Übung zur Vorlesung "Anfrageverarbeitung"

Stefan Werner

Institut für Informationssysteme, Universität zu Lübeck

Wintersemester 2013/2014

Inhalt

Inhalt

Ziele

- Implementierung eines stark vereinfachten Datenbanksystems:
  - Relationales Schema
  - Einbenutzer-System
  - vereinfachte SQL-Anfragesprache (*SimpleSQL*)

(1. Block: 28. Okt. - 2. Dez.)

- Anwendung von Optimierungstechniken zur Beschleunigung von Anfragen

(2. Block: 2. Dez. - 13. Jan.)

- Unterstützung von Transaktionen

(3. Block: 13. Jan. - 10. Feb.)

Inhalt

Exkurs: Git

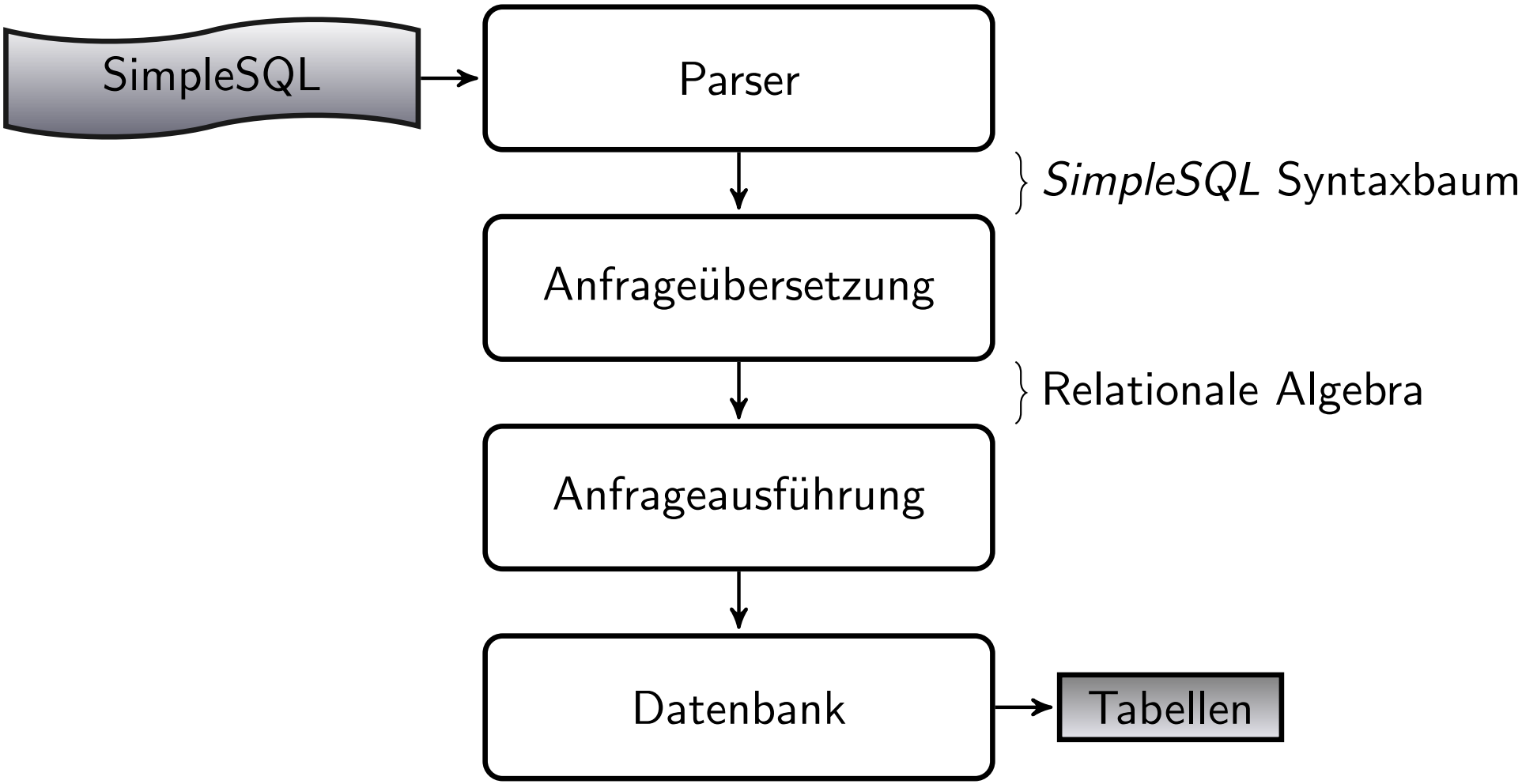
- System zur verteilten Versionsverwaltung von Dateien
- Arbeitskopie $\overset{commit}{\rightarrow}$ Lokales Repository $\overset{push}{\rightarrow}$ Entferntes Repository
- Ein Repository erstmalig auschecken
  `git clone benutzername@host:/pfad/zum/repository`
- Eine Datei zum Index des Repository hinzufügen
  `git add <dateiname>`
- Änderungen an überwachten Dateien ins Repository übernehmen
  `git commit -m "Commit-Nachricht"`
- Änderungen in das entfernte Repository übernehmen
  `git push origin master`
- Lokales Repository auf die neuesten Änderungen aktualisieren
  `git pull`
- GUI-Clients: *EGit* (Eclipse-Plugin), *TortoiseGit* (Windows), ...

Exkurs: Git

- Das für die Übung notwendige Start-Projekt ist bereits in einem Git-Repository hinterlegt
- Server: *server01.pool.ifis.uni-luebeck.de*
  Pfad: */home/anf_grXX/repo/anfrage.git* (mit $XX \in \{01, 02, ...\}$)
  Benutzername und Passwort wird in der ersten Übung vergeben

Inhalt

Einleitung - 1. Block



Einleitung - 1. Block

SimpleSQL - Grammatik

- `SimpleSQL := Query`
  `| Update`
  `| Delete`
  `| Insert`
  `| CreateTable`
  `| DropTable`
- Datei *SimpleSQL.jj* beschreibt diese Grammatik
- Es werden keine verschachtelten Anfragen betrachtet
- Nur ein Datentyp: `VARCHAR`

Einleitung - 1. Block

Beispiele in *SimpleSQL*

- ```
  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")
  ```

Einleitung - 1. Block

SimpleSQL-Parser

- *SimpleSQL.jj* beschreibt die Grammatik
- *Java Tree Builder (JTB)* und *Java Compiler Compiler (JCC)* parsen den übergebenen Ausdruck, prüfen ihn auf syntaktisch Korrektheit und erzeugen einen Syntaxbaum
- Syntaxbaum setzt sich aus folgenden, bereits generierten Klassen zusammen:
  - `parser.*`
  - `parser.gene.*`
  - `parser.syntaxtree.*`
  - `parser.visitor.*`

Einleitung - 1. Block

#1 *SimpleSQL* in Relationale Algebra überführen

- `SELECT <spaltennamen>`
  `FROM <tabellenname_1>,...,<tabellenname_n>`
  `WHERE <bedingung>`
- $\pi_{<spaltennamen>}($
  $\sigma_{<bedingung>}($
  $< tabellenname\_1 > \times ... \times < tabellenname\_n >$
  $)$
  $)$

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

#1a *SimpleSQL* parsen

- Beispiel

```
String simpleSQL = "...";
SimpleSQLParser parser =
    new SimpleSQLParser (new StringReader(simpleSQL));
parser.setDebugALL(false);
try {
    CompilationUnit cu = parser.CompilationUnit();
} catch (ParserException e) { ... }
```

- `CompilationUnit cu` enthält den geparsten Syntaxbaum der *SimpleSQL*-Anfrage
- Syntaxbaum besteht aus Klassen des Packages `parser.syntaxtree`

Einleitung - 1. Block

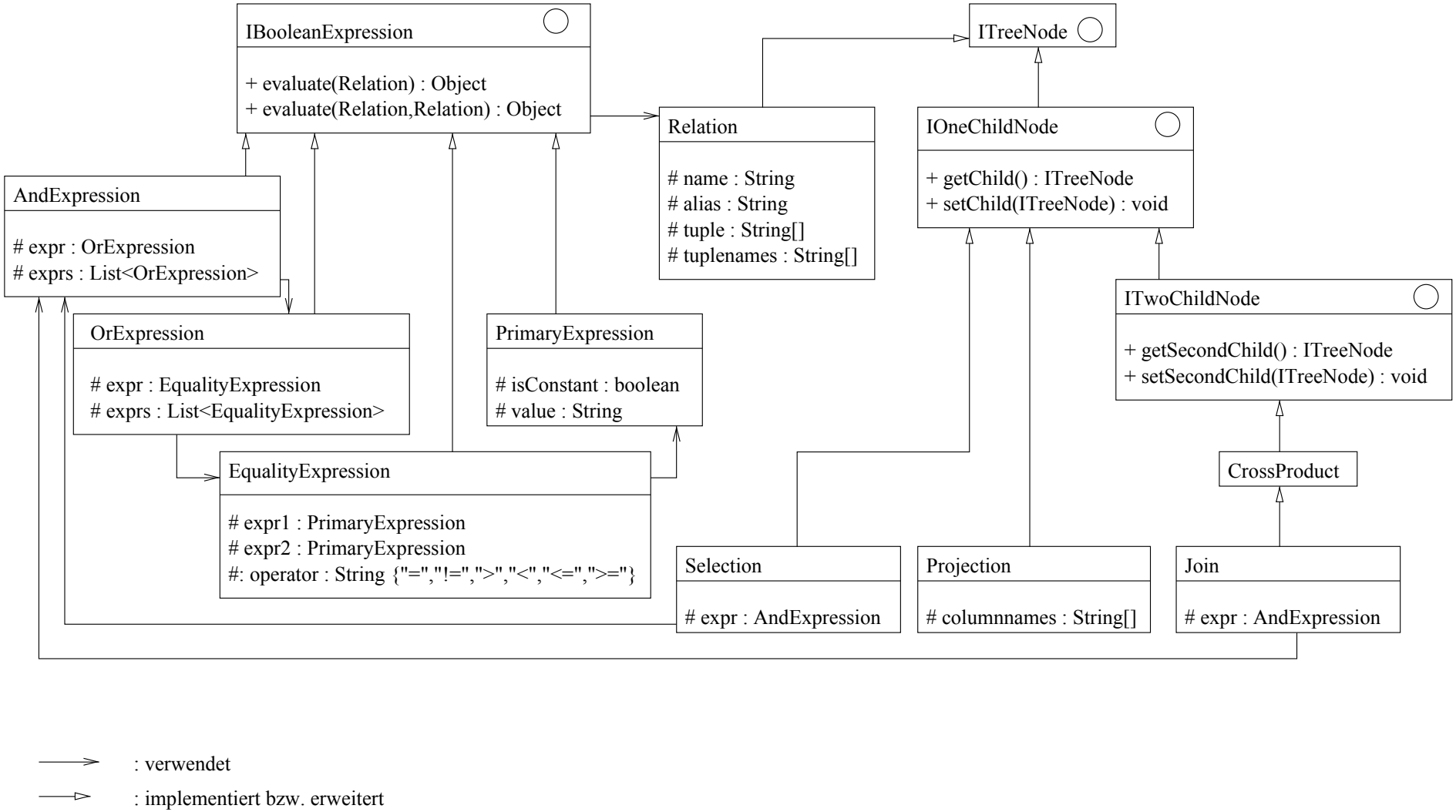
#1b Knoten des Syntaxbaumes besuchen

- Trennung des Algorithmus und Objektstruktur durch das **Visitor**-Entwurfsmuster
- Hinzufügen neuer Operationen auf der Struktur ohne diese verändern zu müssen
- Ausgangsprojekt verfügt bereits über einen allgemeinen Visitor (`parser.visitor.ObjectDepthFirst`), welcher alle Knoten des Syntaxbaumes besucht
- Erweiterung durch einen eigenen, speziellen Visitor, welche ausgehend von den besuchten Knoten die Anfragebaum in der relationalen Algebra erstellt
- Verwendung und Aufruf des Visitors

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

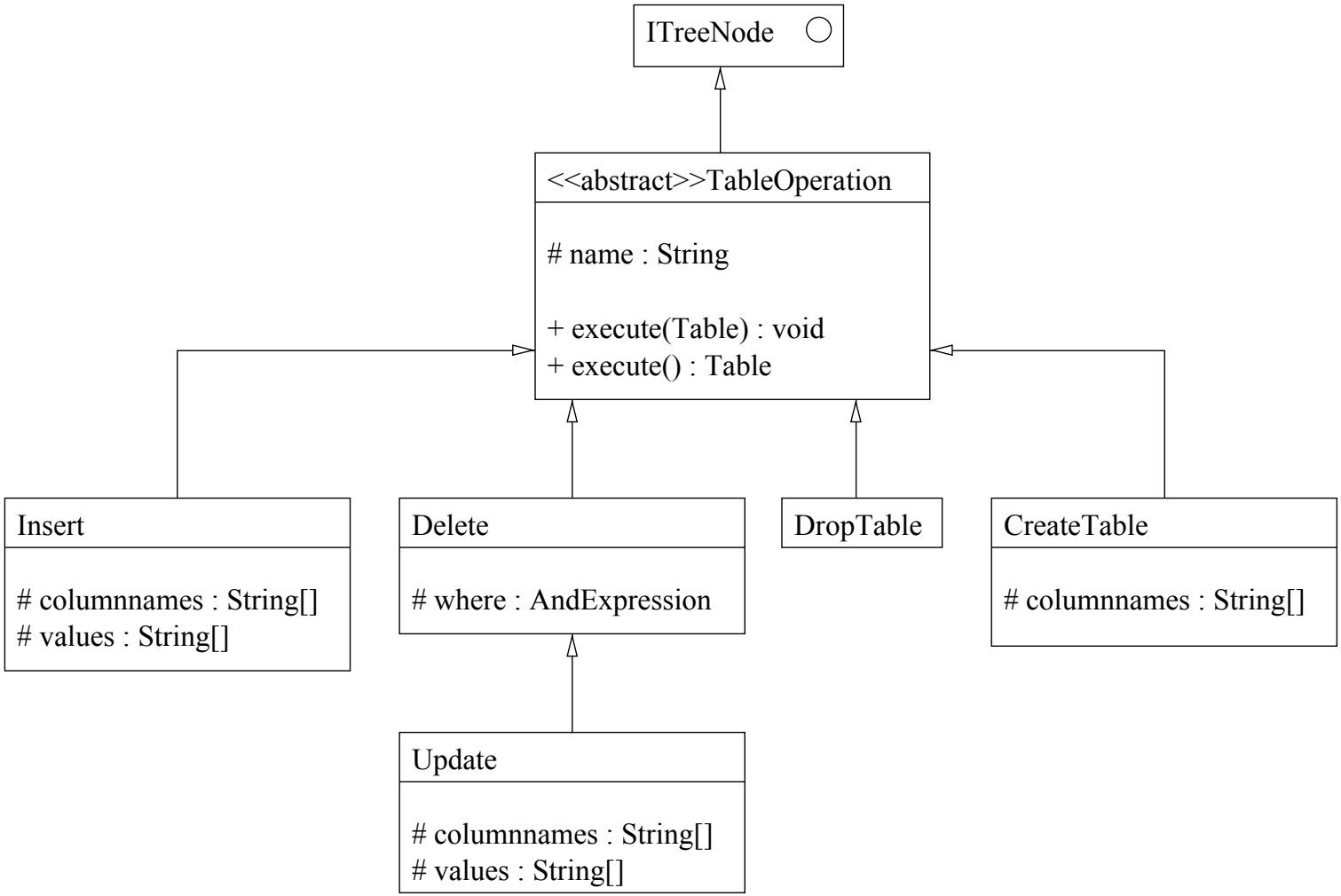
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

#1c Relationale Algebra durch Klassen modellieren - Lesen



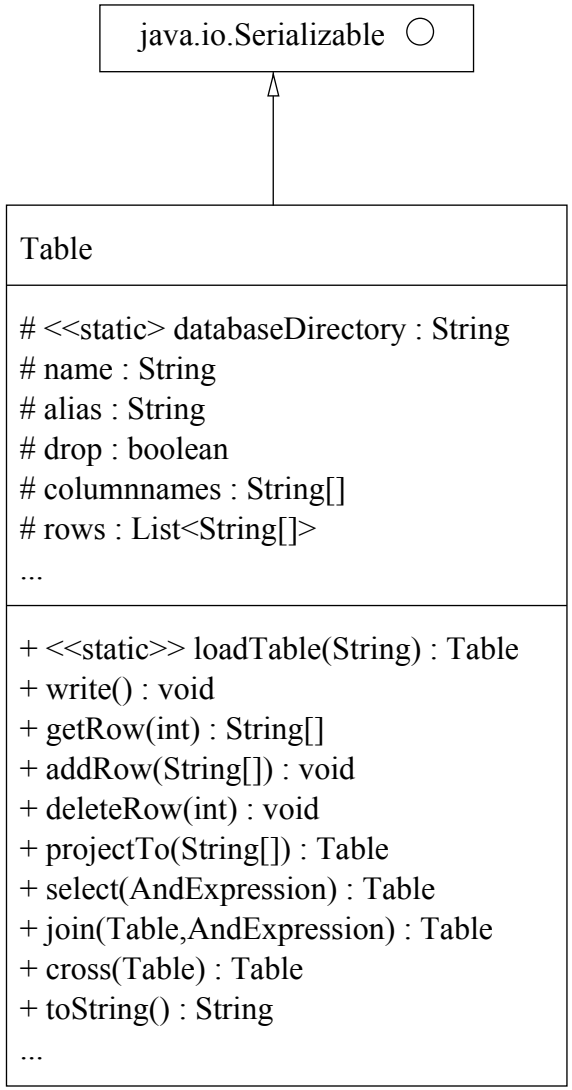
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

#1c Relationale Algebra durch Klassen modellieren - Schreiben



UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

#1d Tabelleninhalt als Datei speichern



#1d Tabelleninhalt als Datei speichern

- 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();
```

Einleitung - 1. Block

#1e Einfache Ausführung

- Zusammenführung der bisherigen Teilschritte
  1. *SimpleSQL* parsen
  2. Syntaxbaum durchlaufen und in Ausdruck der relationalen Algebra überführen
  3. Ausdruck der relationalen Algebra als Operationen auf den Tabellen der Datenbank ausführen
- dazu Klasse `main.Main` vervollständigen

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

#1f Kostenmaß - wie teuer ist eine Anfrage?

- Später werden die Anfragen optimiert, daher wird ein Kostenmaß benötigt um die Effizienzsteigerung messen zu können
- Eigene Ergebnisklassen, welche neben der Ergebnistabelle auch die Kosten zurückliefert

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

$Zeilen_s$ = Zeilen, welche die Bedingung erfüllen

Einleitung - 1. Block

Beispieldaten

- Schema
  - `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)`
- Beispieldatenbank erzeugen und Daten einfügen $\rightarrow$ *kundendb.txt*
- Beispielanfragen $\rightarrow$ *sql.txt*