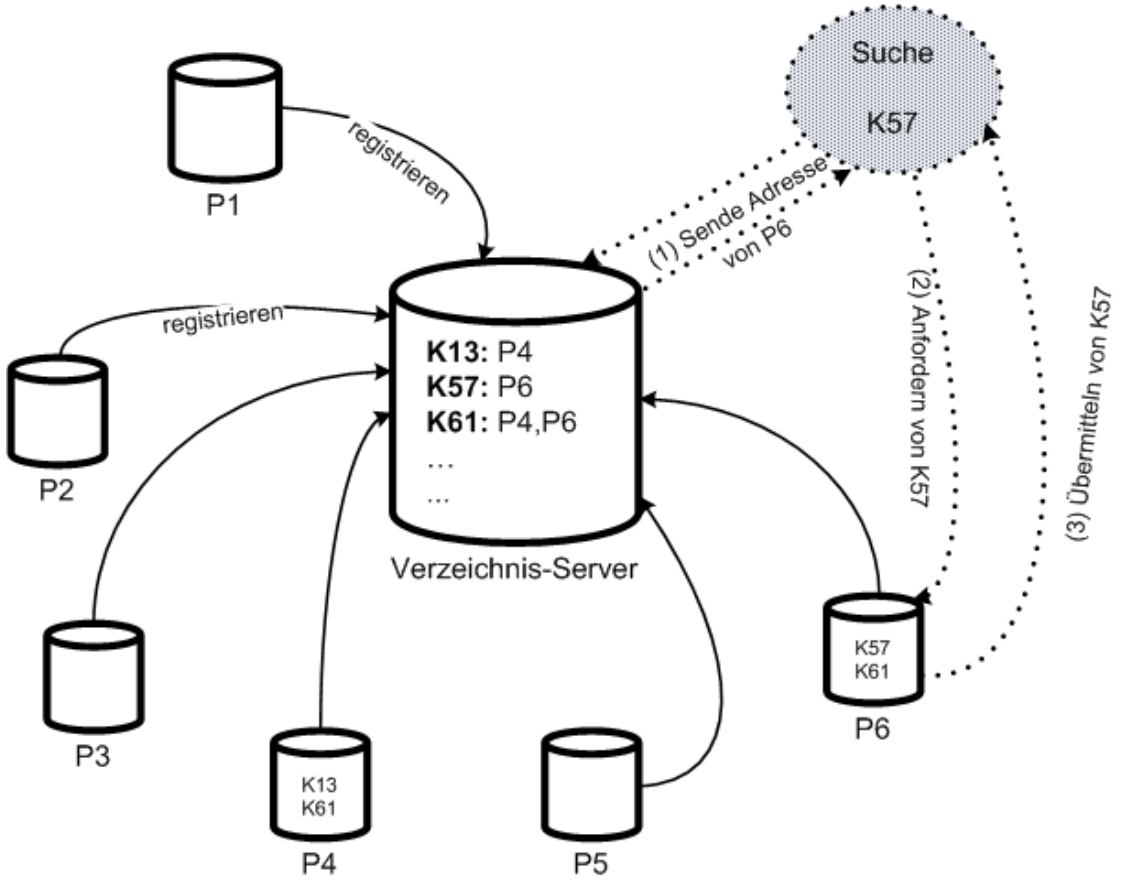


Peer to Peer-Informationssysteme

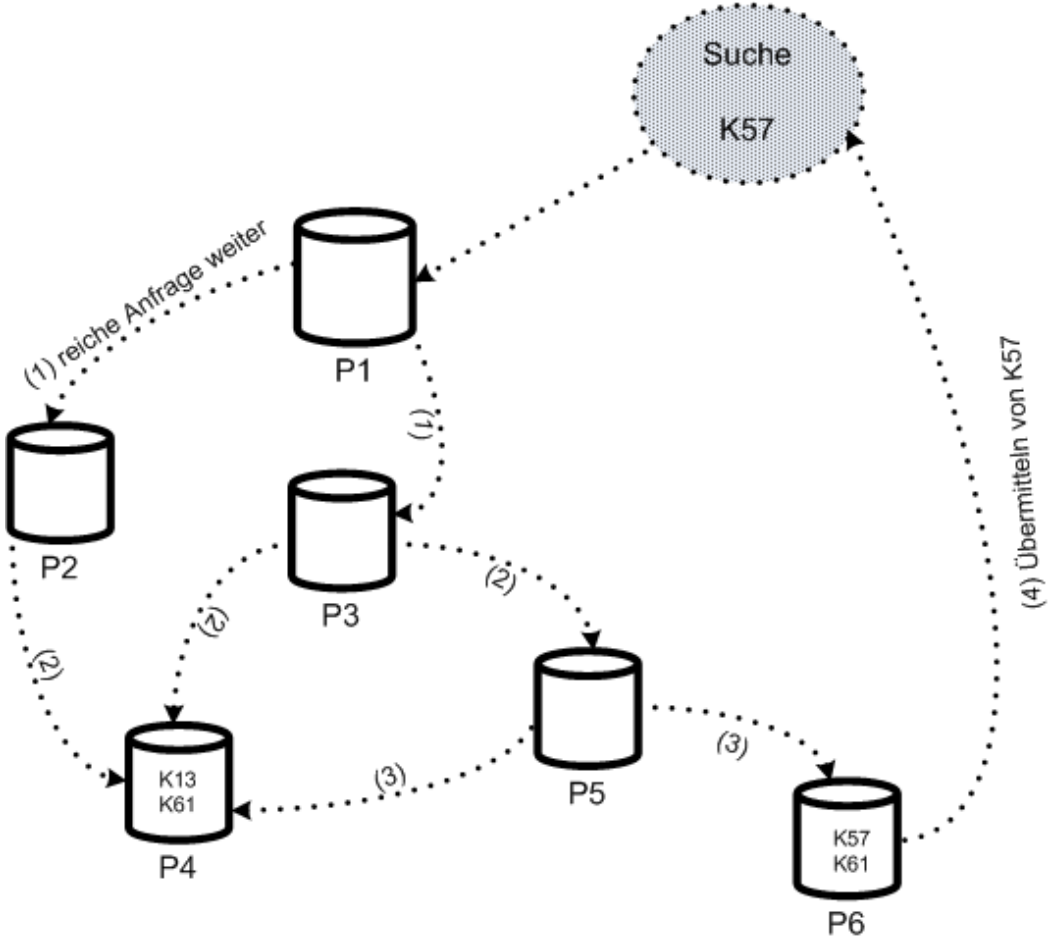
Seti@Home
- P2P number crunching

Napster
- P2P file sharing / Informationsmanagement

Napster-Architektur



Gnutella-Architektur



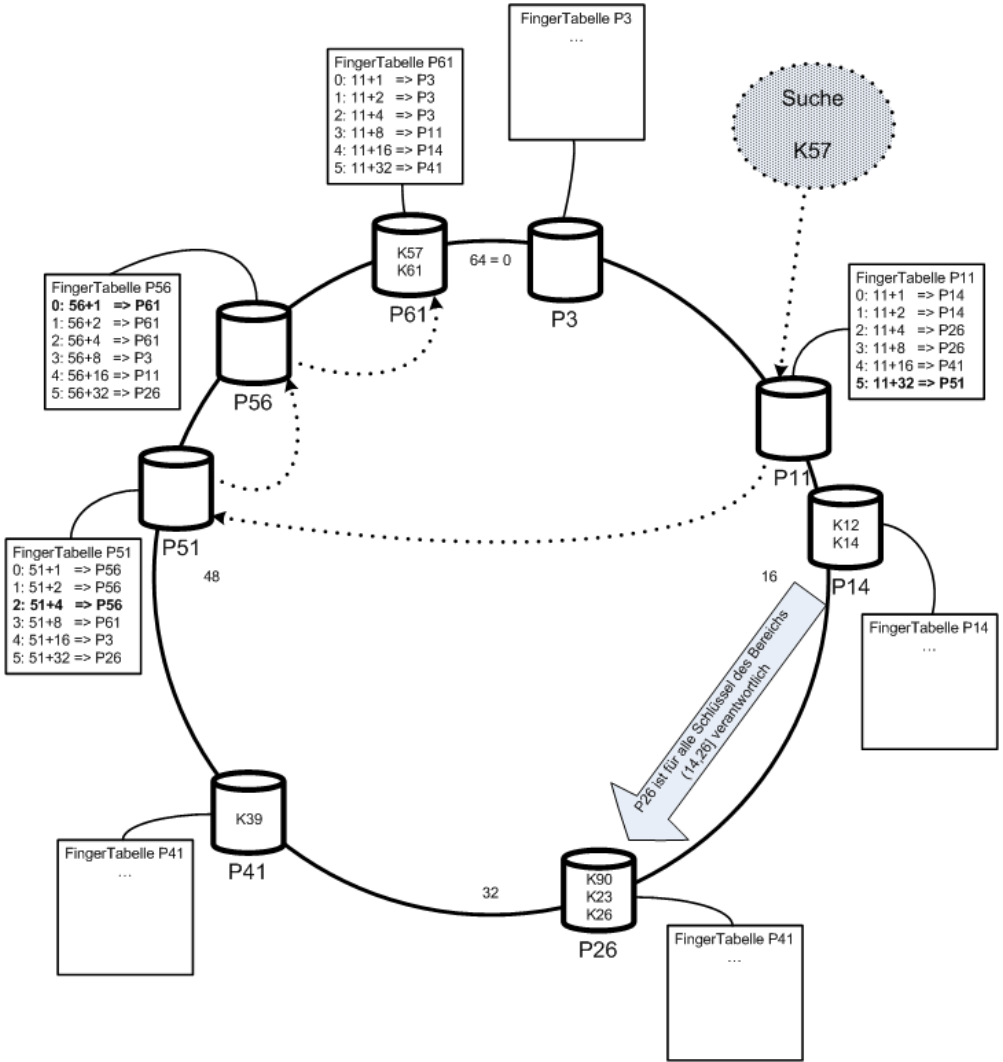
DHT: Distributed Hash Table

Basieren auf „consistent hashing“

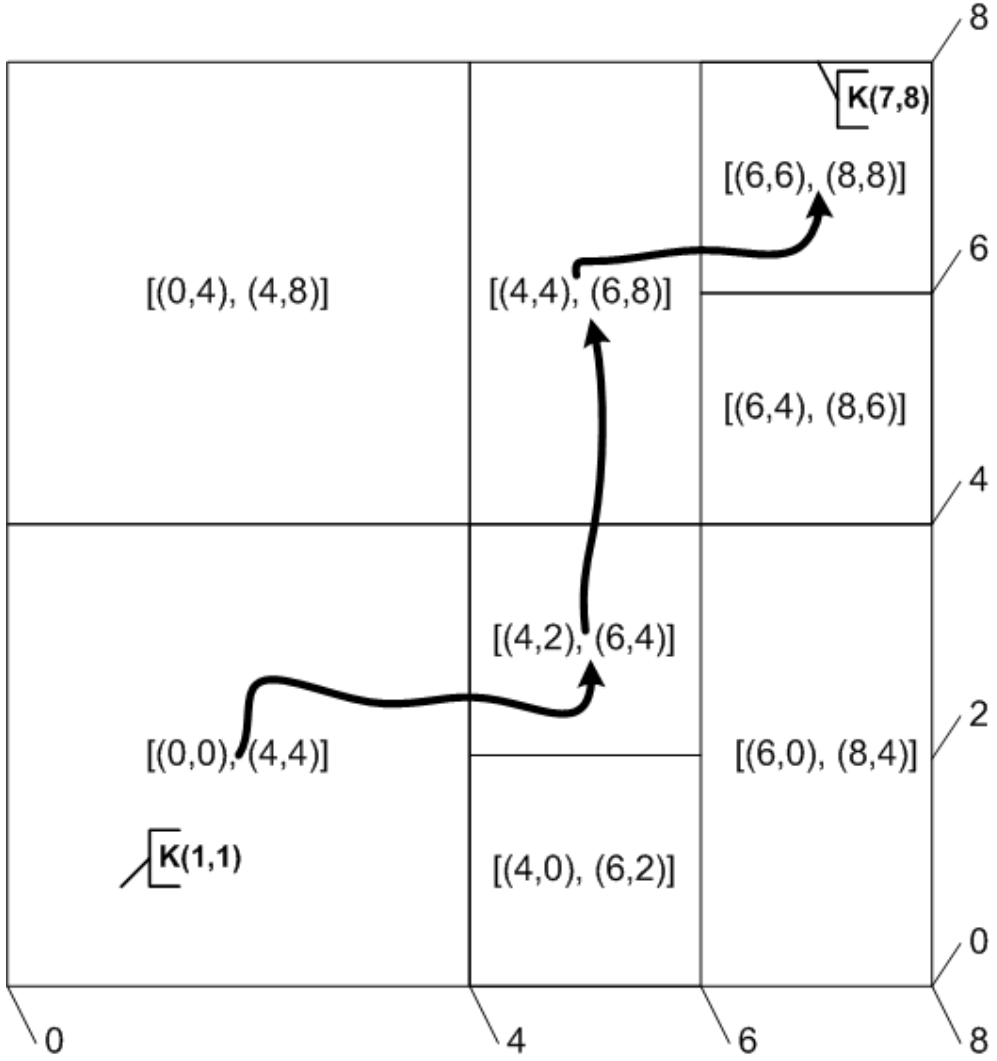
Vollständige Dezentralisierung der Kontrolle

Dennoch zielgerichtete Suche

CHORD


FingerTabelle P3
...
FingerTabelle P61
0: 11+1 => P3
1: 11+2 => P3
2: 11+4 => P3
3: 11+8 => P11
4: 11+16 => P14
5: 11+32 => P41
Suche
K57
K57
K61
P61
64 = 0
P3
FingerTabelle P56
0: 56+1 => P61
1: 56+2 => P61
2: 56+4 => P61
3: 56+8 => P3
4: 56+16 => P11
5: 56+32 => P26
P56
FingerTabelle P11
0: 11+1 => P14
1: 11+2 => P14
2: 11+4 => P26
3: 11+8 => P26
4: 11+16 => P41
5: 11+32 => P51
P11
P51
FingerTabelle P51
0: 51+1 => P56
1: 51+2 => P56
2: 51+4 => P56
3: 51+8 => P61
4: 51+16 => P3
5: 51+32 => P26
48
16
K12
K14
P14
FingerTabelle P14
...
P26 ist für alle Schlüssel des Bereichs (14,26] verantwortlich
K39
FingerTabelle P41
...
P41
32
K90
K23
K26
P26
FingerTabelle P41
...

CAN



No-SQL Datenbanken

Internet-scale Skalierbarkeit
CAP-Theorem: nur 2 von 3 Wünschen erfüllbar
- Konsistenz (Consistency)
- Zuverläassigkeit/Verfügbarkeit (Availability)
- Partitionierungs-Toleranz

No-SQL Datenbanksysteme verteilen die Last innerhalb eines Clusters/Netzwerks
- Dabei kommen oft DHT-Techniken zum Einsatz

Schnittstelle der No-SQL Datenbanken

Insert(k,v)
Lookup(k)
Delete(k)

Extrem einfach ➔ effizient
Aber: wer macht denn die Joins/Selektionen/…

- ➔ das Anwendungsprogramm

Konsistenzmodell: ~~C~~AP

Relaxiertes Konsistenzmodell

- Replizierte Daten haben nicht alle den neuesten Zustand
  - Vermeidung des (teuren) Zwei-Phasen-Commit-Protokolls
- Transaktionen könnten veraltete Daten zu lesen bekommen
- Eventual Consistency
  - Würde man das System anhalten, würden alle Kopien irgendwann (also eventually) in denselben Zustand übergehen
- Read your Writes-Garantie
  - Tx leist auf jeden Fall ihre eigenen Änderungen
- Monotonic Read-Garantie
  - Tx würde beim wiederholten Lesen keinen älteren Zustand als den vorher mal sichtbaren lesen

Systeme

MongoDB
Cassandra
Dynamo
BigTable
Hstore
SimpleDB
S3

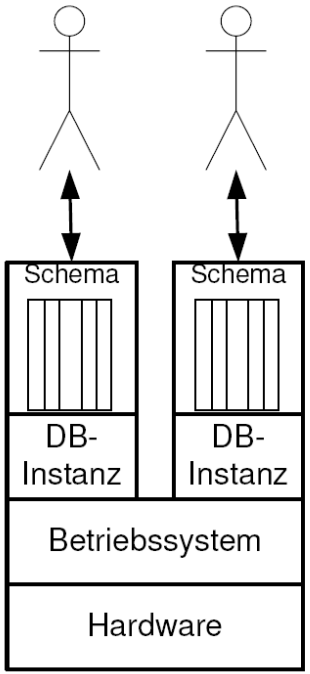
Multi-Tenancy / Cloud-Datenbanken

- **Infrastructure as a Service (IaaS)**: Hierbei werden den Kunden de facto virtuelle Maschinen zur Verfügung gestellt, auf denen dann beliebige Software installiert werden könnte. Kunden könnten also auch existierende Anwendungen auf eine derartige virtuelle Maschine portieren. Insbesondere kann man natürlich „normale“ Datenbanksysteme installieren. Amazon Web Services ist ein typisches Beispiel einer IaaS, die aber zusätzlich auch nicht-relationale (also No-SQL) Datenbankfunktionalität durch die Systeme SimpleDB und S3 anbietet.

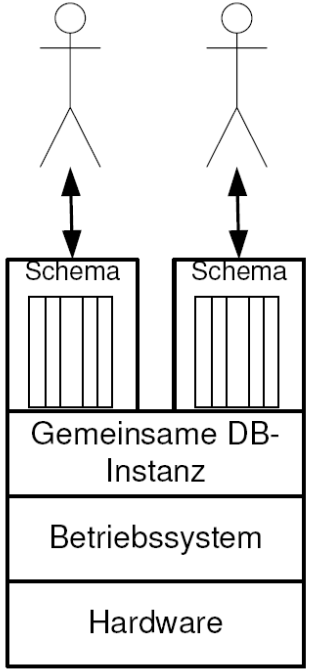
- **Platform as a Service (PaaS)**: Hierzu zählen die Systeme Google AppEngine und Microsoft Azure, die reichhaltige Schnittstellen für die Neuentwicklung von Web-Applikationen bereitstellen. Hierzu dienen insbesondere die Datenspeicher, also Google's App Engine Datastore oder Microsoft's Azure Table Storage.

- **Software as a Service (SaaS)**: Diese Systeme stellen komplexe, anwendungsspezifische Funktionalität zur Verfügung. Bekannteste Beispiele im betrieblichen Umfeld sind das Customer-Relationship-Management-System von SalesForce oder das umfassende betriebliche Anwendungssystem Business-By-Design von SAP. Diese Software wird nicht mehr bei den Nutzern installiert sondern von den Betreibern als „Hosting-Modell“ angeboten und von den Nutzern via Web-Schnittstellen zugegriffen.

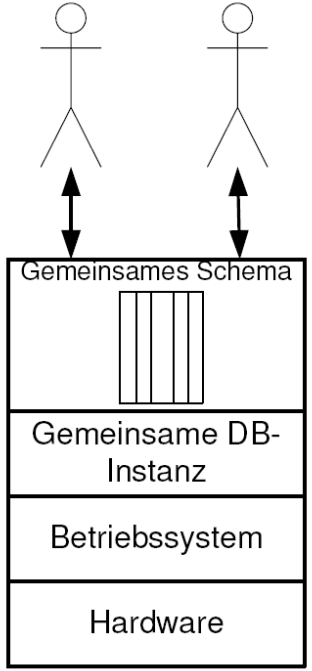
Multi-Tenancy Datenbankarchitekturen



(a)
gemeinsame Maschine
(shared machine)



(b)
gemeinsames Datenbanksystem
(shared process)



(c)
Gemeinsame Relationen
(shared tables)

Shared Tables

Account			
Tenant	**Aid**	Name	...
17	1	Acme	...
17	2	Gump	...
35	1	Ball	...
42	1	Big	...

Die Applikationssoftware muss dann sicherstellen, dass jede Anfrage entsprechend umgeformt wird, je nachdem von welchem Nutzer sie empfangen wurde:

Tenant 35 ⟶ **select** * **from** Account **where** Name **like** 'B%' ⟶ **select** * **from** Account **where** Name **like** 'B%' **and** Tenant = 35

Private Relationen

Account17			
Aid	Name	Hospital	Beds
1	Acme	St. Mary	135
2	Gump	State	1042

Account35	
Aid	Name
1	Ball

Account42		
Aid	Name	Dealers
1	Big	65

Erweiterungs-Relationen

Account_Ext			
Tenant	*Row*	**Aid**	Name
17	0	1	*Acme*
17	1	2	*Gump*
35	0	1	*Ball*
42	0	1	*Big*

Healthcare_Account			
Tenant	**Row**	Hospital	Beds
17	0	*St. Mary*	135
17	1	*State*	1042

Automotive_Account		
Tenant	**Row**	Dealers
42	0	65

Universal Relation

Universal							
Tenant	Table	Col1	Col2	Col3	Col4	Col5	eCol6
17	0	*1*	*Acme*	*St. Mary*	*135*	-	-
17	0	*2*	*Gump*	*State*	*1042*	-	-
35	1	*1*	*Ball*	-	-	-	-
42	2	*1*	*Big*	*65*	-	-	-

Zerlegung: Pivot-Relationen

Pivot_int				
Tenant	**Table**	**Col**	**Row**	Int
17	0	0	0	1
17	0	3	0	135
17	0	0	1	2
17	0	3	1	1042
35	1	0	0	1
42	2	0	0	1
42	2	2	0	65

Pivot_str				
Tenant	**Table**	**Col**	**Row**	Str
17	0	1	0	*Acme*
17	0	2	0	*St. Mary*
17	0	1	1	*Gump*
17	0	2	1	*State*
35	1	1	0	*Ball*
42	2	1	0	*Big*

Ballung logisch verwandter Werte: Chunk Tables

Account_Row			
Tenant	Row	Aid	Name
17	0	1	*Acme*
17	1	2	*Gump*
35	0	1	*Ball*
42	0	1	*Big*

v	
	Name
	Acme
	Gump
	Ball
	Big

Chunk_Row					
Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	135	*St. Mary*
17	0	0	1	1042	*State*
42	2	0	0	65	-

Key/Value-Store

HBase Key/Value-Store		
Row Key	Account	Contact
17Act1	[name:*Acme*, hospital:*St. Mary*, beds:*135*]	
17Act2	[name:*Gump*, hospital:*State*, beds:*1042*]	
17Ctc1		[. . .]
17Ctc2		[. . .]
35Act1	[name:*Ball*]	
35Ctc1		[. . .]
42Act1	[name:*Big*, dealers:*65*]	

XML-basiertes Schema

Account			
Tenant	**Aid**	Name	Ext_XML

17	1	*Acme*	`<ext>`
			`  <hospital>St. Mary</hospital>`
			`  <beds>135</beds>`
			`</ext>`

17	2	*Gump*	`<ext>`
			`  <hospital>State</hospital>`
			`  <beds>1042</beds>`
			`</ext>`

35	1	*Ball*	

42	1	*Big*	`<ext>`
			`  <dealers>65</dealers>`
			`</ext>`