Relationen-Algebra

Relationale Anfragesprachen

- Query Language (QL): Manipulation und Retrieval von Daten einer Datenbank
- Relationenmodell erlaubt einfache, mächtige Anfragesprachen
  - Strenge Formale Grundlagen (basiert auf Logik)
  - Erlauben Optimierung
- QL != Programmiersprache
  - Unterstützen einfachen, effizienten Zugriff auf große Datenmengen
  - Nicht geeignet für komplexe Berechnungen
  - Nicht “Turing-vollständig“
- Unterstützung verschiedener Benutzerklassen
  - Anwendungsprogrammierer
  - DBA
  - Anspruchsvolle Laien
- Vereinheitlichte Sprache angestrebt für:
  - Ad-hoc Anfragen
  - Datenmanipulation und Anfragen, eingebettet in eine Wirtssprache
  - Datendefinition
  - Zugriffs- und Integritätskontrolle

Formale Relationale Anfragesprachen

- 2 mathematische Query-Languages bilden die Basis für “reale“ Sprachen (z.B. SQL)
- Relationenalgebra: operational (gut geeignet für die Darstellung von Ausführungsplänen)
- Relationenkalkül: deklarativ, Beschreibung des Ergebnisses (eher WAS als WIE)
- Algebra und Kalkül sind Grundlagen für das Verständnis von SQL
- Voraussetzung:
  - Anfragen werden auf Instanzen von Relationen (d.h. Mengen von Tupeln angewandt)
  - Resultat ist auch eine Instanz einer Relation (d.h. Menge von Tupeln) = Abgeschlossenheitseigenschaft
  - Schema der Eingaberelation ist fest
  - Schema des Ergebnisses einer Query ist festgelegt durch die Definition in der Anfrage
- Notationen (Identifizierung der Attribute)
  - Position (einfacher für formale Definition)
  - Feldnamen (besser lesbar)

Beispiel-Relationen

- Relation S1 und S2 (Segler)
- Relation R1

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relationenalgebra

- Basis-Operationen:
  - *Selektion* (σ) Auswahl einer Teilmenge von Tupeln aus der Relation
  - *Projektion* (π) Auswahl von Spalten aus einer Relation
  - *Kreuzprodukt* ($\times$) Kombination zweier Relationen
  - *Mengendifferenz* (-) Menge der Tupel aus Relation 1, aber nicht in Relation 2
  - *Vereinigung* ($\cup$) Menge der Tupel aus Relation 1 und Relation 2
- Zusätzliche Operationen:
  - Durchschnitt, *Join*, Division, Umbenennung (nicht wesentlich, aber hilfreich)
- Kombination der Operationen möglich, weil jede Operation eine Relation liefert (Algebra ist abgeschlossen)

Klassische Mengenoperationen

- Vereinigung (UNION) von R und S

  $R \cup S = \{ t \mid t \in R \vee t \in S \}$
- Differenz von R und S

  $R - S = \{ t \mid t \in R \wedge t \notin S \}$
- Durchschnitt (INTERSECTION) von R und S

  $R \cap S = R - (R - S)$

  $= \{ t \mid t \in R \wedge t \in S \}$

- Alle diese Operationen benötigen zwei Eingabe-Relationen, die kompatibel sein müssen
  - Gleiche Anzahl von Attributen
  - Korrespondierende Attribute haben denselben Typ

Selektion

- Formale Definition
  - Auswahl von Tupeln einer Relation über Prädikate
  - abgekürzt σ_p ,$T= \sigma_p (R) = \{ t \mid t \in R \wedge p(t) \}$
  - P = logische Formel, zusammengesetzt aus
    - Arithmetische Vergleichsoperatoren

      $<, =, >, \leq, \geq, \neq$
    - Logische Vergleichsoperatoren

      $\neg, \wedge, \vee$
    - Operanden:

      Konstanten,

      Spalten-Nr. / Attributname
- Keine Duplikate im Ergebnis
- Schema der Ergebnisrelation = Schema der Eingangsrelation
- Ergebnisrelation kann Input für eine weitere relationale Operation sein (Komposition von Operatoren)

Projektion

- Formale Definition
  - Auswahl der Spalten mit den Nr. j1, j2 ..., jm $\in$ { 1,2, ,,,, n } aus einer Relation R, Grad n
  - $L = (j_k \mid k = 1, ..., m \}$
  - $P = \pi_L (R)$

    $= \{ p \mid \exists t \in R: p = t[j1], t[j2], ..., t[jm] \}$
- Ausblenden von Attributen, die nicht in der Projektionsliste stehen
- Ergebnisschema enthält genau die Attribute aus der Projektionsliste mit demselben Namen wie in der Eingangsrelation
- Projektion eliminiert Duplikate (Mengeneigenschaft von Relationen) !!
  - Reale Systeme (relationale DBMS) tun das nicht, sondern es muß explizit angegeben werden (vgl. DISTINCT-Klausel in SQL)

Beispiele: Projektion, Selektion

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname,rating}(S2)$

age
35.0
55.5

$\pi_{age}(S2)$

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$\sigma_{rating>8}(S2)$

sname	rating
yuppy	9
rusty	10

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

Beispiele: Mengenoperationen

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

Kreuzprodukt

- Jedes Tupel aus Relation R (Grad r) wird mit jedem Tupel aus Relation S (Grad s) kombiniert
- $K = R \times S = \{ k \mid \exists x \in R, y \in S : (k = x \mid y) \}$

  $k = x \mid y = \langle x_1 \ldots x_r, y_1, \ldots, y_s \rangle$
- Ergebnisschema hat ein Attribut für jedes Attribut aus R und S (konfligierende Namen müssen durch Umbenennung aufgelöst werden)

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

- Konflikt: R und S haben ein Attribut *sid.*

➢ *Rename*: $\rho(K(1 \rightarrow sid1, 5 \rightarrow sid2), R \times S)$

Joins

- Kartesisches Produkt zwischen zwei Relationen R (Grad r) und S (Grad s), eingeschränkt durch t-Bedingung zwischen i-Spalte von R und j-Spalte von S. Jedes Tupel aus Relation R (Grad r) wird mit jedem Tupel aus Relation S (Grad s) kombiniert (allgemein: Theta-Join)
- Sei $\Theta = \{<, =, >, \geq, \leq, \neq\}$

$$R \bowtie_{i \Theta j} S = \sigma_{i \Theta r+j}(R \times S)$$

- Statt i und j: Verwendung von Attributnamen

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- Ergebnisschema wie bei Kreuzprodukt
- Weniger Tupel als bei Kreuzprodukt (effizientere Berechnung möglich)

Equi-Join und Natural Join

- Equi-Join ist ein Spezialfall mit Θ = ‘=‘

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$R1 =\bowtie_{sid} S1$$

- Natural Join:
  - Gleichverbund über alle gleichen Attribute und Projektion über die verschiedenen Attribute (bei Benennung von Spalten)
  - Wird wie folgt berechnet:

    Man bildet das kartesische Produkt R $\times$ S. Für jedes Attribut, das sowohl in R als auch in S erscheint, selektiert man die Tupel, für die die Werte der gleichnamigen Attribute übereinstimmen. Eine der gleichen Spalten wird ausprojektiert
- Ein Join zwischen R und S heißt verlustfrei, wenn alle Tupel von R und S am Join teilnehmen. Die inverse Operation Projektion erzeugt dann wieder R und S (lossless join).

Outer Join

- Ziel: Verlustfreier Verbund soll erzwungen werden
- Anwendung: Darstellung komplexer Objekte durch Relationen
- Beim Zusammensetzen sollen auch Teilobjekte als Ergebnis geliefert werden
- Trick: Einfügen von NULL-Werten zur künstlichen Erzeugung von Verbund-Partnern
- 3 Arten von Outer Joins
  - Left Outer Join $\supset\!\bowtie$

    Linke Argumentrelation bleibt verlustfrei, d.h. bei Bedarf wird ein Tupel durch NULL-Werte “nach rechts“ aufgefüllt

    Analog wie Left, rechte Relation bleibt verlustfrei
  - Right Outer Join $\bowtie\!\subset$
  - Full Outer Join $\supset\!\bowtie\!\subset$

    Beide Relationen bleiben verlustfrei, fehlende Tupel werden “nach links“ und “nach rechts“ aufgefüllt

Outer Join: Beispiele

Left Outer Join

R	A	B	C
	1	1	1
	2	2	2

S	C	D	E
	1	1	1
	3	2	2

ERG	A	B	C	D	E
	1	1	1	1	1
	2	2	2	--	--

Right Outer Join

R	A	B	C
	1	1	1
	2	2	2

S	C	D	E
	1	1	1
	3	2	2

ERG	A	B	C	D	E
	1	1	1	1	1
	--	--	3	2	2

Full Outer Join

R	A	B	C
	1	1	1
	2	2	2

S	C	D	E
	1	1	1
	3	2	2

ERG	A	B	C	D	E
	1	1	1	1	1
	2	2	2	--	--
	--	--	3	2	2

Division*

- Kein primitiver Operator, aber nützlich für bestimmte Queries wie z.B.

  "Finde Segler, die alle Boote reserviert haben"

- Es sei A mit 2 Attributen, x und y; B mit nur einem Attribut y:
  - A/B = $\{\langle x \rangle \mid \exists \langle x, y \rangle \in A \ \forall \langle y \rangle \in B\}$
- A/B enthält alle x-Tupel ("Segler"), so daß für jedes y-Tupel ("Boot") in B ein xy-Tupel in A existiert
- Oder: Wenn die Menge der y-Werte (Boote), die mit einem x-Wert (Segler) in A assoziiert ist, alle y-Werte in B enthält, so befindet sich der x-Wert in A/B.
- Im allgemeinen können x und y irgendwelche Listen von Attributen sein; y ist die Liste der Attribute in B, und x $\cup$ y ist die Liste der Attribute in A

Division A/B: Beispiele*

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

sno
s1
s2
s3
s4

A/B1

pno
p2
p4

B2

sno
s1
s4

A/B2

pno
p1
p2
p4

B3

sno
s1

A/B3

Beispiel-Anfragen (1)

"Finde die Namen aller Segler, die das Boot #103 reserviert haben"

Lösung 1:

$$\pi_{sname}((\sigma_{bid=103}\text{Reserves}) \bowtie Sailors)$$

Lösung 2:

$$\rho(Temp1, \sigma_{bid=103}\text{Reserves})$$

$$\rho\ (Temp2,\ Temp1 \bowtie Sailors)$$

$$\pi_{sname}\ (Temp2)$$

Lösung 3:

$$\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$$

Beispiel-Anfragen (2)

“Finde die Namen aller Segler, die ein rotes Boot reserviert haben“

Informationen über die Boot-Farbe nur in *Boats* verfügbar, somit zusätzlicher Join erforderlich

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie \mathrm{Re}serves \bowtie Sailors)$$

Effizientere Lösung:

$$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'} Boats) \bowtie \mathrm{Re}s) \bowtie Sailors)$$

Solche effizienten Lösungen werden durch Query-Optimierer gefunden, denen als Input zunächst eine (nicht-optimale) Lösung gegeben wird

Beispiel-Anfragen (3)

“Finde die Namen aller Segler, die ein rotes oder ein grünes Boot reserviert haben“

Zunächst Bestimmung aller roten und grünen Boote, dann Ermitteln der Segler, die eines dieser Boote reserviert haben

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie \mathrm{Re}serves \bowtie Sailors)$$

Tempboats könnte auch durch Union gebildet werden

$$\rho(Tempboats, (\sigma_{color='red'}\ Boats) \cup (\sigma_{color='green'}\ Boats))$$

Vorsicht bei der Verwendung der logischen Operatoren

Beispiel-Anfragen (4)

"Finde die Namen der Segler, die ein rotes und ein grünes Boot reserviert haben"
Erfordert Überarbeitung gegenüber vorheriger Lösung: Identifiziere die Segler, die rote Boote reserviert haben, die Segler, die grüne Boote reserviert haben, und bilde deren Durchschnitt (*sid* ist der Identifikator eines Seglers)

$$\rho\ (Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves))$$

$$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie Reserves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Beispiel-Anfragen (5)*

“Finde die Namen der Segler, die alle Boote reserviert haben“

Verwendung von Division: Schema der Eingabe-Relationen muß sorgfältig gewählt sein:

$$\rho\ (Tempsids, (\pi_{sid,bid}\mathrm{Re}serves) / (\pi_{bid} Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

“Finde die Segler, die alle “Interlake“-Boote reserviert haben“

$$\ldots / \pi_{bid}(\sigma_{bname='Interlake'} Boats)$$

Zusammenfassung

- Das relationale Modell hat einfache und mächtige Query-Languages definiert
- Relationale Algebra ist eher prozedural; nützlich für die interne Darstellung von Ausführungs-Plänen für Anfragen
- Es gibt unterschiedliche Möglichkeiten, eine Anfrage auszudrücken; ein Query-Optimierer wählt die effizienteste davon aus