



## Übung zur Vorlesung 'Informationssysteme: Verteilte und Web-Datenbanken'

Richard Kuntschke (richard.kuntschke@in.tum.de)

### Blatt 2

**Abgabe: Dienstag, 2. November 2004 bis 12.00 Uhr** im Raum MI 02.11.018 im entsprechend beschrifteten Fach.

### Aufgabe 1 - Joins in verteilten DBMSen (Bind-Join) (7 + 3 Punkte)

Nehmen Sie folgende Situation an: Ein DBMS auf einem Rechner  $A$  will den Join von zwei Relationen  $R$  und  $S$  berechnen. Die Relation  $S$  befindet sich allerdings nicht auf Rechner  $A$ , sondern auf einem entfernten Rechner  $B$ . Leider erlaubt das DBMS auf Rechner  $B$  nicht das Auslesen der kompletten Relation  $S$ , d. h. es ist nicht möglich die Relation  $S$  einfach zu Rechner  $A$  zu übertragen und dort einen normalen Join der beiden Relationen auszuführen. Stattdessen kann das DBMS auf Rechner  $B$  aber zu einem gegebenen Tupel  $t$  eine Menge möglicher Joinpartner aus  $S$  zurückliefern. Aus dieser Menge und dem Tupel  $t$  kann das DBMS auf Rechner  $A$  dann einen Teil des Join-Ergebnisses berechnen. Nachdem das DBMS auf Rechner  $A$  so alle Tupel  $t \in R$  verarbeitet hat, ist der Join von  $R$  und  $S$  vollständig berechnet.

- a) Berechnen Sie, welche Bedeutung die Latenzzeit in diesem Fall für die Auswertung des Bind-Joins von  $R$  und  $S$  hat, verglichen mit einer kompletten Übertragung der Relation  $S$  zum Rechner  $A$ . Vernachlässigen Sie dabei die Berechnungskosten der DBMSen. Es seien noch folgende Werte gegeben:

Größe von $R$	$r$ Tupel der Größe $k_R$
Größe von $S$	$s$ Tupel der Größe $k_S$
Bandbreite zwischen $A$ und $B$	$b$
Latenzzeit zwischen $A$ und $B$	$l$
Kardinalität der Menge der möglichen Joinpartner aus $S$ für ein Tupel $t \in R$	$j$

- b) Wo tritt diese Situation in der Realität auf?

### Aufgabe 2 - Operatoren für die verteilte Anfragebearbeitung (5 + 5 Punkte)

- a) Implementieren Sie die beiden Operatoren `Tablescan` und `NLJoin` (Nested-Loops-Join) als Java-Klassen. Die beiden Operatoren sollen sich an das Iterator-Konzept halten. Leiten Sie die Operatoren dazu von einem Java-Interface `DBIterator` mit den folgenden Methoden ab:

- `String[] open() throws Exception`  
Öffnet die Eingaberelationen. Das zurückgegebene String-Array soll die Attributnamen der Ergebnisrelation enthalten. Man soll einen Operator auch mehrere Male öffnen können, d. h. nach einem `open`-Aufruf soll beim nächsten `next`-Aufruf wieder das erste Tupel zurückgeliefert werden.
- `Object[] next() throws Exception`  
Liefert das nächste Ergebnistupel der Ergebnisrelation als Objekt-Array zurück oder `null`, falls alle Tupel geliefert wurden. Aus diesem Array können auch die Typen der einzelnen Attribute bestimmt werden!

- `void close() throws Exception`  
Schließt die Eingaberelationen.

Der Operator `Tablescan` erhält in seinem Konstruktor den Namen einer Datei, die er einlesen soll. Die Signatur des Konstruktors sieht also folgendermassen aus:

```
Tablescan(String filename) throws Exception
```

Die einzulesende Datei soll folgendes Format besitzen:

```
Attributname_1 <TAB> Attributname_2 <TAB> Attributname_3 ... <RETURN>
Attributtyp_1 <TAB> Attributtyp_2 <TAB> Attributtyp_3 ... <RETURN>
Wert <TAB> Wert <TAB> Wert ... <RETURN>
...
```

An Attributtypen sollen die Java-Typen `String`, `Integer`, `Double` und `Float` unterstützt werden. In der Eingabedatei sollen Strings in Anführungszeichen eingeschlossen werden. Nehmen Sie außerdem an, dass Strings keine Anführungszeichen, Tabulatoren und Zeilenumbrüche (sowie sonstige Sonderzeichen) enthalten.

Der Operator `MLJoin` soll einen natürlichen Verbund mit Hilfe eines Nested-Loops-Algorithmus berechnen. Die Klasse soll folgenden Konstruktor besitzen:

```
MLJoin(DBIterator left, DBIterator right) throws Exception
```

Die beiden Objekte vom Typ `DBIterator` bilden die linke und rechte Eingaberelation des Joinoperators.

Testen Sie ihre Operatoren anhand verschiedener Eingabedateien. Welche zusätzlichen Parameter könnte man dem Joinoperator übergeben?

- b) Realisieren Sie die verteilte Auswertung eines einfachen Anfrageplans bestehend aus zwei `Tablescan`-Operatoren und einem `MLJoin`-Operator auf zwei Rechnern. Das Programm auf dem ersten Rechner soll einen `Tablescan`-Operator, den `MLJoin`-Operator und einen speziellen Kommunikationsoperator (`ClientProxy`) erzeugen. Der `ClientProxy`-Operator soll für den Verbindungsaufbau zum zweiten Rechner mit dem anderen Teilplan bzw. Programm sorgen. Das zweite Programm auf dem anderen Rechner soll den zweiten `Tablescan`-Operator und ebenfalls einen speziellen Kommunikationsoperator (`ServerProxy`) erzeugen. Der `ServerProxy`-Operator startet einen Server, der die Verbindung vom ersten Rechner entgegennimmt, und übernimmt die Kommunikation mit dem lokalen `Tablescan`-Operator. Achten Sie auf die korrekte Verbindung der Operatoren gemäß dem Iterator-Konzept! Diskutieren Sie die Vor- und Nachteile Ihrer Lösung.

**Hinweis:** Schicken Sie Ihren Sourcecode in einer ZIP-Datei als Anhang an `richard.kuntschke@in.tum.de`. Verwenden Sie als Subject/Betreff bitte 'VWDB: Blatt 2'. Schreiben Sie in den Nachrichtentext der E-Mail die Namen aller Gruppenmitglieder und die Gruppennummer. Legen Sie Ihrer Abgabe außerdem einen Ausdruck des Sourcecodes bei, ebenso ein Protokoll Ihrer Programmtests. Dokumentieren Sie Ihre Programme sorgfältig.

#### Allgemeine Hinweise

- Die Aufgaben der Übungsblätter sind gemeinsam von den Mitgliedern eines Teams zu lösen. Die Teameinteilung (je 3 oder 4 Teilnehmer) ist während des Semesters fest!
- Auf jedes Blatt sind die Namen der Team-Mitglieder und die Team-Nummer zu schreiben. Jedes Team-Mitglied muss die von seinem Team bearbeiteten Aufgaben 'vorrechnen' können (d. h. der Lösungsansatz seines Teams und evtl. dabei auftretende Probleme müssen bekannt und verstanden sein).