Partitionierung und Allokation

Literatur: Buch von P. Dadam – Kapitel 4

Ziele:

1. Speichere Daten dort, wo sie gebraucht werden.
   (Vermeide Kommunikationskosten)
2. Verteile Daten gleichmäßig auf alle Knoten
   (Lastbalancierung, Parallelität – höhere Kommunikation)

Übersicht:

- Arten der Partitionierung
  horizontal, abgeleitet, vertikal, gemischt
- Wie teilt man Relationen auf?
  (Partitionierung = logische Ebene)
- Wo speichert man die einzelnen Teile?
  (Allokation = physische Ebene)

Vorbemerkungen

- Um es richtig zu machen, muß man sehr komplexe Kostenberechnungen anstellen, die alle Details des gesamten Systems berücksichtigen.
  (Anfragen, Updates, Indexe [wechselseitige Abhängigkeit], ...)
- Partitionierung und Allokation sind zusätzliche Schritte des traditionellen (zentralen) Datenbankentwurfs

  Frage: Wiese zwei getrennte Schritte?
- Korrektheit und Partitionierung:

  – Verlustfreiheit
  – Rekonstruierbarkeit
  – Disjunktheit

Ausgangspunkt – globale Relationen (Ergebnis von ER-Modellierung):

Teile(<u>TeileNr</u>, LiefNr, Preis)
Lieferant(<u>LiefNr</u>, LiefName, Stadt)

Partitionierung

$\text{Teile}_1 = \text{funktion}_{T1}(\text{Teile})$
$\text{Teile}_2 = \text{funktion}_{T2}(\text{Teile})$
$\text{Teile}_3 = \text{funktion}_{T3}(\text{Teile})$
$\text{Lieferant}_1 = \text{function}_{L1}(\text{Lieferant})$
$\text{Lieferant}_2 = \text{function}_{L2}(\text{Lieferant})$

Allokation

$\text{Teile}_1 \rightarrow \text{Knoten3}$
$\text{Teile}_2 \rightarrow \text{Knoten2}$
$\text{Teile}_3 \rightarrow \text{Knoten2}$

$\text{Lieferant}_1 \rightarrow \text{Knoten1}$
$\text{Lieferant}_2 \rightarrow \text{Knoten2}$

Arten der Partitionierung

1. Horizontale Partitionierung

Prinzip:

Semantik:

$\text{Teile} = \text{Teile}_1 \cup \text{Teile}_2 \cup \text{Teile}_3$

Möglichkeiten den Schnitt zu ziehen:

1. Round-Robin (später, Kapitel 7)
   - schlecht für Ziel 1 (Kommunikation)
   - manchmal gut für Ziel 2 (Lastbalancierung, Parallelität)
2. Hash-Basiert (später, Kapitel 7)
   - schlecht für Ziel 1
   - häufig gut für Ziel 2 (bei gute Hashfunktion)
3. Prädikat-Basiert (heute)
   - gut für Ziel 1 (Kontrolle wo was hinkommt)
   - häufig schlecht für Ziel 2 (Partitionen unterschiedlicher Größe)

Abgeleitete Horizontale Partitionierung

- Zugehörigkeit zu einer Partition hängt vom Joinpartner ab.

  Teile_x = Teile SEMI-JOIN (σ(Lieferant, Stadt = x))

- Es gilt weiterhin (Rekonstruierbarkeit):

  $$\text{Teile} = \cup_x \text{Teile}_x$$

- Verlustfreiheit: Teile.LiefNr NOT NULL, referentielle Integrität
- Disjunktheit: Lieferant.LiefNr ist Schlüsselkandidat

Vertikale Partitionierung

Ausgangspunkt:

$R(\underline{S_1, S_2,} A_1, A_2, A_3, A_4, A_5)$

Vertikale Partitionen

$R_1(\underline{S_1, S_2,} A_1, A_2)$

$R_2(\underline{S_1, S_2,} A_3, A_4, A_5)$

Rekonstruierbarkeit: $R = R_1 \; Join \; R_2$

Verlustfreiheit: $\surd$

Disjunktheit: gilt bzgl. Nicht-Schlüsselattributen

Bemerkung:

Achte auf das Einfügen und Löschen von (globalen) Tupeln. Das muß gleich doppelt gemacht werden.

Gemischte Partitionierung

Verlustfreiheit, Rekonstruierbarkeit, Disjunktheit ergibt sich aus den einzelnen (horizontalen und vertikalen) Schnitten.

$R = R_1 \cup (R_{21} \bowtie (R_{221} \cup R_{222}))$

Wie teilt man Relationen auf?

- "Durch Hingucken"
  (so wird es üblicherweise gemacht)
- "Durch *gezieltes* Hingucken"
  (so machen wir es)

Wir erinnern uns: Ziel 1 = Speichere Daten, wo sie gebraucht werden.

Wir kümmern uns zunächst nur um horizontale Partitionierung mit Prädikaten.

Betrachte folgende 3 Anfragen:

```
SELECT *                    SELECT *
FROM Teile                  FROM Teile
WHERE TeileNr = 314         WHERE Preis > 500
```

```
SELECT *
FROM Teile
WHERE Preis < 800
```

Wie würdet Ihr partitionieren?

Allgemeines Verfahren

1. Sammele "einfache" Prädikate aus Anfragen.
   p_i = Attribute $<$ Konstante; $P = \{p_1, p_2, \ldots, p_n\}$
2. Bilde $M_n(P)$; enthält alle n-stelligen Konjunktionen von einfachen Prädikaten aus P, die entweder in natürlicher Form (p_i^+) oder als Negation (p_i^-) auftreten.

$$M_n(P) = \{m \mid m = \bigwedge_{p_i \in P} p_i^{\pm}, 1 \leq i \leq n\}$$

   $\Rightarrow$ Sehr viele Partitionen (2^n), deshalb
3. Eliminiere nicht relevante Prädikate
4. Schmeiße (leere) Partitionen weg – 2 Regeln

Regel 1: Elimination unerfüllbarer MIN-Terme

p_1 := “TeileNr = 4711”

p_2 := “TeileNr = 31415”

Regel 2: Elimination abhängiger Prädikate

Lieferant “Strunz” ist der einzige Lieferant, der Teile im Wert von > 800 liefert.

p_1 := “Preis > 1000”

p_2 := “LiefNr = Strunz”

(Beachte $p_1 \rightarrow p_2$)

Relevanz eine Prädikates p

Daumenregel: p ist irrelevant, falls

$$\frac{\text{access}(R_{p^+})}{\text{card}(R_{p^+})} \approx \frac{\text{access}(R_{p^-})}{\text{card}(R_{p^-})}$$

D.h.: Partitioniere, wenn auf einem kleinen Teil der Daten sehr häufig zugegriffen wird.

Beispiel: Strunz liefert 50 Teile der insgesamt 2050 Teile. 300 von insgesamt 500 Anfragen (in der Sekunde) sind von der Art:
SELECT *
FROM Teile
WHERE LiefNr = “Strunz” $\wedge$ ...

$$\begin{array}{ll} \text{access}(R_{p^+}) = 500 & \text{access}(R_{p^-}) = 200 \\ \text{card}(R_{p^+}) = 50 & \text{card}(R_{p^-}) = 2000 \end{array}$$

$$\frac{500}{50} \gg \frac{200}{2000}$$

Also, partitioniere nach “Strunz.”

Abgeleitete horizontale Partitionierung

Man schaut sich Prädikate von Join Queries an.

Beispiel:

SELECT *
FROM Teile t, Lieferant l
WHERE t.LiefNr = l.LiefNr $\wedge$ l.Stadt = "Paris"

- Wenn Teile überwiegend in (Join) Anfragen zusammen mit "Lieferant" auftritt, dann ist das ein Indiz, daß eine abgeleitete horizontale Partitionierung sinnvoll sein könnte.
- Verwende selbe Technik wie für reguläre horizontale Partitionierung – bei abgeleiteter Partitionierung von Teile betrachte allerdings das Prädikat "l.Stadt = Paris" auf *Lieferant*.
- Wiederum Zugriffshäufigkeit von Anfragen für Relevanz wichtig.
- Man kann natürlich abgeleitete Partitionierung auch zusätzlich zur regulären horizontalen Partitionierung einsetzen.
  (Und dabei unser Verfahren anwenden.)
- Zusätzlicher Aspekt: Partitioniere auch Lieferant; dann besonders gut, da man den Join sehr effizient (lokal) ausführen kann.

Vertikale Partitionierung

- Betrachte Auftreten von *Attributen* (anstatt von Prädikaten) in Anfragen.
- Außerdem betrachte ebenfalls die Häufigkeit von Einfüge- und Löschoperationen.
- Lösungsansatz: Paper von Muthuraj et al.: A Formal Approach to the Vertical Partitioning in Distributed Database Design. Proc. 2nd Conf. on PDIS, Januar 1993, San Diego, CA.
- Aber zu schwer für uns.
  (Somit ist auch das gemischte Partitionierungsproblem zu schwer für uns.)