Prof. Dr. V. Linnemann
Stefan Werner
Universität zu Lübeck
Institut für Informationssysteme

Lübeck, den 28. November 2011

Anfrageverarbeitung

Wintersemester 2011/2012

2. Block

Das Ziel des zweiten Blocks ist die heuristische, algebraische Optimierung der **Simple SQL**-Anfragen. Im ersten Block sind **Simple SQL**-Anfragen in einen kanonischen Ausdruck der relationalen Algebra übersetzt und auf den Tabellen der Datenbank ausgeführt worden. Ausgangspunkt der Optimierung ist der kanonische Ausdruck der relationalen Algebra, der mittels Optimierung in einen anderen optimierten Ausdruck der relationalen Algebra überführt wird. Der neue Ausdruck kann dann wie gehabt ausgeführt werden. Die einzelnen Schritte der heuristischen Optimierung umfassen:

- Verschieben der `select`-Operationen nach unten im Anfragebaum.
- Kreuzprodukt und `select`-Operationen werden durch `join`-Operationen ersetzt.
- `project`-Operationen werden im Anfragebaum nach unten verschoben.

Die Kosten einiger Beispielanfragen sollten nun deutlich geringer ausfallen. Die folgenden Aufgaben beschreiben das Vorgehen zur Realisierung dieser Anfrageoptimierung.

Aufgabe 1: Kaskadierung von `select`-Operationen

Um `select`-Anweisungen möglichst weit nach unten in den Anfragebaum verschieben zu können, müssen sie zunächst kaskadiert werden. Unter Verwendung der Regel:

Kaskade von σ: Eine konjunktive Auswahlbedingung kann in eine Kaskade (d.h. Sequenz) einzelner σ-Operationen aufgeteilt werden:
$\sigma_{c_1 and c_2 and ... c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(...\sigma_{c_n}(R)...))$

werden eventuell vorhandene `select`-Operationen mit konjunktiven Bedingungen in eine Kaskade von `select`-Operationen aufgeteilt. Dies erlaubt einen grösseren Freiheitsgrad für deren Verschiebung.

Die Kaskadierung und auch alle weiteren Schritte der Optimierung können beispielsweise wie folgt implementiert werden. Schreiben Sie jeweils eine Klasse, hier z.B. `optimierung.CascadeSelects`, die die Schnittstelle `optimierung.IOptimierung` implementiert. Die Schnittstelle `optimierung.IOptimierung` beinhaltet die Methode `void optimize(ITreeNode plan)`. Die Variable `plan` stellt dabei den Ausführungsplan der relationalen Algebra dar, der optimiert werden soll.

Aufgabe 2: `select`-Operationen verschieben

Unter Verwendung der folgenden Regeln:

Kommutativität von σ: Die σ-Operation ist kommutativ:
$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$

Vertauschen von σ mit π: Beinhaltet die Auswahlbedingung c nur die Attribute $A1, ..., An$ in der Projektionsliste, können die beiden Operationen vertauscht werden:
$\pi_{A1,A2,...,An}(\sigma_c(R)) \equiv \sigma_c(\pi_{A1,A2,...,An}(R))$

Vertauschung von σ mit $\bowtie$ (oder $\times$): Wenn alle Attribute in der Auswahlbedingung c nur die Attribute einer der zu vereinigenden Relationen, z.B. R, beinhalten, können die beiden Operationen wie folgt vertauscht werden:
$\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$

in Bezug auf die Kommutativität von `select` mit anderen Operationen wird jede `select`-Operation so weit im Anfragebaum nach unten verschoben, wie es die in der `select`-Bedingung enthaltenen Attribute zulassen.

Implementieren Sie diesen Optimierungsschritt beispielsweise in der Klasse `optimierung.MoveSelection`. Um die jeweils beteiligten Attribute zu ermitteln, bietet es sich an, sowohl im Interface `relationenalgebra.IBooleanExpression`

als auch im Interface `relationenalgebra.ITreeNode` eine Methode `List<String> getAttributes()` zu definieren und entsprechend zu implementieren. Die Methode ermittelt rekursiv die beteiligten Attributnamen.

Aufgabe 3: Zusammenfassen mittels `join`-Operationen

Unter Verwendung der Regel:

Konvertierung einer (σ,$\times$)-Sequenz in $\bowtie$: Wenn die Bedingung c einer σ-Operation, die einer $\times$-Operation folgt, einer Join-Bedingung entspricht, wird die (σ,$\times$)-Sequenz wie folgt in einen $\bowtie$ konvertiert:
$(\sigma_c(R \times S)) \equiv (R \bowtie_c S)$

d.h. es wird eine Kreuzprodukt-Operation mit einer nachfolgenden `select`-Operation im Baum zu einer Join-Operation kombiniert, wenn die Bedingung eine Join-Bedingung darstellt.

Implementieren Sie diesen Optimierungsschritt z.B. mittels der Klasse `optimierung.DetectJoins`.

Aufgabe 4: `project`-Operationen verschieben

Unter Verwendung der Regeln:

Kaskade von π: In einer Kaskade (Sequenz) von π-Operationen können alle ausser der letzten ignoriert werden:
$\pi_{List1}(\pi_{List2}(...(\pi_{Listn}(R))...)) \equiv \pi_{List1}(R)$

Vertauschen von σ mit π.

Vertauschen von π mit $\bowtie$ (oder $\times$): Gegeben sei die Projektionsliste
$L = \{A_1, ..., A_n, B_1, ..., B_m\}$ mit $A_1, ..., A_n$ als Attribute aus R und $B_1, ..., B_m$ Attribute aus S. Beinhaltet die Join-Bedingung c nur Attribute aus L, können die beiden Operationen wie folgt vertauscht werden:
$\pi_L(R \bowtie_c S) \equiv (\pi_{A_1,...,A_n}(R)) \bowtie_c (\pi_{B_1,...,B_m}(S))$
Enthält die Join-Bedingung c zusätzliche, in L nicht vorkommende Attribute, müssen diese zur Projektionsliste hinzugefügt werden und es bedarf auch einer letzten π-Operation. Sind beispielsweise die Attribute $A_{n+1}, ..., A_{n+k}$ von R und $B_{m+1}, ..., B_{m+p}$ von S von der Join-Bedingung c betroffen, jedoch nicht in der Projektionsliste L, können die Operationen wie folgt vertauscht werden:
$\pi_L(R \bowtie_c S) \equiv \pi_L((\pi_{A1,...,An,An+1,...,An+k}(R)) \bowtie_c (\pi_{B1,...,Bm,Bm+1,...,Bm+p}(S)))$

in Bezug auf die Kaskadierung und Kommutativität von Project- mit anderen Operationen werden Listen von Projektionsattributen aufgeteilt und im Baum so weit wie

möglich nach unten verschoben, indem je nach Bedarf neue Project-Operationen erstellt werden. Nur die in dem Anfrageresultat und in späteren Operationen im Anfragebaum benötigten Attribute sollen nach jeder Project-Operation beibehalten werden.

Implementieren Sie diesen Optimierungsschritt z.B. mittels der Klasse `optimierung.MoveProjection`.

Aufgabe 5: Beispielanfragen

Testen Sie die folgenden Beispielanfragen auf der gegebenen Testdatenbank und ermitteln Sie die Kosten, die entstehen:

a) bei keiner Optimierung

b) bei Verwendung der Optimierungsschritte inklusive Teilaufgabe 2.

c) ... inklusive Teilaufgabe 3.

d) ... inklusive Teilaufgabe 4.

```
select B.Titel
from Buch as B,
     Kunde as K,
     Buch_Bestellung as BB,
     Kunde_Bestellung as KB
where K.Name="KName1" and
      K.ID=KB.K_ID and
      KB.B_ID=BB.Be_ID and
      BB.Bu_ID=B.ID
```

```
select B.ID, K.Name
from Bestellung as B, Kunde as K, Kunde_Bestellung as KB
where KB.K_ID=K.ID and KB.B_ID=B.ID and B.ID="Bestellung5"
```

```
select Name
from Kunde,Kunde_Bestellung
where ID=K_ID and Name="KName1"
```

Abgabetermin: Montag, den 9. Januar 2012