

# Anfrageverarbeitung

Wintersemester 2013/2014

## 1. Block

Das Ziel dieses ersten Blocks ist es, ein vereinfachtes relationales Einbenutzer-Datenbanksystem ohne Optimierung und Transaktionsverwaltung zu erstellen.

- Der Benutzer kann an die Datenbank Anfragen in **SimpleSQL** stellen. **SimpleSQL** beinhaltet sowohl lesende als auch schreibende Anfragen.
- Die SQL-Anfragen werden in eine Anfrage der relationalen Algebra übersetzt
- und anschließend auf relationalen, persistenten Tabellen ausgeführt.
- Für den weiteren Verlauf steht ein Kostenmaß für die Ausführungszeit von lesenden **SimpleSQL**-Anfragen zur Verfügung sowie Beispieldaten eines Buchverstands.

Das Ausgangsprojekt für Eclipse ist einem Git-Repository hinterlegt. Die Zugangsdaten erhalten Sie in der ersten Übung. Schreiben Sie bitte Ihrem Übungsgruppenleiter eine E-Mail, falls Sie keine Zugangsdaten erhalten haben sollten.

## Aufgabe 1: Anfragen der Relationalen Algebra

In diesem Teil sollen Java-Klassen geschrieben werden, die Anfragen der relationalen Algebra mittels einer Baumstruktur modellieren. Alle Anfragen in **SimpleSQL** werden zunächst in eine kanonische Form der relationalen Algebra überführt. Später ist diese kanonische Form der Ausgangspunkt für Optimierungsprozesse. Ganz allgemein lässt sich eine **SimpleSQL**-Anfrage in die folgende kanonische Form überführen.

- SimpleSQL:

```
SELECT <spaltennamen>  
FROM <tabellenname_1>, ..., <tabellenname_n>  
WHERE <bedingung>
```

- Relationale Algebra

```
 $\pi_{\langle \text{spaltennamen} \rangle} ($   
 $\sigma_{\langle \text{bedingung} \rangle} ($   
 $\langle \text{tabellenname}_1 \rangle \times \dots \times \langle \text{tabellenname}_n \rangle$   
 $)$   
 $)$ 
```

Bedingungen sind boolesche Ausdrücke in einer kanonischen Form. Implementieren Sie alle benötigten Klassen als Bestandteil des Packages **relationalgebra**. Das Klassendiagramm in Abbildung 1 ist ein Vorschlag zur Modellierung relationaler Anfragen und darin benötigter boolescher Ausdrücke, die ausgehend von **Simple SQL** entstehen können.

## Aufgabe 2: Schreibende Anfragen

Neben rein lesenden Anfragen bietet **SimpleSQL** auch schreibende Anfragen, wie zum Beispiel `create table` und `insert`. Der Einheitlichkeit halber sollen auch diese Anfragen in Ausdrücke einer erweiterten relationalen Algebra überführt werden. Implementieren Sie die hierfür benötigten Java-Klassen ebenfalls im Package **relationalgebra**. Das Klassendiagramm in Abbildung 2 zeigt eine mögliche Lösung.

## Aufgabe 3: SimpleSQL → Relationale Algebra

Nun stehen alle Java-Klassen zur Verfügung, um einen Syntaxbaum in **SimpleSQL** sowie Ausdrücke der relationalen Algebra zu repräsentieren. In dieser Aufgabe soll nun die Überführung eines **SimpleSQL**-Syntaxbaums in einen Ausdruck der relationalen Algebra implementiert werden. Der resultierende Ausdruck der relationalen Algebra soll sich in der bereits eingeführten kanonischen Form befinden. Verwenden Sie für die Übersetzung einer **SimpleSQL**-Anfrage in die entsprechende relationale Anfrage das Visitor-Muster indem Sie eine Klasse **SimpleSQLToRelAlgVisitor**, die von `parser.visitor.ObjectDepthFirst` erbt, implementieren.

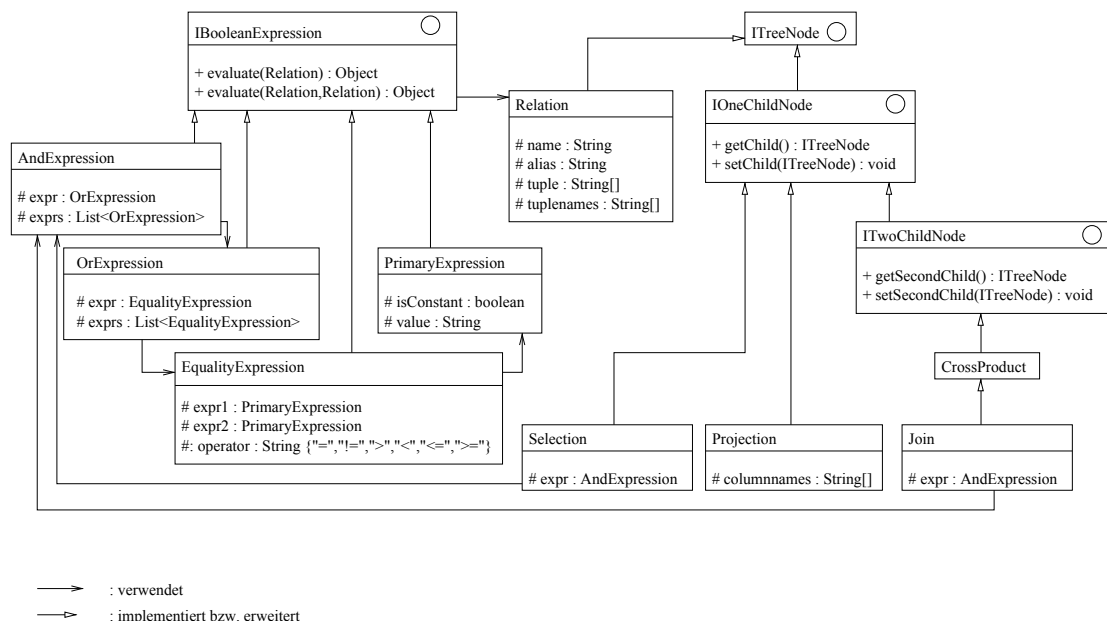


Abbildung 1: Klassen zur Modellierung relationaler Anfragen

#### Aufgabe 4: Datenbanktabellen

In dieser Aufgabe werden die persistenten Tabellen der relationalen Datenbank modelliert. In dieser stark vereinfachten relationalen Datenbankversion soll das folgendermaßen geschehen. Die Datenbank ist ein Dateiverzeichnis. Tabellen der Datenbank werden auf einzelne Dateien innerhalb des Datenbankverzeichnisses abgebildet. Die Tabellen-datei wird mittels Java eigener Objektserialisierung erzeugt. Implementieren Sie hierfür die Klasse **database.Table**. Ein Vorschlag für die Struktur der Klasse ist in Abbildung 3 dargestellt. Die `toString` Methode sollte so überschrieben werden, dass eine Datenbank-tabelle samt Inhalt geeignet ausgegeben wird.

#### Aufgabe 5: Einfache Ausführung

In dieser Aufgabe wird die eigentliche Ausführung eines Ausdrucks der relationalen Algebra implementiert. Der Ausdruck der relationalen Algebra muß in Operationen, die auf den Tabellen der Datenbank arbeiten, übersetzt werden. Implementieren Sie hierfür gegebenenfalls die Methoden der Klasse **main.Main**.

#### Aufgabe 6: Kostenmaß

In dieser Aufgabe soll ein Kostenmaß für die Ausführungszeit lesender Anfragen implementiert werden. Neben der reinen Ausführung der Anfrage sollen nun zusätzlich auch noch deren Kosten berechnet und ausgegeben werden. Wenn die Funktion

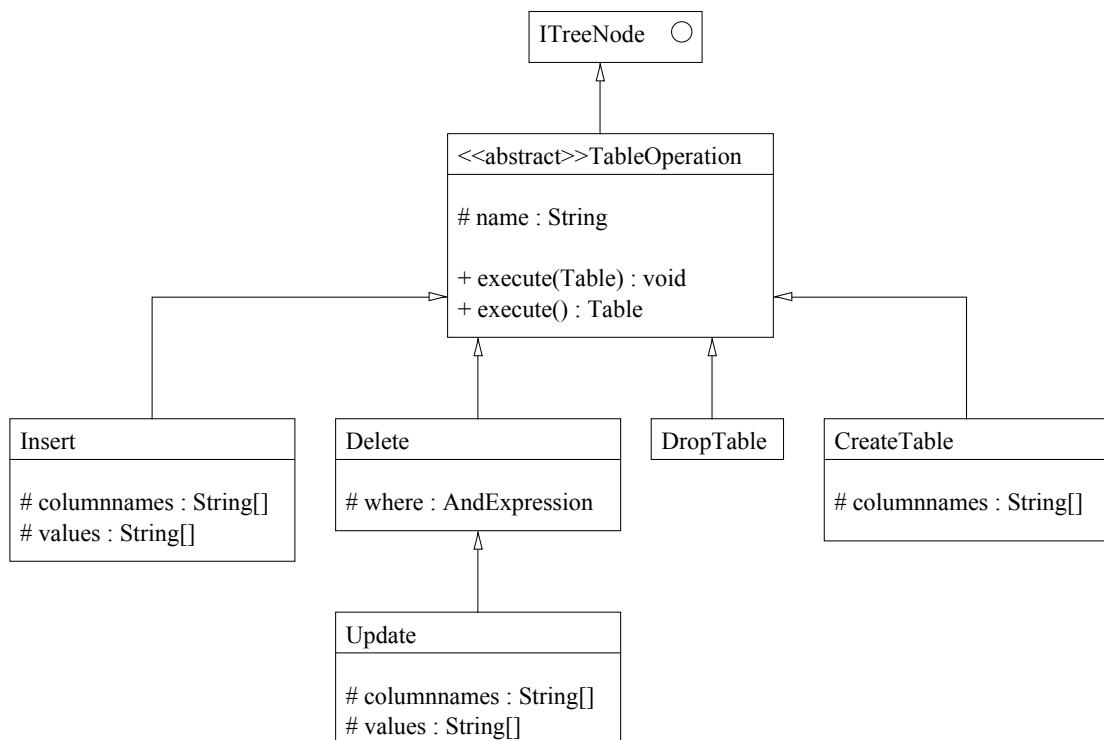


Abbildung 2: Klassen zur Modellierung schreibender Anfragen

$Zeilen(T)$  die Zeilenanzahl einer Tabelle und die Funktion  $Spalten(T)$  deren Spaltenanzahl liefert, dann können die Kosten für lesende Anfragen wie folgt ermittelt werden. Im Falle einer Selektion bzw. eines Joins werden mittels  $Zeilen_s(T)$  allerdings nur diejenigen Zeilen zurück gegeben, die tatsächlich überprüft worden sind.

Operation	Kosten
$\sigma_b(T)$	$Zeilen_s(T) * Spalten(T)$
$\pi_{c_1, \dots, c_n}(T)$	$Zeilen(T) * n$
$T_1 \times T_2$	$Zeilen(T_1) * Zeilen(T_2) * (Spalten(T_1) + Spalten(T_2))$
$T_1 \bowtie_b T_2$	$Zeilen_s(T_1) * Zeilen_s(T_2) * (Spalten(T_1) + Spalten(T_2))$

Erweitern Sie hierfür die Tabellenklasse und weitere betroffene Klassen geeignet, beziehungsweise implementieren neue.

### Aufgabe 7: Beispieldaten

Das Projekt beinhaltet eine Textdatei namens `kundendb.txt`, die **SimpleSQL**-Befehle zur Erzeugung einer Testdatenbank enthält. Erweitern Sie die Klasse `main.Main`, so dass eine Datei mit **SimpleSQL**-Befehlen eingelesen und abgearbeitet werden kann. Jeder einzelne Befehl wird mit einem Semikolon abgeschlossen. Führen Sie anschließend

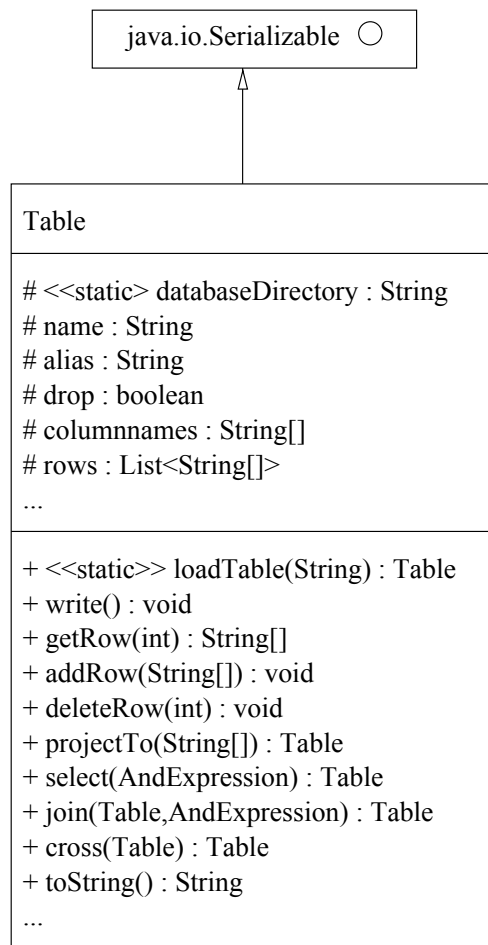


Abbildung 3: Klasse zur Modellierung einer Datenbanktabelle

die Beispiel-Anfragen der Datei `sql.txt` aus und geben Sie die berechneten Kosten an.

---

**Abgabetermin:** Montag, den 2. Dezember 2013 in der Übung