Anfrageverarbeitung
- Übung -

Wintersemester 2011/2012

Stefan Werner

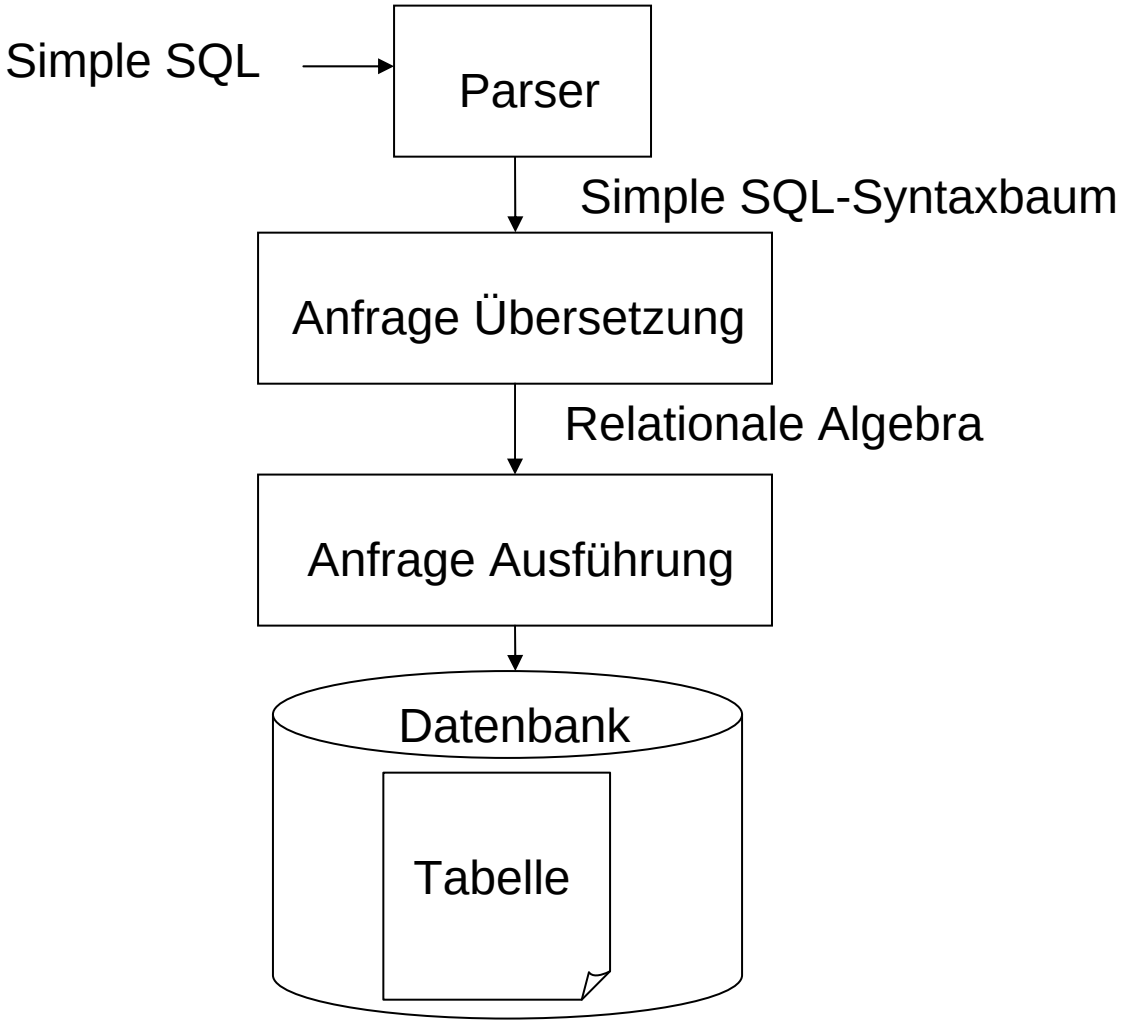
Übersicht

- Implementierung eines stark vereinfachten Datenbanksystems:
  - Einbenutzersystem
  - relational
  - vereinfachte SQL-Anfragesprache (*SimpleSQL*)

Übersicht (2)

- 1. Block (27. Oktober – 28. November)
  - Übersetzung von Simple SQL-Anfragen in relationale Algebra und Ausführung auf Tabellen
- 2. Block (28. November – 9. Januar)
  - Optimierung von Simple SQL-Anfragen
- 3. Block (9. Januar – 6. Februar)
  - Transaktionsunterstützung
- je Block ca. 4-5 Wochen

1. Block - Übersicht



Simple SQL - Grammatik

- Simple SQL ::= Query
  | Update
  | Delete
  | Insert
  | CreateTable
  | DropTable
- Siehe Grammatik-Datei SimpleSQL.jj
- Keine verschachtelten Anfragen
- Nur Zeichenketten-Datentyp (varchar)

Simple SQL - Beispiel

- select B.Titel
  from Buch as B, Buch_Autor as BA
  where BA.Autor=“Frank Schätzing“
  and BA.B_ID=B.ID
- insert into Buch (ID,Titel)
  values (“BID1“,“Der Schwarm“)
  …

Der Simple SQL Parser

- Grammatik-Datei SimpleSQL.jj
- Java Tree Builder (JTB) und Java Compiler Compiler (JCC) -> Parser und Syntaxbaum…
- Generierte Klassen:
  - parser.*,
  - parser.gene.*,
  - parser.syntaxtree.* und
  - parser.visitor.*

Ziel 1: Simple SQL -> Relat. Algebra - Kanonische Form

- In Simple SQL:

```
SELECT <Spaltennamen>
FROM <Tabellenname 1>,...,<Tabellenname n>
WHERE <Bedingungen>
```

- In Relationaler Algebra:

$$\pi_{<Spaltennamen>}(\sigma_{<Bedingungen>}(<Tabellenname\ \ 1> \times ... \times <Tabellenname\ \ n>))$$

Ziel 1: Simple SQL -> Relat. Algebra
- Simple SQL parsen

- String simpleSQL = “…“;
- SimpleSQLParser parser =
  new SimpleSQLParser(
  new StringReader(simpleSQL));
- parser.setDebugAll(false);
- try{ CompilationUnit cu =
  parser.CompilationUnit();
  }catch(ParseException e){…}

Ziel 1: Simple SQL -> Relat. Algebra - Simple SQL parsen (2)

- CompilationUnit:
  - ist der geparste Syntaxbaum der Simple SQL-Anfrage
- Klassen des Syntaxbaumes:
  - Package parser.syntaxtree

Ziel 1: Simple SQL -> Relat. Algebra - Syntaxbaum besuchen…

- **Das Visitor-Entwurfsmuster:**

  Trennung Algorithmus und Objektstruktur → Hinzufügen neuer Operationen auf dieser Struktur, ohne diese Struktur verändern zu müssen
- Allgemeinen Visitor für einen Simple SQL-Syntaxbaum : →parser.visitor.ObjectDepthFirst
- Eigenen, speziellen Visitor durch Erweiterung schreiben

Ziel 1: Simple SQL -> Relat. Algebra - Syntaxbaum besuchen…(2)

- Verwendung und Aufruf des Visitors:

```
CompilationUnit cu = …;
ObjectDepthFirst visitor =
                              new
ObjectDepthFirst();
cu.accept(visitor,null);
```

Ziel 2: Relationale Algebra - Modellierung

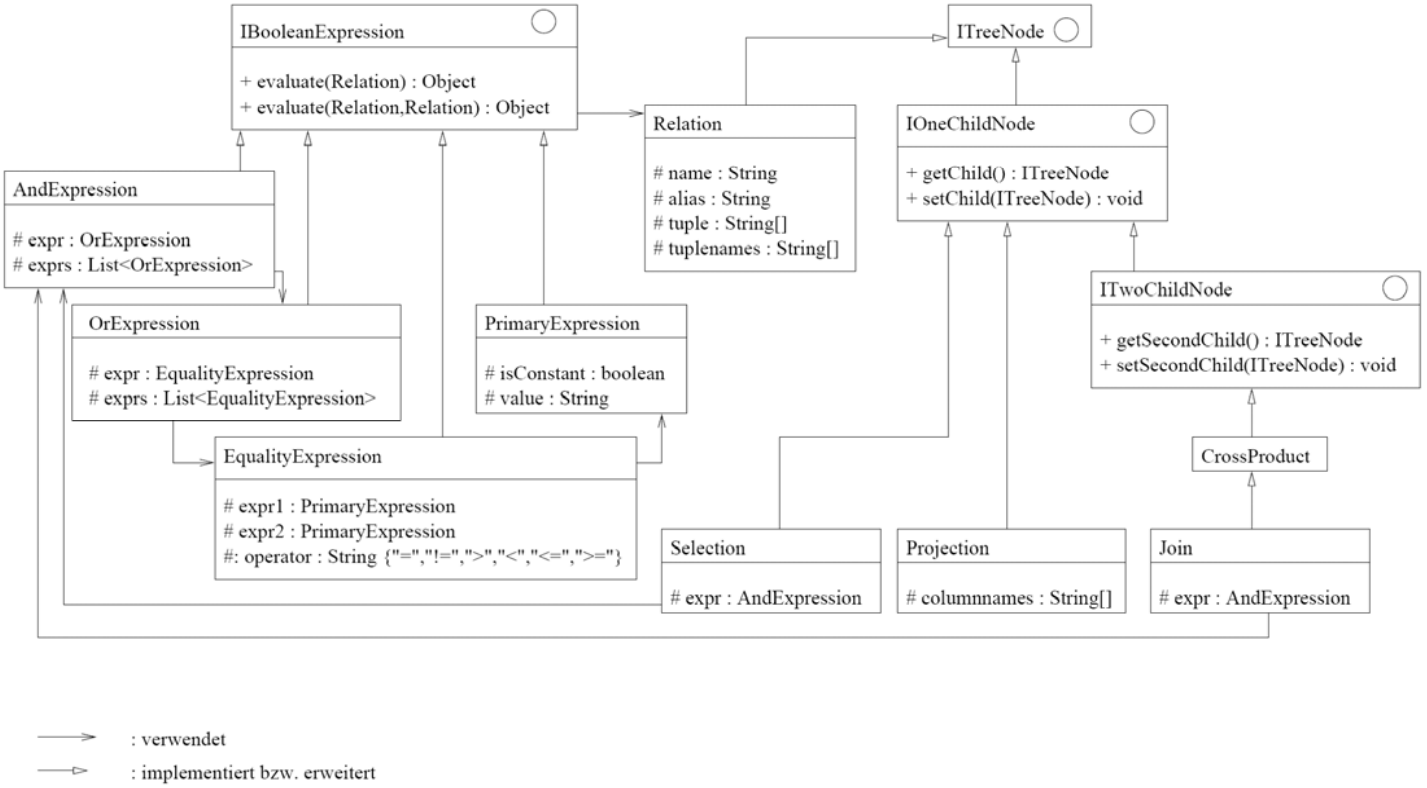


Abbildung 1: Klassen zur Modellierung relationaler Anfragen

Ziel 2: Relationale Algebra - Modellierung (2)

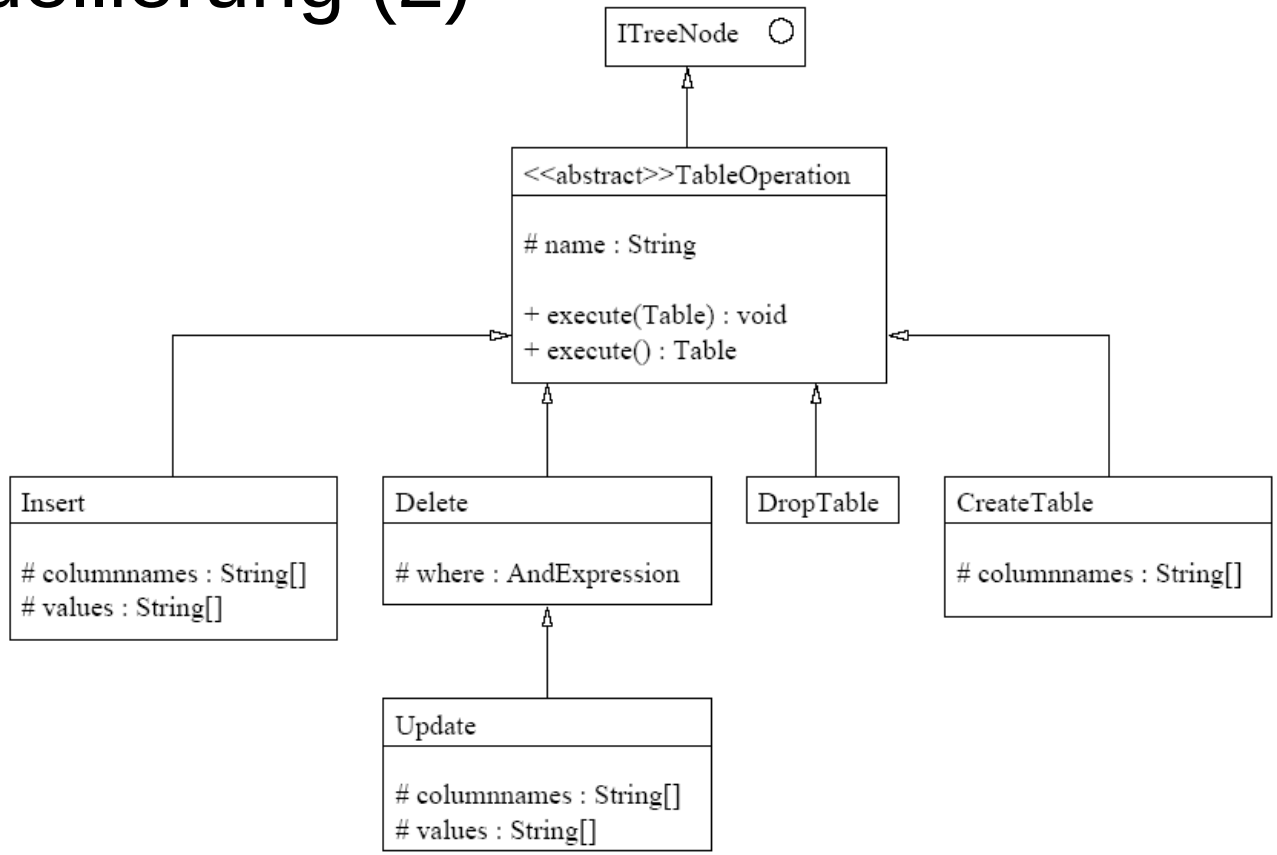


Abbildung 2: Klassen zur Modellierung schreibender Anfragen

Ziel 3: Datenbanktabellen

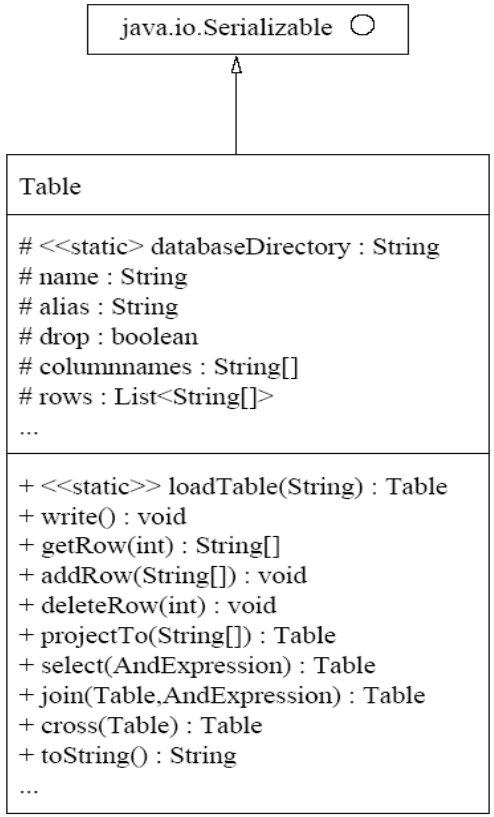


Abbildung 3: Klasse zur Modellierung einer Datenbanktabelle

Ziel 3: Datenbanktabellen (2)

- Schreiben:

```
FileOutputStream fos = new FileOutputStream(databaseDirectory+"\\"+name);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(this);
oos.close();
```

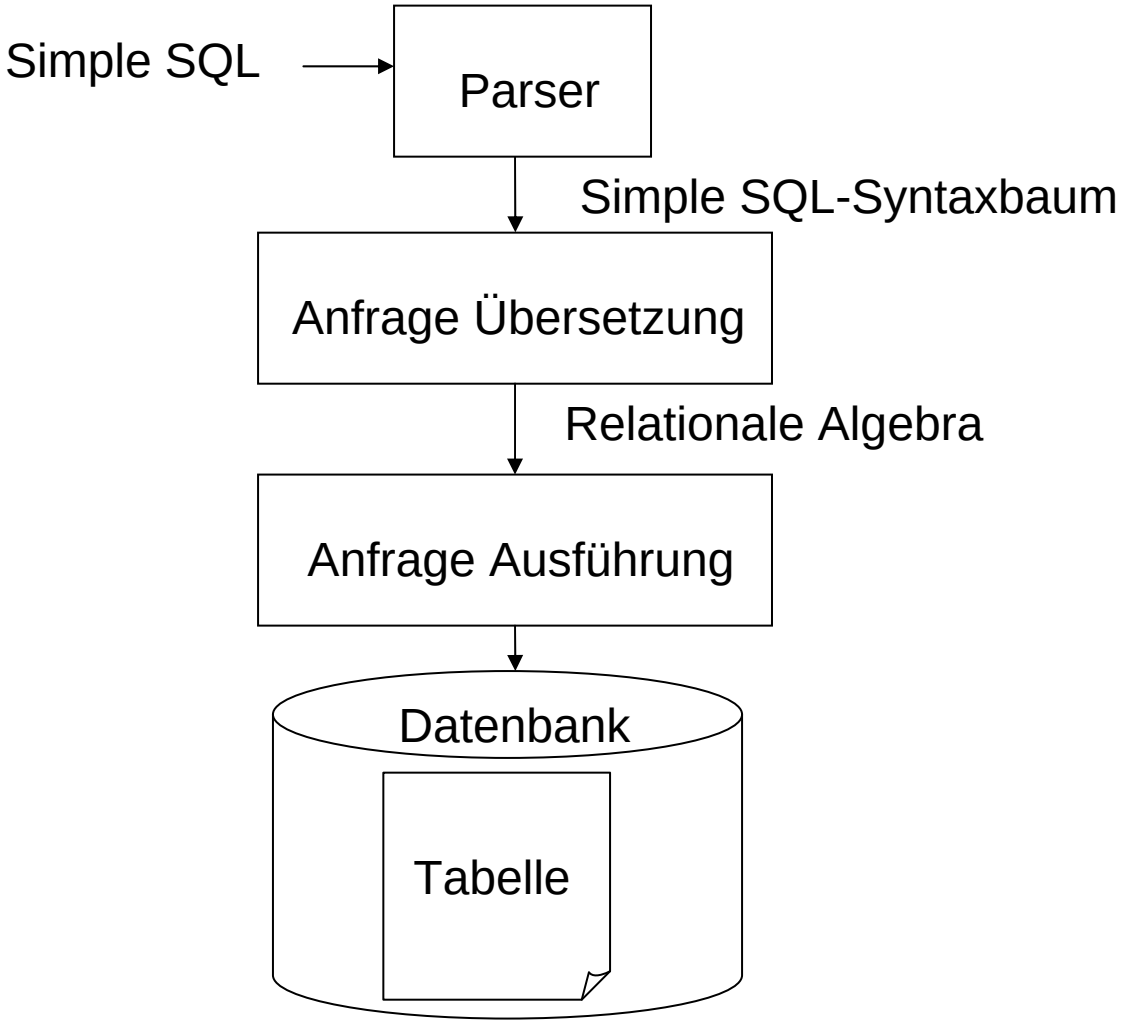
- Lesen:

```
FileInputStream fis = new
FileInputStream(databaseDirectory+"\\"+name);
ObjectInputStream ois = new ObjectInputStream(fis);
Table table = (Table) ois.readObject();
ois.close();
```

Ziel 4: Einfache Ausführung

- Bisher: Simple SQL geparst
- Eigentlich: Simple SQL validieren
- Simple SQL in Relationalen Algebra-Ausdruck überführt
- Relationalen Algebra-Ausdruck als Operationen auf den Datenbanktabellen ausführen
- → main.Main

1. Block - Übersicht



Ziel 5: Kostenmaß

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

- Implementierung z.B. indem Tabellenmethoden neben der Ergebnistabelle auch die Kosten zurückliefern → eigene Ergebnisklasse
- → Was bewirken spätere Optimierungsprozesse?

Ziel 6: Beispieldaten - Datenbankschema

- → kundendb.txt und sql.txt
- Buch(ID,Titel),
  Kunde(ID,Name,Vorname,Adresse),
  Bestellung(ID,Datum,Preis,IstBezahlt)
- Buch_Autor(B_ID,Autorenname),
  Kunde_Bestellung(K_ID,B_ID),
  Buch_Bestellung(Bu_ID,Be_ID)