Vorlesung

Cloud- und Web-Technologien

(CS3140)

Operatoren der Relationalen Algebra in MapReduce

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Chronologische Übersicht über die Themen

Nr	Thema	
1	Einleitung	
2	Einführung in das Semantic Web, RDF und SPARQL	 Datenmodell
3	Die Semantic Web-Ontologiesprachen RDFS und OWL	
4	Multiplattform-Entwicklung mit Kotlin	 Multiplattform
5	Fortgeschrittene Themen mit Kotlin	
6	Einstieg in Cloud Computing, Hadoop	 Backend
7	**Operatoren der relationalen Algebra in Hadoop**	
8	Datenverarbeitung mit Pig	
9	Einführung in Spark und Flink	
10	Stromverarbeitung mit Flink	
11	Knotenzentrische Algorithmen mit Flink	
12	HTML und CSS	 Web
13	Browserprogrammierung mit JS/JQuery und Serverprogrammierung mit PHP Hypertext Preprocessor	
14	Zusammenfassung und Ausblick	

Relationale Algebra

- Grundlage von SQL
- Jede Anfragesprache sollte mindestens die Mächtigkeit der Relationalen Algebra haben
  => Relationale Vollständigkeit
- Aber: Relationale Algebra nicht Turing-vollständig
  - Keine Möglichkeit zur Berechnung der transitiven Hülle
  - Fehlen von Rekursion
  - ...
    Durch diese Einschränkung besser optimierbar!

Kriterien für Anfragesprachen 1/3

- Ad-Hoc-Formulierung
  - Einfache Formulierung der Anfrage
  - Ohne ein ganzes Programm formulieren zu müssen
- Deskriptivität/Deklarativität
  - Deklarativ: „Was?“ und nicht „Wie?“
- Optimierbarkeit
  - Anfrage kann auf wenige Basisoperationen abgebildet werden, für die es Optimierungsregeln gibt

Kriterien für Anfragesprachen 2/3

- Mengenorientiert
  - Operationen auf Mengen von Daten
- Abgeschlossenheit
  - Ergebnis ist wieder eine Relation und kann als Eingabe für nächste Anfrage dienen
    ($\longrightarrow$ Verschachtelbare Ausdrücke)
- Adäquatheit
  - Unterstützung aller Konstrukte des darunterliegenden Datenmodells
- Orthogonalität
  - Sprachkonstrukte sind in ähnlichen Situationen auch ähnlich anwendbar

Kriterien für Anfragesprachen 3/3

- Effizienz
  - Effiziente Ausführbarkeit jeder Operation
  - Relationales Modell: $\leq O(n^2)$, n Anzahl Tupel der Eingangsrelation
- Sicherheit
  - Syntaktisch korrekte Anfragen liefern endliche Ergebnisse und terminieren (keine Endlosschleife und keine unendlichen Ergebnisse)
- Vollständigkeit
  - Sprache muss mindestens so mächtig wie eine Standardsprache sein (z.B. relationale Algebra)

Relationale Algebra - Operatoren

Minimale Menge von Operatoren

(es existieren auch andere minimale Mengen)

Operator	Symbol
Projektion	π
Selektion	σ
Kreuzprodukt	$\times$
Vereinigung	$\cup$
Differenz	$-$
Umbenennung	ρ

Weitere Operatoren

Operator	Symbol
Durchschnitt	$\cap$
Division/Quotient	$\div$
Join	$\bowtie$

Erweiterte Relationale Algebra

(Außer Semi-Join ($\rtimes/\ltimes$) und MIN/MAX nicht durch Operatoren der minimalen Menge darstellbar)

Operator	Symbol
Semi Join	$\rtimes$, $\ltimes$
Äußere Joins	$\bowtie$, $\bowtie$, $\bowtie$
Duplikateliminierung	δ
Aggregation	$SUM, AVG, MIN, MAX, COUNT, \ldots$
Gruppierung	γ
Sortierung	τ
Erweiterte Projektion	π
Äußere Vereinigung	$\uplus$

Mengen versus Multimengen

- Menge
  - Jedes Element ist höchstens einmal enthalten
  - Reihenfolge spielt keine Rolle
  - SQL mit Duplikateliminierung (DISTINCT)
- Multimenge
  - Jedes Element kann mehrfach vorkommen
  - Reihenfolge spielt keine Rolle
  - SQL ohne Duplikateliminierung (ohne DISTINCT)
  - Meist effizienter, da aufwändige Duplikateliminierung entfällt (falls gewünscht Duplikateliminierung evtl. nur einmal abschließend)
- Operatoren der Relationalen Algebra
  - Traditionell Mengensemantik
  - Zusätzlich Vorstellung der Multimengensemantik

Selektion σ_C 1/2

- $\sigma_C(R) := \{t | t \in R \wedge C(t)\}$ mit
  - R Relation
  - C Selektionsbedingung:
    - $l \Theta r$ mit l und r Konstanten oder Attribute aus R, $\Theta \in \{=, \neq, <, \leq, >, \geq\}$ bei geordneten Wertebereichen, sonst $\Theta \in \{=, \neq\}$
    - $C_1 \wedge C_2, C_1 \vee C_2, \neg C_1$ mit C_1, C_2 Selektionsbedingungen
    - $true, false$

Selektion σ_C 2/2

Eingangsrelation R

$R_E := \sigma_{Preis \geq 500 \wedge Lieferzeit \leq 3}(Teil)$

$Teil$

PID	Bez.	Liefer-zeit	Preis
22	PC1	5	1000
23	PC2	3	520
24	PC3	1	100

R_E

PID	Bez.	Liefer-zeit	Preis
23	PC2	3	520

Mengensemantik: Duplikate können nicht entstehen, da Eingangsrelation keine Duplikate enthalten kann
Multimengensemantik: Evtl. vorhandene Duplikate bleiben bestehen

- MapReduce
  - Map
    - $\forall t \in R$ mit $C(t) = true$: Emit (t, t) as key/value-pair

Duplikateliminierung δ

- Wandelt Multimenge in Menge um
  - SQL: Mengensemantik bei DISTINCT meist durch genau eine finale Duplikateliminierung realisiert
- $\delta(R) := \{ t | t \in R \}$ mit
  - R Relation
- MapReduce
  - Map
    - $\forall t \in R$: Emit (t, t) as key/value-pair
    - gut mit anderen Operatoren (wie Projektion π oder Selektion σ) mit nur einer Map-Phase kombinierbar
  - Reduce
    - for each fetched $(t, [t, ..., t])$: Emit (t, t) as key/value-pair

Projektion π 1/2

- $\pi_{A_1,...,A_k}(R) := \{(t[A_1], ..., t[A_k]) | t \in R\}$ mit
  - R Relation
  - $A_1, ..., A_k$ Attribute aus R

Eingangsrelation R

BestID	Datum	PID → Teil	KID → Kunde
1	2.1.18	10	14
2	3.1.18	12	15
3	4.1.18	10	14

$R_E := \pi_{PID,KID}(Bestellung)$

Mengensemantik

PID → Teil	KID → Kunde
10	14
12	15

Multimengensemantik

PID → Teil	KID → Kunde
10	14
12	15
10	14

Projektion π 2/2

- $\pi_{A_1,...,A_k}(R) := \{(t[A_1], ..., t[A_k]) | t \in R\}$ mit
  - R Relation
  - $A_1, ..., A_k$ Attribute aus R
- MapReduce
  - Map
    - $\forall t \in R$: Emit $((t[A_1], ..., t[A_k]), (t[A_1], ..., t[A_k]))$ as key/value-pair
  - Reduce
    - Bei Mengensemantik Reduce-Schritt der Duplikateliminierung
    - Bei Multimengensemantik entfällt Reduce-Schritt

Vereinigung $\cup$ 1/2

- $R \cup S = \{t | t \in R \vee t \in S\}$

Mengensemantik

	R	S	$R \cup S$
Tupel in	✓		✓
		✓	✓
	✓	✓	✓

Bsp.:

R

A	B
1	2
3	4

S

A	B
3	4
5	6

$R \cup S$

A	B
1	2
3	4
5	6

Multimengensemantik

	R	S	$R \cup S$
Tupel in	n-mal	m-mal	(n+m)-mal

Bsp.: $R \cup S$

A	B
1	2
3	4
3	4
5	6

Vereinigung $\cup$ 2/2

- $R \cup S = \{t | t \in R \vee t \in S\}$
- Hadoop MapReduce unterstützt in der Map-Phase das Lesen von mehreren Eingaben gleichzeitig
- MapReduce
  - Map
    - $\forall t \in R$: Emit (t, t) as key/value-pair
    - $\forall t \in S$: Emit (t, t) as key/value-pair
  - Reduce
    - Bei Mengensemantik Reduce-Schritt der Duplikateliminierung
    - Bei Multimengensemantik entfällt Reduce-Schritt

Durchschnitt $\cap$ 1/2

- $R \cap S = \{t | t \in R \wedge t \in S\}$

Mengensemantik

	R	S	$R \cap S$
Tupel in	✓		
		✓	
	✓	✓	✓

Bsp.:

R

A	B
1	2
3	4

S

A	B
3	4
5	6

$R \cap S$

A	B
3	4

Multimengensemantik

	R	S	$R \cap S$
Tupel in	n-mal	m-mal	min(n, m)-mal

Bsp.:

R

A	B
1	2
1	2
3	4

S

A	B
1	2
1	2
1	2
3	4
5	6

$R \cap S$

A	B
1	2
1	2
3	4

Durchschnitt $\cap$ 2/2

- $R \cap S = \{t | t \in R \wedge t \in S\}$
- Hadoop MapReduce unterstützt in der Map-Phase das Lesen von mehreren Eingaben gleichzeitig
- MapReduce
  - Map
    - $\forall t \in R$: Emit $(t, "R")$ as key/value-pair
    - $\forall t \in S$: Emit $(t, "S")$ as key/value-pair
  - Reduce
    - Mengensemantik:
      - for each fetched $(t, [a_1, ..., a_n])$ mit $\exists i, j \in \{1, ..., n\}$: $a_i = "R" \wedge a_j = "S"$: Emit (t, t) as key/value-pair
    - Multimengensemantik:
      - for each fetched (t, A) mit $A := [a_1, ..., a_n]$: $r := |\{a | a \in A \wedge a = "R"\}| \wedge s := |\{a | a \in A \wedge a = "S"\}|$: $min(r, s)$-times Emit (t, t) as key/value-pair

Differenz — 1/2

- $R - S = \{t | t \in R \wedge t \notin S\}$

Mengensemantik

	R	S	$R - S$
Tupel in	✓		✓
		✓	
	✓	✓	

Bsp.:

R

A	B
1	2
3	4

S

A	B
3	4
5	6

$R - S$

A	B
1	2

$S - R$

A	B
5	6

Multimengensemantik

	R	S	$R - S$
Tupel in	n-mal	m-mal	max(0, n-m)-mal

"Jedes Tupel t in S eliminiert ein t in R"

Bsp.:

R

A	B
1	2
1	2
3	4
7	8

S

A	B
1	2
1	2
1	2
3	4
5	6

$R - S$

A	B
7	8

$S - R$

A	B
1	2
5	6

Differenz — 2/2

- $R - S = \{t | t \in R \wedge t \notin S\}$
- Hadoop MapReduce unterstützt in der Map-Phase das Lesen von mehreren Eingaben gleichzeitig
- MapReduce
  - Map
    - $\forall t \in R$: Emit $(t, "R")$ as key/value-pair
    - $\forall t \in S$: Emit $(t, "S")$ as key/value-pair
  - Reduce
    - Mengensemantik:
      - for each fetched $(t, [a_1, ..., a_n])$ mit $\exists i \in \{1, ..., n\}: a_i = "R" \wedge \nexists j \in \{1, ..., n\}: a_j = "S"$: Emit (t, t) as key/value-pair
    - Multimengensemantik ("Jedes Tupel t in S eliminiert ein t in R"):
      - for each fetched (t, A) mit $A := [a_1, ..., a_n]$: $r := |\{a | a \in A \wedge a = "R"\}| \wedge s := |\{a | a \in A \wedge a = "S"\}|$: $max(r - s, 0)$-times Emit (t, t) as key/value-pair

Kartesisches Produkt $\times$

$h(r) \setminus h(s)$	0	1	...	$n-1$
0		1		
1	n	$n+1$	...	$2 \times n - 1$
...		...		
$n-1$		$(n-1) \times n + 1$		

Verteilungsschema für $r \in R$ mit $h(r) = 1$ (Zeile 1) bzw. für $s \in S$ mit $h(s) = 1$ (Spalte 1)

- MapReduce
  - Map
    - $\forall t \in R : \forall i \in \{0, ..., n-1\}$: Emit $(h(t) \times n + i, (t, "R"))$
    - $\forall t \in S : \forall i \in \{0, ..., n-1\}$: Emit $(h(t) + i \times n, (t, "S"))$
  - Reduce
    - Multi-/Mengensemantik:
      - for each fetched $(t, [a_1, ..., a_n])$ mit $\exists i, j \in \{1, ..., n\}: a_i = (r, "R") \wedge a_j = (s, "S")$: Emit $(r \circ s, r \circ s)$ as key/value-pair

Equi-Join $\bowtie$ 1/2

- $R \bowtie_C S = \sigma_C(R \times S)$ mit C ist Joinbedingung von der Form:
  - $r = s$ mit r Attribut aus R und s Attribut aus S
  - $C_1 \wedge C_2$ mit C_1 und C_2 Joinbedingungen

Mengensemantik

Bsp.:

R

A	B
1	2
0	2
3	4
7	8

S

B	C
2	3
2	4
5	6
4	5

$R \bowtie_{R.B=S.B} S$

A	R.B	S.B	C
1	2	2	3
0	2	2	3
1	2	2	4
0	2	2	4
3	4	4	5

Multimengensemantik

Bsp.:

R

A	B
1	2
0	2
7	8
3	4

S

B	C
2	3
4	5
4	5
4	5
5	6

$R \bowtie_{R.B=S.B} S$

A	R.B	S.B	C
1	2	2	3
0	2	2	3
3	4	4	5
3	4	4	5
3	4	4	5

Equi-Join $\bowtie$ 2/2

- $R \bowtie_C S = \sigma_C(R \times S)$ mit C ist Joinbedingung von der Form:
  - $r = s$ mit r Attribut aus R und s Attribut aus S
  - $C_1 \wedge C_2$ mit C_1 und C_2 Joinbedingungen
- MapReduce
  - Map ($C := r_1 = s_1 \wedge ... \wedge r_n = s_n$)
    - $\forall t \in R$: Emit $((t[r_1], ..., t[r_n]), (”R”, t[1], ..., t[|t|]))$ as key/value-pair
    - $\forall t \in S$: Emit $((t[s_1], ..., t[s_n]), (”S”, t[1], ..., t[|t|]))$ as key/value-pair
  - Reduce
    - for each fetched (j, A) : $\forall (”R”, r[1], ..., r[n]) \in A$:
      $\forall (”S”, s[1], ..., s[m]) \in A : t := (r[1], ..., r[n], s[1], ..., s[m])$:
      Emit (t, t) as key/value-pair

Semi-Join $\ltimes$ 1/2

- $R \ltimes_C S = \pi_{\text{Attribute aus } R}(R \bowtie_C S)$ mit C ist Joinbedingung von der Form:
  - $r = s$ mit r Attribut aus R und s Attribut aus S
  - $C_1 \wedge C_2$ mit C_1 und C_2 Joinbedingungen

Mengensemantik

Bsp.: R

A	B
1	2
0	2
3	4
7	8

S

B	C
2	3
2	4
5	6
4	5

$R \ltimes_{R.B=S.B} S$

A	B
1	2
0	2
3	4

Multimengensemantik

Bsp.: R

A	B
1	2
0	2
7	8
3	4

S

B	C
2	3
4	5
4	5
4	5
5	6

$R \ltimes_{R.B=S.B} S$

A	B
1	2
0	2
3	4
3	4
3	4

Semi-Join $\ltimes$ 2/2

- $R \ltimes_C S = \pi_{\text{Attribute aus } R}(R \bowtie_C S)$ mit C ist Joinbedingung von der Form:
  - $r = s$ mit r Attribut aus R und s Attribut aus S
  - $C_1 \wedge C_2$ mit C_1 und C_2 Joinbedingungen
- MapReduce
  - Map ($C := r_1 = s_1 \wedge ... \wedge r_n = s_n$)
    - $\forall t \in R$: Emit $((t[r_1], ..., t[r_n]), (”R”, t[1], ..., t[|t|]))$ as key/value-pair
    - $\forall t \in S$: Emit $((t[s_1], ..., t[s_n]), (”S”, t[1], ..., t[|t|]))$ as key/value-pair
  - Reduce
    - for each fetched (j, A) : $\forall (”R”, r[1], ..., r[n]) \in A$: $\forall (”S”, s[1], ..., s[m]) \in A$: $t := (r[1], ..., r[n])$: Emit (t, t) as key/value-pair
    - Bei Mengensemantik nur $1\times$ Versenden von (t, t)

Äußere Joins $⟕$, $⋈$, $⟖$ - 1/2

- „Ergebnis des Joins plus die Tupel aus einer Eingangsrelation ohne Joinpartner“
- $R ⟕ S = (R - (R ⋉ S)) ⊎ (R ⋈ S)$
- $R ⟖ S = (R ⋈ S) ⊎ (S - (S ⋉ R))$
- $R ⟗ S = (R - (R ⋉ S)) ⊎ (R ⋈ S) ⊎ (S - (S ⋉ R))$

Mengensemantik

Bsp.:

R

A	B
1	2
0	2
3	4
7	8

S

B	C
2	3
2	4
4	5
5	6

$R ⟕ S$

A	B	C
1	2	3
0	2	3
1	2	4
0	2	4
3	4	5
	5	6
7	8	

Multimengensemantik

Bsp.:

R

A	B
1	2
0	2
3	4
7	8

S

B	C
2	3
4	5
4	5
4	5
5	6

$R ⟕ S$

A	B	C
1	2	3
0	2	3
3	4	5
3	4	5
3	4	5
	5	6
7	8	

Äußere Joins ⟗, ⟕, ⟖ - 2/2

- MapReduce
  - Map ($C := r_1 = s_1 \wedge ... \wedge r_n = s_n$)
    - $\forall t \in R$: Emit $((t[r_1], ..., t[r_n]), (”R”, t[1], ..., t[|t|]))$ as key/value-pair
    - $\forall t \in S$: Emit $((t[s_1], ..., t[s_n]), (”S”, t[1], ..., t[|t|]))$ as key/value-pair
  - Reduce
    - for each fetched (j, A) : $\forall (”R”, r[1], ..., r[n]) \in A$:
      $\forall (”S”, s[1], ..., s[m]) \in A : t := (r[1], ..., r[n], s[1], ..., s[m])$:
      Emit (t, t) as key/value-pair
    - Voller (⟗) und Linker (⟕) äußerer Join: for each fetched (j, A) :
      $\nexists (”S”, s[1], ..., s[m]) \in A \wedge \forall (”R”, r[1], ..., r[n]) \in A$:
      $t := (r[1], ..., r[n])$: Emit (t, t) as key/value-pair
    - Voller (⟗) und Rechter (⟖) äußerer Join: for each fetched (j, A) :
      $\nexists (”R”, r[1], ..., r[n]) \in A \wedge \forall (”S”, s[1], ..., s[m]) \in A$:
      $t := (s[1], ..., s[m])$: Emit (t, t) as key/value-pair

Gruppierung γ 1/3

- Partitionierung der Tupel einer Relation (in „Gruppen“) gemäß ihrer Werte in einem oder mehreren Attributen
  - Hauptzweck: Aggregation auf den jeweiligen Gruppen
- $\gamma_{(F_1,\ldots,F_n)}(R)$, wobei F_i von folgender Form ist:
  - ein Attribut von R
  - ein Ausdruck $X \rightarrow Y$, wobei Y ein neuer Attributname ist, und X von folgender Form:
    - ein Attribut von R
    - eine Aggregationsfunktion (SUM, $COUNT$, AVG, MIN, MAX, …)
    - eine (verschachtelbare) Berechnungsvorschrift (arithmetische Formel, String-Operation, Klammern, …)
  - Ist kein F_i angegeben, so ist ganz R die (einzige) Gruppe

Gruppierung γ 2/3

- $\gamma_{(A_1,\ldots,A_k,F_1,\ldots,F_n)}(R)$, wobei A_i Attribute aus R sind und F_i Zuweisungen von Ausdrücken der Form $X \rightarrow Y$ sind

Bestellung

BestID	Datum	PID → Teil	KID → Kunde
1	2.1.18	10	14
2	3.1.18	12	15
3	4.1.18	10	14

$\gamma_{KID,COUNT(PID)\rightarrow NR,MAX(Datum)\rightarrow MaxDatum}(Bestellung)$

KID	Nr	MaxDatum
14	2	4.1.18
15	1	3.1.18

Gruppierung γ 3/3

- $\gamma_{(A_1,\ldots,A_k,F_1,\ldots,F_n)}(R)$, wobei A_i Attribute aus R sind und F_i Zuweisungen von Ausdrücken der Form $X \rightarrow Y$ sind
- MapReduce
  - Map
    - $\forall t \in R$: Emit $((t[A_1], \ldots, t[A_k]), t)$ as key/value-pair
  - Reduce
    - for each fetched $((t[A_1], \ldots, t[A_k]), [t_1, \ldots, t_m])$:
      $t' := (t[A_1], \ldots, t[A_k], F_1(t_1, \ldots, t_m), \ldots, F_n(t_1, \ldots, t_m))$:
      Emit (t', t') as key/value-pair

Sortierung τ 1/2

- Ergebnis von $\tau_{A_1,\ldots,A_k}(R)$ ist eine Liste von Tupeln aus R, die primär nach A_1, sekundär nach A_2 usw. sortiert ist

Bestellung

BestID	Datum	PID → Teil	KID → Kunde
1	2.1.18	10	14
2	3.1.18	12	15
3	4.1.18	10	14

$\tau_{KID,Datum}$ (Bestellung)

BestID	Datum	PID → Teil	KID → Kunde
1	2.1.18	10	14
3	4.1.18	10	14
2	3.1.18	12	15

- Wichtig:
  - Ergebnis der Sortierung ist keine (Multi-)Menge, sondern eine Liste
  - Sortieroperation immer letzte Operation, da ansonsten wieder (Multi-) Mengen entstehen
  - In DBMS wird häufig auch zwischendurch sortiert (oder auf vorsortierte Eingaben aus einem Index zugegriffen), um optimierte Algorithmen anzuwenden, die sortierte Eingaben erfordern

Sortierung τ 2/2

- Ergebnis von $\tau_{A_1,\ldots,A_k}(R)$ ist eine Liste von Tupeln aus R, die primär nach A_1, sekundär nach A_2 usw. sortiert ist
- MapReduce
  - Map
    - $\forall t \in R$: Emit $((t[A_1], \ldots, t[A_k]), t)$ as key/value-pair
  - Reduce
    - MapReduce ordnet die Reducer nach der Ordnung der Schlüssel der Eingabe, d.h. Reducer i erhält nur Schlüssel-Wert-Paare, deren Schlüssel kleiner sind als die von Reducer $i+1$

SPARQL-Anfrage in der Cloud

- Welche Map- und Reduce-Phasen sind notwendig zum Verarbeiten folgender SPARQL-Anfrage?

```
PREFIX …
SELECT ?person ?name
WHERE {
  ?article rdf:type bench:Article .
  ?article dc:creator ?person .
  ?person  foaf:name  ?name .
}
ORDER BY ?name
```

Annahme:
Durch geschickte Verwendung von Indexen können die Ergebnisse der Tripelmuster mit einem Indexzugriff erfolgen.

Zusammenfassung

- Relationale Algebra als Basis von vielen Anfragesprachen
- Operatoren der Relationalen Algebra als Map/Reduce-Jobs ausführbar
- Komplexe Anfragen erfordern Hintereinanderschaltung mehrerer Map/Reduce-Jobs