

8 Abfrageoptimierung

Wie werden Abfragen evaluiert? Das wird hier näher angeschaut.

Abfrageoptimierung basiert auf Beschreibungsdaten, genannt Metadaten, auf. Diese sind in Systemtabellen (Systemkatalog) gespeichert, und werden benutzt, um zu entscheiden, den besten Weg um die Abfrage zu evaluieren.

SQL Abfragen werden in einer erweiterter Form des relationalen Algebra übersetzt und als Baum dargestellt, samt relationale Operatoren und Labels, die die Algorithmen für jedes Node identifizieren.

Relationale Operatoren sind also die Bausteine der Abfrageevaluation, und ihre Implementation ist sorgfältig optimiert für gute Performanz.

Im Normalfall sind mehrere Operatoren in einer Evaluation involviert, und sie können in verschiedene Wege kombiniert werden, diese Wege werden auch Pläne genannt. Was wir mit Abfrageoptimierung verstehen ist das Prozess, einen guten Plan zu finden (ideal: der beste Plan, in Praxis: die schlimmsten Pläne eliminieren!).

Herausforderungen:

- Für eine bestimmte Abfrage, welche Pläne sollen wir überhaupt betrachten?
- Wie können wir die Kosten eines bestimmten Plan einschätzen?

8.1 Relationale Algebra

Das relationale Algebra ist die formale Sprache, die hinter dem relationalen Modell versteckt ist.

Im rel. Algebra, Abfragen werden als Zusammensetzung von (relationalen) Operatoren angesehen.

Ein relationales Operator nimmt zwei Relationen als Argumente und generiert eine neue Relationsinstanz als Output.

Diese Eigenschaft macht es möglich, dass komplexere Operationen als Zusammensetzung von anderen Operatoren gebaut werden können. Wir sprechen dann von Ausdrücke [expressions] des relationalen Algebras.

Ein relationaler Ausdruck ist rekursiv definiert als:

- Eine Relation (d.h. eine Tabelle), oder
- Ein Operator mit einem einziges Argument, angewendet an ein einziges Ausdruck, oder
- Ein binäres Operator (also mit zwei Argumente), angewendet an zwei Ausdrücke.

Basisoperatoren sind:

- Selektion
- Projektion
- Union
- Kreuzprodukt

- Differenz

Jede relationale Abfrage kann als eine schrittweise durchführbare Prozedur, die auf die mit der Ordnung der Operatoren definiert ist. Es kann auch also als ein Rezept oder **Plan**, betrachtet werden.

Beispiel: man betrachtet die folgende Instanz einer Relation:

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

8.1.1 Projektion:

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname, rating}(S_2)$$

age
35.0
55.5

$$\pi_{age}(S_2)$$

8.1.2 Selektion:

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

$$\sigma_{rating > 8}(S_2)$$

8.1.3 Kombination von Operatoren:

sname	rating
yuppy	9
rusty	10

$$\pi_{sname, rating}(\sigma_{rating > 8}(S_2))$$

8.1.4 Kreuzprodukt:

Das Kreuzprodukt ist das wichtigste der Mengenoperationen: hier das Beispiel von Kreuzprodukt von folgenden Relationen :

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

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

ergibt:

(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

Interessanterweise sieht man hier einen Konflikt: beide S1 x R1 haben ein Feld, das sid genannt ist. Die meiste Zeit müssen die „unsinnige“ Zeilen aufgrund von diesem Konflikt eliminiert werden. Dies führt uns zur Betrachtung einer weiteren Operation, genannt Join.

8.1.5 Konzeptuelle Evaluationsstrategie

Jetzt sind wir so weit, dass wir die konzeptuelle Strategie zur Abfrageevaluation angeben können:

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.rid AND R.bid = 103
```

- Zuerst berechnen wir das Kreuzprodukt von Sailors und Reserves
- Dann streichen wir die Datensätze (also Zeilen), welche die Qualifikation nicht erfüllen
- Dann Streichen wir die Attribute (also Kolonnen), welche nicht abgefragt werden.

Es ist vermutlich jeder und jede klar hier, dass die Abfragen nicht in dieser Art implementiert sein können, da dieses Kreuzprodukt einfach viel zu gross ist. Deshalb werden weitere Operatoren als Basisoperatoren betrachte, die Joins.

8.1.6 Joins

Ein Join kann als Kreuzprodukt, gefolgt von Selektionen und Projektionen. Diese kommen, wie in dem obigen Beispiel ersichtlich, viel öfters als die Kreuzprodukte vor. Dazu ist das Resultat eines Join viel kleiner als das von einem Kreuzprodukt, also es ist sehr wichtig, dass man sie implementiert, ohne das unterliegenden Kreuzprodukt zu materialisieren. Deshalb werden sie meistens als Basisoperation auch implementiert.

Allgemeine Form: Condition Join:

$$R \bowtie_c S = \sigma_c (R \times S)$$

Beispiel:

(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

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

Sehr oft benutzt: **Equijoin**, wobei die Condition c die Gleichheit ist:

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

$$S1 \bowtie_{sid} R1$$

Schlussendlich auch sehr nützlich ist der Natural Join: es ist ein Equijoin wo die Gleichheitsbedingung für alle Felder, die die gleichen Namen haben in beide Relationen, gilt. Unser Beispiele oben ist eigentlich auch ein natural join, weil nur sid in beide Relationen vorkommt.

So können wir auch sehen, dass unsere konzeptuelle Evaluationsstrategie einen natural join, gefolgt von einer Selektion und Projektion, entspricht.

Nach dieser Exkurs können wir weiter auf die Abfrageoptimierung weitergehen.

8.2 Einige gewöhnliche Techniken

Algorithmen zur Evaluation von relationalen Operatoren benutzen einige Techniken intensiv:

- Indizierung: Man kann die WHERE Bedingung benutzen, um kleinere Mengen von Datensätze zu zugreifen, wenn eine Selektion oder ein Join vorkommt.
- Iteration: Manchmal ist es schneller, eine Relation ganz zu durchsuchen, als ein Index zu benutzen (scanning). In dem fall schaut man alle Datensätze, ein nach der anderen. Falls wir nur eine Untermenge von Attribute der gesamten Relation brauchen, und es gibt ein Index mit allen diesen Attribute, kann man diesen Index scannen. Scanning eines Indexes macht keine Gebrauch von der Struktur des Index, z.B. bei einem Baum schaut man nur die Blätter des Baumes an.
- Partitionierung: Wenn wir partitionieren Datensätze auf ein Suchschlüssel, können wir oft eine Operation in kleiner, weniger aufwändige Operationen zerlegen. Sorting und Hashing sind zwei gewöhnliche Partitionierungstechniken.

8.3 Systemkatalog

Um abfragen zu optimieren muss Information über die beteiligten Relationen und Indizes gespeichert sein.

Das Systemkatalog enthält deshalb mindestens:

- Anzahl Datensätze und Anzahl Seiten für jede Relation;
- Anzahl distinkte Schlüsselwerte und Seiten für jedes Index;
- Indexhöhe, min/max Schlüsselwerte für jede Baumindex.

Katalogen werden periodisch aktualisiert, aber Aktualisierung kann nicht nach jeder Wertmodifikation stattfinden. Es ist auch egal, weil es sowieso viele Annäherungen gibt, also kleine Inkonsistenzen sind OK.

Manchmal werden mehr detaillierte Information gespeichert, z.B. Histogramme der Werte in bestimmten Feldern.

8.4 Selektion

8.4.1 Access Paths

Ein **Zugriffspfad** bezeichnet eine Methode, um Datensätze zuzugreifen: es kann entweder ein File scan sein, oder ein Zugriff auf ein Index sein, das zu einer Selektion in einer Abfrage passt [match].

Ein Baum-Index passt zu einer Konjunktion of Terme wenn nur Attribute in einem Prefix des Suchschlüssels vorkommen.

Z.B., Baum Index auf $\langle a, b, c \rangle$ passt zur Selektion $a=5 \text{ AND } b=3$, und zur Selektion $a=5 \text{ AND } b>6$, aber nicht zu $b=3$.

Ein Hash Index passt zu einer Konjunktion von Terme, die ein Term "Attribut = Wert" für jede Attribut in der Suchschlüssel des Index hat

Z.B., Hash index auf $\langle a, b, c \rangle$ passt zu $a=5 \text{ AND } b=3 \text{ AND } c=5$, aber nicht zu $b=3$, oder $a=5 \text{ AND } b=3$, oder zu $a>5 \text{ AND } b=3 \text{ AND } c=5$.

8.4.2 Bemerkung über komplexe Selektionen

Selektionsklauseln werden zuerst in *conjunctive normal form (CNF)* umgeschrieben:

$(\text{day} < 8/9/94 \text{ AND } \text{rname} = \text{'Paul'}) \text{ OR } \text{bid} = 5 \text{ OR } \text{sid} = 3$

wird damit

$(\text{day} < 8/9/94 \text{ OR } \text{bid} = 5 \text{ OR } \text{sid} = 3) \text{ AND } (\text{rname} = \text{'Paul'} \text{ OR } \text{bid} = 5 \text{ OR } \text{sid} = 3)$

Jede Klammer ist Konjunkt genannt

Wir behandeln hier nur die Fälle ohne ORs. Siehe Buch (Kap. 14) für allgemeinere Fälle.

8.4.3 Eine Möglichkeit, um mit Selektionen umzugehen

Finde den meist selektiven Zugriffspfad [*most selective access path*], greife die Datensätze mit dem zu und danach wende aller bleibende Terme an, die dem Index nicht passen, an.

- Meist selektiven Zugriffspfad: Ein Index oder File Scan mit geschätzter kleinsten Page I/Os.
- Terme, die dieser Index passen, reduzieren die Anzahl Datensätze, die gelesen werden müssen. Die weitere Terme werden benutzt, um einige zugegriffene Datensätze zu entfernen, aber sie haben kein Einfluss auf die Anzahl zugegriffenen Seiten.
- Betrachten wir $\text{day} < 8/9/94 \text{ AND } \text{bid} = 5 \text{ AND } \text{sid} = 3$. Ein B+ tree Index auf day kann benutzt werden. Dann müssen die Datensätze für $\text{bid} = 5$ and $\text{sid} = 3$ überprüft werden. In der gleiche Weise, ein has index auf $\langle \text{bid}, \text{sid} \rangle$ kann benutzt werde, $\text{day} < 8/9/94$ muss dann überprüft werden

8.4.4 Benutzung eines Indexes für die Selektion

Die Kosten hängen von Anzahl qualifizierten Typeln, und Clustering.

Die Kosten setzen sich wie folgt zusammen:

- Finde die qualifizierten Datensätze: was typisch klein ist, und
- Greife die Datensätze: was sehr gross sein kann, wenn es kein Clustering gibt.

8.4.5 Beispiel

```
SELECT *
FROM Reserves R
WHERE R.rname < 'C%'
```

Nehmen wir an, wir haben in R 100 Seiten und 10000 Datensätze. Wenn Namen uniform verteilt sind, qualifizieren etwa 10% der Datensätze. Mit einem clustered Index bedeutet es ein wenig mehr als 100 I/Os, mit einem unclustered Index aber bis 10,000 I/Os!

8.5 Projection

Das teuerste ist, die duplizierten Datensätze zu entfernen.

Deshalb entfernen SQL Systeme diese Duplikate nicht automatisch. Das Wort DISTINCT muss in der Abfrage spezifiziert werden.

Methode mit Sortieren: Auf <sid, bid> sortieren, und die duplizierte Werte entfernen. (Man kann das optimieren, in dem man nicht gebrauchte Information auch laufend eliminiert.)

Hashing Approach: Hash on <sid, bid> to create partitions. Load partitions into memory one at a time, build in-memory hash structure, and eliminate duplicates.

Falls es ein Index gibt mit beides R.sid und R.bid im Suchschlüssel, kann es sein, dass es billiger ist, alle Datensätze direkt zu sortieren!

8.6 Joins

Wir betrachten 3 Varianten für die Implementation von Joins:

- Simple Nested Loop:
 - for each tuple r in R do
 - for each tuple s in S do
 - if r.sid = s.sid then add < r, s > to result
- Index Nested Loops:
 - for each tuple r in R do
 - for each tuple s in S where r.sid = s.sid do
 - add < r, s > to result
- Sort Merge Join: man sortiert R und S, und dann scannt man sie um ein „merge“ zu machen (auf der Join Kolonne)

8.7 System R Optimizer

Dieses System ist das meist benutzte. Es läuft bestens mit weniger als 10 Joins.

Einschätzung des Kostens: Kunst der Approximation! Es basiert auf statistische Daten, die in dem System Catalog verwaltet sind. Diese werden benutzt, um

einzuschätzen, wieviel gewisse Operationen kosten und wie gross die Ergebnisse von Operationen sein werden.

Das System R betrachtet nicht nur I/O Kosten, sondern auch CPU Kosten.

Der Raum von Pläne ist zu gross, und muss zuerst verringert werden. Es wird nur der Raum von links-tiefe Pläne [left-deep plans] betrachtet. Links-tief Pläne erlauben, den Output von jeder Operator in den nächsten Operator zu pipelinen. So muss die temporäre Relation im allgemeinen nicht gespeichert werden.

Cartesische Produkte werden vermieden.

8.7.1 Kosteneinschätzung

Für jeden Plan, der zur Betrachtung kommt, müssen folgende Kosten eingeschätzt:

- Kosten für die Operationen, die im Plan-Baum vorkommen. Diese hängen von den Input-Kardinalitäten.
- Kosten für die einzelnen Operationen (sequential scan, index scan, joins, etc.)
- Kosten auch für das Ergebnis von Jede Operation im Baum! Dafür wird die Information über die Input-Relationen benutzt.

8.7.2 Einschätzung der Grösse und Reduktionsfaktoren

Wir betrachten die folgende Abfrage:

```
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```

Maximum # Datensätze in dem Ergebnis ist das Produkt der Kardinalität der Relationen in der FROM Klausel.

Die Terme in der WHERE Klausel geben uns einen Verminderungsfaktor [*Reduction factor (RF)*]

Das RF assoziiert mit jedem Term entspricht der Impact, mit welchem er das Resultat verkleinert. Es gilt also die folgende Formel:

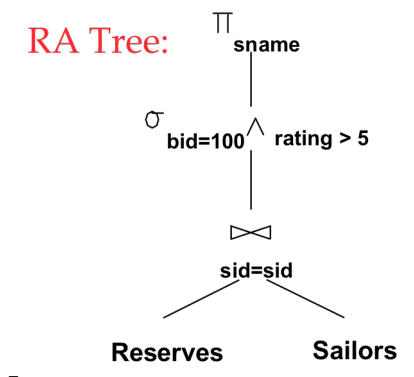
$$Result\ cardinality = Max\ \# \ tuples \ * \ product\ of\ all\ RF's.$$

8.7.3 Beispiel

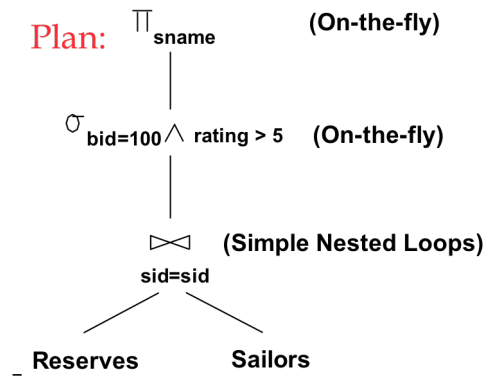
Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)

Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

Das einfachste Baum, das das relationale Algebra entspricht:



Der Plan, der dem R.A. Baum entspricht:

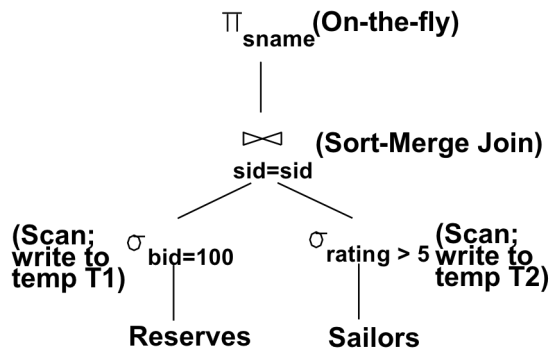


Ist aber bei weitem nicht der schlechteste Plan (man denkt z.B. an kartesische Produkt)

Es verpasst aber einige Gelegenheiten, um schneller zu werden. Vor allem hätten die Selektionen “gepusht” werden können. Indizes hätten benutzt werden können.

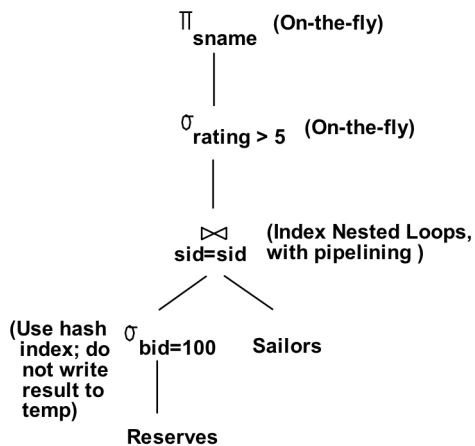
Ziel der Optimierung:: ein Plan zu finden, das das gleiche Antwort prozessiert, aber das mehr effizient ist.

8.7.4 Alternativpläne (ohne Index)



Hauptunterschied: Selects sind gestossen [pushed].

8.7.5 Alternativpläne (mit Index)



-