Script generated by TTT

Title: groh: profile1 (22.05.2015)

Date: Fri May 22 09:17:43 CEST 2015

Duration: 87:59 min

Pages: 64

Anfragen über mehrere Relationen

Beispiel: Welcher Prof liest Mäeutik?

```
select Name, Titel
        from Professoren, Vorlesungen
        where PersNr=gelesenVon and Titel='Mäeutik';
```

$\Pi_{Name,Titel}\, \sigma_{PersNr=gelesenVon \wedge Titel='Mäeutik'}(Professoren \times Vorlesungen)$

Anfragen über mehrere Relationen

Beispiel: Welcher Prof liest Mäeutik?

```
select Name, Titel
        from Professoren, Vorlesungen
        where PersNr=gelesenVon and Titel='Mäeutik';
```

$\Pi_{Name,Titel}\, \sigma_{PersNr=gelesenVon \wedge Titel='Mäeutik'}(Professoren \times Vorlesungen)$

129

Anfragen über mehrere Relationen

Beispiel: Welcher Prof liest Mäeutik?

```
select Name, Titel
        from Professoren, Vorlesungen
        where PersNr=gelesenVon and Titel='Mäeutik';
```

$\Pi_{Name,Titel}\, \sigma_{PersNr=gelesenVon \wedge Titel='Mäeutik'}(Professoren \times Vorlesungen)$

129

Anfragen über mehrere Relationen: Tupelvariablen

Bsp.: Welche Studenten hören welche Vorlesungen?

```
select Name, Titel
from Studenten, hören, Vorlesungen
where Studenten.MatrNr = hören.MatrNr and
                hören.VorlNr = Vorlesungen.VorlNr;
```

alternativ (empfohlen):Einsatz von Tupelvariablen):

```
select s.Name, v.Titel
from Studenten s, hören h, Vorlesungen v
where s. MatrNr = h. MatrNr and
                h.VorlNr = v.VorlNr
```

(entspricht ρ Operator)

Mengenoperationen union, intersect, minus

- Beispiel für union:

```
( select Name
  from Assistenten )
union
( select Name
  from Professoren);
```

$\Pi_{Name}(Assistenten) \cup \Pi_{Name}(Professoren)$

- intersect, minus: analog

Geschachtelte Unteranfragen: exists

Existenzquantor exists

Bsp.: Profs die keine Vorlesung halten:

```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```


Geschachtelte Unteranfragen: exists

Existenzquantor exists

Bsp.: Profs die keine Vorlesung halten:

```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```

Geschachtelte Unteranfragen: exists

Existenzquantor exists

Bsp.: Profs die keine Vorlesung halten:

```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```


Geschachtelte Unteranfragen: exists

Existenzquantor exists

Bsp.: Profs die keine Vorlesung halten:

```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```

134

Geschachtelte Unteranfragen: in

Bsp.: Profs die keine Vorlesung halten:

Korrelation

```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```

Bsp.: Profs die keine Vorlesung halten: Variante mit in

```
select p.Name
from Professoren p
where p.PersNr not in ( select gelesenVon
                        from Vorlesungen );
```


Geschachtelte Unteranfragen: in

Bsp.: Profs die keine Vorlesung halten:

Korrelation

```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```

Bsp.: Profs die keine Vorlesung halten: Variante mit in

```
select p.Name
from Professoren p
where p.PersNr not in ( select gelesenVon
                        from Vorlesungen );
```

136

Der Vergleich mit "all"

Bsp: Finde die Studis, die am längsten studieren!

```
select Name
from Studenten
where Semester >= all ( select Semester
                        from Studenten);
```


Der Vergleich mit "all"

Bsp: Finde die Studis, die am längsten studieren!

```
select Name
from Studenten
where Semester >= all ( select Semester
                        from Studenten);
```

138

Der Vergleich mit "all"

Bsp: Finde die Studis, die am längsten studieren!

```
select Name
from Studenten
where Semester >= all ( select Semester
                        from Studenten);
```

effizienter:

```
select Name
from Studenten
where Semester >= ( select max(Semester)
                    from Studenten);
```


Gruppierung und Aggregation

- γ nicht mehr Teil der relationalen Algebra.
  Dennoch sehr wichtiger Operator (siehe `select` mit `groupby`)
- $\gamma_{A;\,f}(E)$ gruppiert nach Attribut(menge) A und wendet auf jede Gruppe (eine Menge von) Aggregatfunktion(en) f (count(), max(), min(), sum(), avg() etc.) an

$\gamma_{Semester;\ count(*)}(Studenten)$

Semester	count(*)
18	1
12	1
10	1
8	1
6	1
3	1
2	2

$\gamma_{gelesenVon;\ count(*),sum(SWS)}(Vorlesungen)$

gelesenVon	count(*)	sum(SWS)
2125	3	10
2126	3	8
2133	1	2
2134	1	2
2137	2	8

Gruppierung und Aggregation

- γ nicht mehr Teil der relationalen Algebra.
  Dennoch sehr wichtiger Operator (siehe `select` mit `groupby`)

- $\gamma_{A;\,f}(E)$ gruppiert nach Attribut(menge) A und wendet auf jede Gruppe (eine Menge von) Aggregatfunktion(en) f (count(), max(), min(), sum(), avg() etc.) an

$\gamma_{\text{Semester; count(*)}}(\text{Studenten})$

$\gamma_{\text{Semester;count(*)}}(\text{Studenten})$	
Semester	count(*)
18	1
12	1
10	1
8	1
6	1
3	1
2	2

$\gamma_{\text{gelesenVon; count(*),sum(SWS)}}(\text{Vorlesungen})$

$\gamma_{\text{gelesenVon; count(*),sum(SWS)}}(\text{Vorlesungen})$		
gelesenVon	count(*)	sum(SWS)
2125	3	10
2126	3	8
2133	1	2
2134	1	2
2137	2	8

Der Vergleich mit "all"

Bsp: Finde die Studis, die am längsten studieren!

```
select Name
from Studenten
where Semester >= all ( select Semester
                        from Studenten);
```

effizienter:

```
select Name
from Studenten
where Semester >= ( select max(Semester)
                    from Studenten);
```

Bemerkung (1) : neben **all** gibt es auch **any**

Bemerkung (2): entspricht NICHT dem Allquantor;
"Studis die **alle** 4std Vorl. hoeren" geht damit NICHT

Aggregatfunktionen und Gruppierung

- Aggregatfunktionen: **avg, max, min, count, sum, median**

Beispiel:

```
select avg(Semester) from Studenten;
```

- Gruppierung:

Beispiel:

```
select gelesenVon, sum (SWS)
    from Vorlesungen
    group by gelesenVon;
```


Aggregatfunktionen und Gruppierung

- Aggregatfunktionen: **avg, max, min, count, sum, median**

Beispiel:

```
select avg(Semester) from Studenten;
```

- Gruppierung:

Beispiel:

```
select gelesenVon, sum (SWS)
    from Vorlesungen
    group by gelesenVon;
```

Gruppierung: Beispiel 2: Welcher C4 Professor liest im Durchschnitt lange Vorlesungen?

```
select gelesenVon, Name, sum(SWS)
        from Vorlesungen, Professoren
        where gelesenVon = PersNr and Rang = 'C4'
        group by gelesenVon, Name
                having avg(SWS) >= 3;
```


Aggregatfunktionen und Gruppierung

Gruppierung: Beispiel 2: Welcher C4 Professor liest im Durchschnitt lange Vorlesungen?


```
select gelesenVon, Name, sum(SWS)
        from Vorlesungen, Professoren
        where gelesenVon = PersNr and Rang = 'C4'
        group by gelesenVon, Name
                having avg(SWS) >= 3;
```


Vorlesung x Professoren

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2125	Sokrates	C4	226
5001	Grundzüge	4	2137	2126	Russel	C4	232
...	...	...	...	...	...	...	...
5001	Grundzüge	4	2137	2137	Kant	C4	7
5041	Ethik	4	2125	2125	Sokrates	C4	226
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7


```
select gelesenVon, Name, sum(SWS)
        from Vorlesungen, Professoren
        where gelesenVon = PersNr and Rang = 'C4'
        group by gelesenVon, Name
                having avg(SWS) >= 3;
```


Selektion mit where Bedingung

$\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren)

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2137	Kant	C4	7
5041	Ethik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

$\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren)

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2137	Kant	C4	7
5041	Ethik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7


```
select gelesenVon, Name, sum(SWS)
        from Vorlesungen, Professoren
        where gelesenVon = PersNr and Rang = 'C4'
        group by gelesenVon, Name
                having avg(SWS) >= 3;
```


Gruppierung mit groupBy

$\gamma_{gelesenVon, Name}(\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren))

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7
5001	Grundzüge	4	2137	2137	Kant	C4	7

$\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren)

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2137	Kant	C4	7
5041	Ethik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

```
select gelesenVon, Name, sum(SWS)
  from Vorlesungen, Professoren
  where gelesenVon = PersNr and Rang = 'C4'
  group by gelesenVon, Name
    having avg(SWS) >= 3;
```

Gruppierung mit groupBy

$\gamma_{gelesenVon, Name}(\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren))

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7
5001	Grundzüge	4	2137	2137	Kant	C4	7

$\gamma_{gelesenVon, Name; avg(SWS)}(\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren))

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum	avg(SWS)
5041	Ethik	4	2125	2125	Sokrates	C4	226	3.333
5049	Mäeutik	2	2125	2125	Sokrates	C4	226	
4052	Logik	4	2125	2125	Sokrates	C4	226	
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232	2.666
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232	
5216	Bioethik	2	2126	2126	Russel	C4	232	
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7	4
5001	Grundzüge	4	2137	2137	Kant	C4	7	

```
select gelesenVon, Name, sum(SWS)
  from Vorlesungen, Professoren
  where gelesenVon = PersNr and Rang
  group by gelesenVon, Name
    having avg(SWS) >= 3;
```

Selektion der Gruppen mit having Bedingung

$\sigma_{avg(SWS)>3}(\gamma_{gelesenVon, Name; avg(SWS)}(\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren)))

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum	avg(SWS)
5041	Ethik	4	2125	2125	Sokrates	C4	226	3.333
5049	Mäeutik	2	2125	2125	Sokrates	C4	226	
4052	Logik	4	2125	2125	Sokrates	C4	226	
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7	4
5001	Grundzüge	4	2137	2137	Kant	C4	7	

$\sigma_{avg(SWS)>3}(\gamma_{gelesenVon, Name; avg(SWS)}(\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren)))

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum	avg(SWS)
5041	Ethik	4	2125	2125	Sokrates	C4	226	3.333
5049	Mäeutik	2	2125	2125	Sokrates	C4	226	
4052	Logik	4	2125	2125	Sokrates	C4	226	
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7	4
5001	Grundzüge	4	2137	2137	Kant	C4	7	

```
select gelesenVon, Name, sum(SWS)
  from Vorlesungen, Professoren
  where gelesenVon = PersNr and Rang
  group by gelesenVon, Name
    having avg(SWS) >= 3;
```

Berechnung von Aggregatfunktion sum

$\sigma_{avg(SWS)>3}(\gamma_{gelesenVon, Name; avg(SWS),sum(SWS)}(\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren)))

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum	avg(SWS)	sum(SWS)
5041	Ethik	4	2125	2125	Sokrates	C4	226	3.333	10
5049	Mäeutik	2	2125	2125	Sokrates	C4	226		
4052	Logik	4	2125	2125	Sokrates	C4	226		
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7	4	8
5001	Grundzüge	4	2137	2137	Kant	C4	7		

Projektion

$\Pi_{gelesenVon,sum(SWS)}(\sigma_{avg(SWS)>3}(\gamma_{gelesenVon, Name; avg(SWS),sum(SWS)}(\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren))))

gelesen Von	Name	sum(SWS)
2125	Sokrates	10
2137	Kant	8

$\sigma_{avg(SWS)>3}(\gamma_{gelesenVon, Name; avg(SWS),sum(SWS)}(\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren)))

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum	avg(SWS)	sum(SWS)
5041	Ethik	4	2125	2125	Sokrates	C4	226	3.333	10
5049	Mäeutik	2	2125	2125	Sokrates	C4	226		
4052	Logik	4	2125	2125	Sokrates	C4	226		
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7	4	8
5001	Grundzüge	4	2137	2137	Kant	C4	7		

Projektion

$\Pi_{gelesenVon,sum(SWS)}(\sigma_{avg(SWS)>3}(\gamma_{gelesenVon, Name; avg(SWS),sum(SWS)}(\sigma_{gelesenVon = PersNr \wedge Rang = 'C4'}$(Vorlesung x Professoren))))

gelesen Von	Name	sum(SWS)
2125	Sokrates	10
2137	Kant	8

Aggregatfunktionen und Gruppierung

- Eigentliche Verarbeitung ist nicht so schematisch
- Anwendung von Aggregatfunktionen: SQL erzeugt eigentlich pro Gruppe nur **ein** Ergebnistupel

  ⇒ Es dürfen - außer den aggregierten – in der **select**-Klausel nur Attribute aufgeführt werden, die auch in der **group by**-Klausel aufgeführt werden!
- Beispiel: Wie viele Vorlesungen besucht jeder Student?

```
select count(h.VorlNr), h.MatrNr, s.name
    from hoeren h, Studenten s
    where h.MatrNR = s.MatrNR
    group by h.MatrNr, s.Name
```


Aggregatfunktionen und Gruppierung

- Eigentliche Verarbeitung ist nicht so schematisch
- Anwendung von Aggregatfunktionen: SQL erzeugt eigentlich pro Gruppe nur **ein** Ergebnistupel

  ⇒ Es dürfen - außer den aggregierten – in der **select**-Klausel nur Attribute aufgeführt werden, die auch in der **group by**-Klausel aufgeführt werden!
- Beispiel: Wie viele Vorlesungen besucht jeder Student?

```
select count(h.VorlNr), h.MatrNr, s.name
    from hoeren h, Studenten s
    where h.MatrNR = s.MatrNR
    group by h.MatrNr, s.Name
```


Aggregatfunktionen und Gruppierung

- Eigentliche Verarbeitung ist nicht so schematisch
- Anwendung von Aggregatfunktionen: SQL erzeugt eigentlich pro Gruppe nur **ein** Ergebnistupel

  ⇒ Es dürfen - außer den aggregierten – in der **select**-Klausel nur Attribute aufgeführt werden, die auch in der **group by**-Klausel aufgeführt werden!
- Beispiel: Wie viele Vorlesungen besucht jeder Student?

```
select count(h.VorlNr), h.MatrNr, s.name
    from hoeren h, Studenten s
    where h.MatrNR = s.MatrNR
    group by h.MatrNr, s.Name
```


Geschachtelte Anfragen

1. Unteranfrage in der **where**-Klausel

Beispiel: Welche Prüfungen sind besser als durchschnittlich verlaufen?

```
select *
   from prüfen p
   where p.Note < ( select avg (Note)
                    from prüfen );
```

Geschachtelte Anfragen

1. Unteranfrage in der **where**-Klausel

Beispiel: Welche Prüfungen sind besser als durchschnittlich verlaufen?

```
select *
   from prüfen p
   where p.Note < ( select avg (Note)
                    from prüfen );
```


Geschachtelte Anfragen

2. Unteranfrage in der **select**-Klausel

- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt

- Beispiel: Ermittele Lehrbelastung der Profs:

```
select p.PersNr, p.Name, ( select sum (v.SWS) as Lehrbelastung
                           from Vorlesungen v
                           where v.gelesenVon=p.PersNr )
   from Professoren p;
```

Unteranfrage ist hier korreliert (greift auf Attribute der umschließenden Anfrage zu)

Korrelierte versus unkorrelierte Unteranfragen

Bsp: Welche Studenten sind älter als Professoren?

- korrelierte Formulierung:

```
select s.*
   from Studenten s
   where exists ( select p.*
                  from Professoren p
                  where p.GebJahr > s.GebJahr );
```

- unkorrelierte Formulierung:

```
select s.*
   from Studenten s
   where s.GebJahr < (select max (p.GebJahr)
                      from Professoren p);
```

Vorteil: Unteranfrage muss nur einmal ausgewertet werden.

Korrelierte versus unkorrelierte Unteranfragen

Bsp: Welche Studenten sind älter als Professoren?

- korrelierte Formulierung:

```
select s.*
   from Studenten s
   where exists ( select p.*
                  from Professoren p
                  where p.GebJahr > s.GebJahr );
```

- unkorrelierte Formulierung:

```
select s.*
   from Studenten s
   where s.GebJahr < (select max (p.GebJahr)
                      from Professoren p);
```

Vorteil: Unteranfrage muss nur einmal ausgewertet werden.

Entschachtelung korrelierter Unteranfragen durch Join

Bsp.: Welcher Assistent hat einen Boss der jünger ist als er selbst?

```
select a.*
from Assistenten a
where exists
      ( select p.*
        from Professoren p
        where a.Boss = p.PersNr and p.GebJahr>a.GebJahr )
```

--> Entschachtelung durch Join:

```
select a.*
from Assistenten a, Professoren p
where a.Boss=p.PersNr and p.GebJahr > a.GebJahr;
```


Entschachtelung korrelierter Unteranfragen durch Join

Bsp.: Welcher Assistent hat einen Boss der jünger ist als er selbst?

```
select a.*
from Assistenten a
where exists
      ( select p.*
        from Professoren p
        where a.Boss = p.PersNr and p.GebJahr>a.GebJahr )
```

--> Entschachtelung durch Join:

```
select a.*
from Assistenten a, Professoren p
where a.Boss=p.PersNr and p.GebJahr > a.GebJahr;
```


Entschachtelung korrelierter Unteranfragen durch Join

Bsp.: Welcher Assistent hat einen Boss der jünger ist als er selbst?

```
select a.*
from Assistenten a
where exists
      ( select p.*
        from Professoren p
        where a.Boss = p.PersNr and p.GebJahr>a.GebJahr )
```

--> Entschachtelung durch Join:

```
select a.*
from Assistenten a, Professoren p
where a.Boss=p.PersNr and p.GebJahr > a.GebJahr;
```


Geschachtelte Anfragen

3. Unteranfrage in der **from**-Klausel:

--> Verwertung der Ergebnismenge einer Unteranfrage:

Beispiel: Wer hört mehr als 2 Vorlesungen?

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from ( select s.MatrNr, s.Name, count(*) as VorlAnzahl
       from Studenten s, hören h
       where s.MatrNr=h.MatrNr
       group by s.MatrNr, s.Name )  tmp
where tmp.VorlAnzahl > 2;
```

Geschachtelte Anfragen

3. Unteranfrage in der **from**-Klausel:

--> Verwertung der Ergebnismenge einer Unteranfrage:

Beispiel: Wer hört mehr als 2 Vorlesungen?

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from ( select s.MatrNr, s.Name, count(*) as VorlAnzahl
       from Studenten s, hören h
       where s.MatrNr=h.MatrNr
       group by s.MatrNr, s.Name ) tmp
where tmp.VorlAnzahl > 2;
```


Allquantifizierung durch count-Aggregation

- Allquantifizierung kann immer auch durch eine **count**-Aggregation ausgedrückt werden.
- Bsp: MatrNr der Studenten, die *alle* Vorlesungen hören:

```
select h.MatrNr
  from hören h
  group by h.MatrNr
       having count (*) = (select count (*) from Vorlesungen);
```

(count(*) nach having zählt alle Tupel einer Gruppe)

Allquantifizierung durch count-Aggregation

- Allquantifizierung kann immer auch durch eine **count**-Aggregation ausgedrückt werden.
- Bsp: MatrNr der Studenten, die *alle* Vorlesungen hören:

```
select h.MatrNr
  from hören h
  group by h.MatrNr
       having count (*) = (select count (*) from Vorlesungen);
```

(count(*) nach having zählt alle Tupel einer Gruppe)

Allquantifizierung durch count-Aggregation

Bsp: MatrNr der Studenten die alle vierstündigen Vorlesungen hören:

```
select s.MatrNr
from Studenten s
where not exists
    ( select v.*
      from Vorlesungen v
      where v.SWS=4 and not exists
          ( select h.*
            from hören h
            where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr ) );
```

folgt sinngemäß der logischen Identität $\forall x\, Bed(x) \equiv \neg\exists x\, \neg Bed(x)$
(wobei x eine Variable und $Bed(x)$ ein prädikatenlogischer Ausdruck ist)

Allquantifizierung durch count-Aggregation

Bsp: MatrNr der Studenten die alle vierstündigen Vorlesungen hören:

```
select s.MatrNr
from Studenten s
where not exists
      ( select v.*
        from Vorlesungen v
        where v.SWS=4 and not exists
              ( select h.*
                from hören h
                where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr ) );
```

folgt sinngemäß der logischen Identität $\forall x\ Bed(x) \equiv \neg\exists x\ \neg Bed(x)$
(wobei x eine Variable und $Bed(x)$ ein prädikatenlogischer Ausdruck ist)

Allquantifizierung durch count-Aggregation

Bsp: MatrNr der Studenten die alle vierstündigen Vorlesungen hören:

```
select s.MatrNr
from Studenten s
where not exists
      ( select v.*
        from Vorlesungen v
        where v.SWS=4 and not exists
              ( select h.*
                from hören h
                where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr ) );
```

folgt sinngemäß der logischen Identität $\forall x\ Bed(x) \equiv \neg\exists x\ \neg Bed(x)$
(wobei x eine Variable und $Bed(x)$ ein prädikatenlogischer Ausdruck ist)

Allquantifizierung durch count-Aggregation

Bsp: MatrNr der Studenten die alle vierstündigen Vorlesungen hören:

```
select s.MatrNr
from Studenten s
where not exists
      ( select v.*
        from Vorlesungen v
        where v.SWS=4 and not exists
              ( select h.*
                from hören h
                where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr ) );
```

oder:

```
select h.MatrNr
from hören h, Vorlesungen v
where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr and v.SWS = 4
group by h.MatrNr
      having count (*) = ( select count (*)
                           from Vorlesungen v
                           where v.SWS=4 );
```

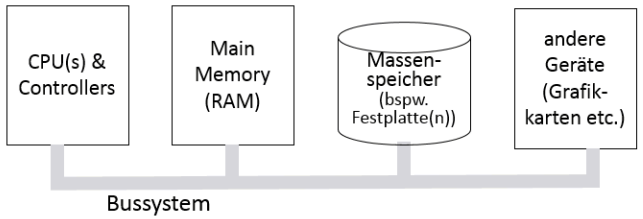

Programme

- *Erinnerung:* Informatik: Gegenstand: Informationen + deren Verarbeitung (Repräsentation, Speicherung, Abbildung, Übertragung etc.) mit Computersystemen.
- Bisher: DBMS: realisieren eine Art, Daten systematisch zu speichern und ermöglichen Anfragen auf Datenbasis (Eingabe: bspw. SQL select, Ausgabe: Antworttabelle).
- Allgemeinere Art der Verarbeitung von Informationen? --> mit Programmen (in formalen Programmiersprachen) auf (programmierbaren) Computersystemen.

Achtung: Informatik $\neq$ Programmieren !!!!!!!!!

Programme

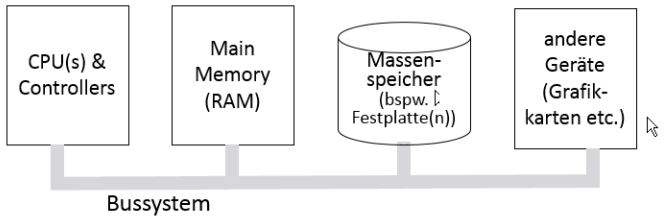
- Programme[1] bestehen aus Instruktionen (Deklarationen und Statements) in einer formalen Programmiersprache (--> Syntax und Semantik)
- Statements[1] werden schrittweise auf einem Computersystem ausgeführt. Schritte ändern den Zustand (bspw. Speicherzustand) des Computersystems.
- Typisches Computersystem: Von-Neumann-Architektur (stark vereinfacht):



[1]: bezieht sich hauptsächlich auf imperative Sprachen.

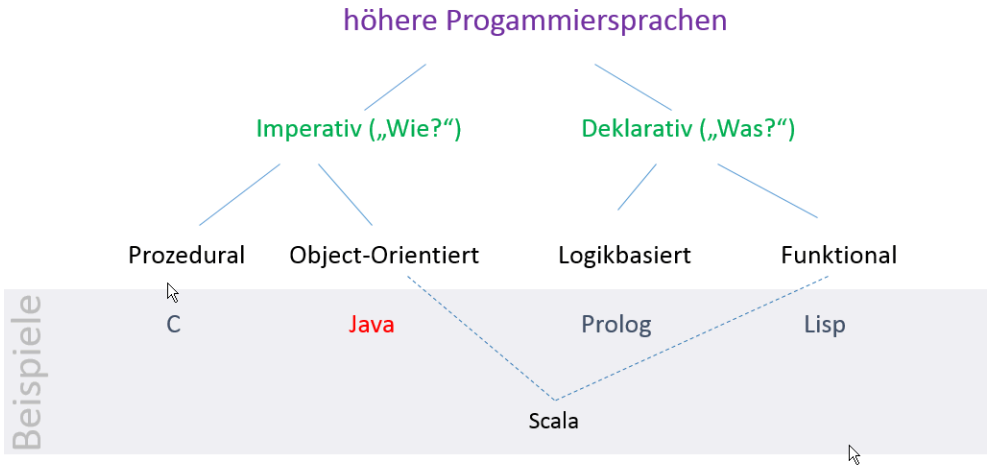
Programme

- Programme[1] bestehen aus Instruktionen (Deklarationen und Statements) in einer formalen Programmiersprache (--> Syntax und Semantik)
- Statements[1] werden schrittweise auf einem Computersystem ausgeführt. Schritte ändern den Zustand (bspw. Speicherzustand) des Computersystems.
- Typisches Computersystem: Von-Neumann-Architektur (stark vereinfacht):



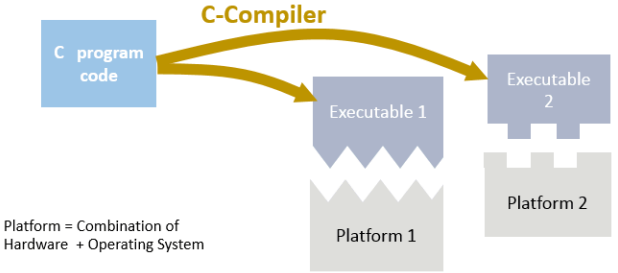
[1]: bezieht sich hauptsächlich auf imperative Sprachen.

Höhere Programmiersprachen

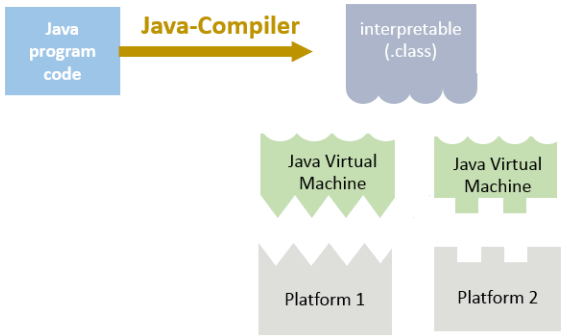


Java + Virtual Machine

Compiled Languages: (Bsp.: C)

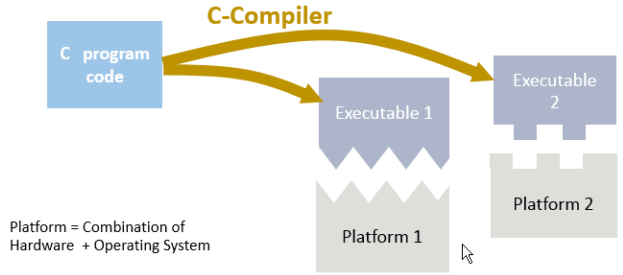


„Virtual Machine Languages“: (Bsp.: Java)

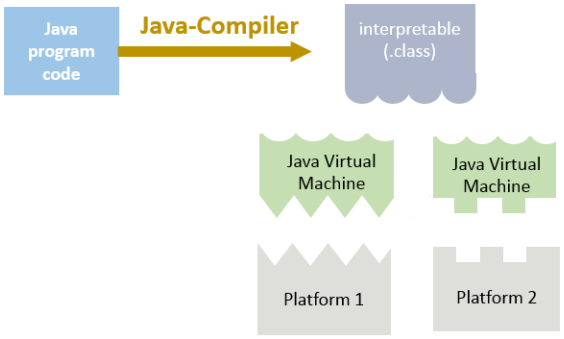


Java + Virtual Machine

Compiled Languages: (Bsp.: C)

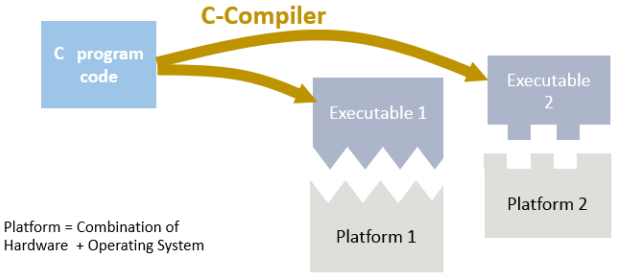


„Virtual Machine Languages“: (Bsp.: Java)

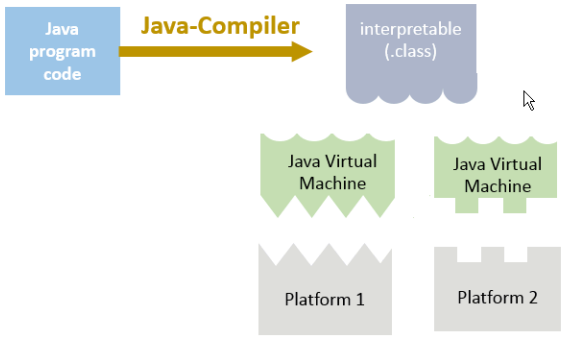


Java + Virtual Machine

Compiled Languages: (Bsp.: C)



„Virtual Machine Languages“: (Bsp.: Java)



7

Programm und vereinfachtes Speichermodell

```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
...
```

Vereinfachtes Speicher-Modell

Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
1124			Daten
1125	horst		
1126	heiner		
1127			
1128	fritz		
4027			Programm
4028		int horst;	
4029		int heiner;	
4030		int fritz;	
4029		horst = 101;	
4030		heiner = 2;	

Programm und vereinfachtes Speichermodell

```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
...
```

Vereinfachtes Speicher-Modell

Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
1124			Daten
1125	horst		
1126	heiner		
1127			
1128	fritz		
4027			Programm
4028		int horst;	
4029		int heiner;	
4030		int fritz;	
4029		horst = 101;	
4030		heiner = 2;	

8

Programm und vereinfachtes Speichermodell

```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
...
```

Vereinfachtes Speicher-Modell

Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
		⋮	
1124			Daten
1125	horst		Daten
1126	heiner		Daten
1127			Daten
1128	fritz		Daten
		⋮	
4027			Programm
4028		int horst;	Programm
4029		int heiner;	Programm
4030		int fritz;	Programm
4029		horst = 101;	Programm
4030		heiner = 2;	Programm
		⋮	

Programm und vereinfachtes Speichermodell

```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
...
```

Vereinfachtes Speicher-Modell

Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
		⋮	
1124			Daten
1125	horst		Daten
1126	heiner		Daten
1127			Daten
1128	fritz		Daten
		⋮	
4027			Programm
4028		int horst;	Programm
4029		int heiner;	Programm
4030		int fritz;	Programm
4029		horst = 101;	Programm
4030		heiner = 2;	Programm
		⋮	

Programm und vereinfachtes Speichermodell

```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
...
```

Vereinfachtes Speicher-Modell

Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
		⋮	
1124			Daten
1125	horst		Daten
1126	heiner		Daten
1127			Daten
1128	fritz		Daten
		⋮	
4027			Programm
4028		int horst;	Programm
4029		int heiner;	Programm
4030		int fritz;	Programm
4029		horst = 101;	Programm
4030		heiner = 2;	Programm
		⋮	

Kontrollfluss: Bedingte Verzweigung

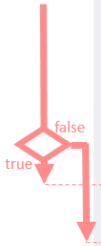
```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
if(heiner == 2)
      horst = 10;
else
      horst = 11;
fritz = 17;
...
```

false
true

Vereinfachtes Speicher-Modell

Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
		⋮	
1124			Daten
1125	horst	2000	Daten
1126	heiner	2	Daten
1127			Daten
1128	fritz	103	Daten
		⋮	
4027			Programm
4028		int horst;	Programm
4029		int heiner;	Programm
4030		int fritz;	Programm
4029		horst = 101;	Programm
4030		heiner = 2;	Programm
		⋮	

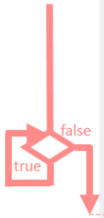
Kontrollfluss: Bedingte Verzweigung




```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
if(heiner == 2)
      horst = 10;
else
      horst = 11;
fritz = 17;
...
```

Vereinfachtes Speicher-Modell			
Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
		⋮	
1124			Daten
1125	horst	2000	Daten
1126	heiner	2	Daten
1127			Daten
1128	fritz	103	Daten
		⋮	
4027			Programm
4028		int horst;	Programm
4029		int heiner;	Programm
4030		int fritz;	Programm
4029		horst = 101;	Programm
4030		heiner = 2;	Programm
		⋮	

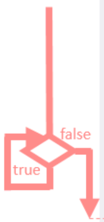
Kontrollfluss: Schleife




```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
while(heiner > 0)
   heiner = heiner - 1;
fritz = 17;
...
```

Vereinfachtes Speicher-Modell			
Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
		⋮	
1124			Daten
1125	horst	2000	Daten
1126	heiner	2	Daten
1127			Daten
1128	fritz	103	Daten
		⋮	
4027			Programm
4028		int horst;	Programm
4029		int heiner;	Programm
4030		int fritz;	Programm
4029		horst = 101;	Programm
4030		heiner = 2;	Programm
		⋮	

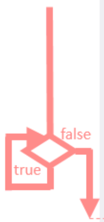
Kontrollfluss: Schleife




```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
while(heiner > 0)
   heiner = heiner - 1;
fritz = 17;
...
```

Vereinfachtes Speicher-Modell			
Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
		⋮	
1124			Daten
1125	horst	2000	Daten
1126	heiner	2	Daten
1127			Daten
1128	fritz	103	Daten
		⋮	
4027			Programm
4028		int horst;	Programm
4029		int heiner;	Programm
4030		int fritz;	Programm
4029		horst = 101;	Programm
4030		heiner = 2;	Programm
		⋮	

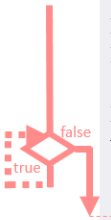
Kontrollfluss: Schleife




```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
while(heiner > 0)
   heiner = heiner - 1;
fritz = 17;
...
```

Vereinfachtes Speicher-Modell			
Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
		⋮	
1124			Daten
1125	horst	2000	Daten
1126	heiner	2	Daten
1127			Daten
1128	fritz	103	Daten
		⋮	
4027			Programm
4028		int horst;	Programm
4029		int heiner;	Programm
4030		int fritz;	Programm
4029		horst = 101;	Programm
4030		heiner = 2;	Programm
		⋮	

Kontrollfluss: Schleife




```
...
int horst;
int heiner;
int fritz;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
while(heiner > 0)
   heiner = heiner - 1;
fritz = 17;
...
```

Vereinfachtes Speicher-Modell

Zell-Nr (Adresse)	Zell-Name (Variablenname)	Zell-Inhalt	
		⋮	
1124			Daten
1125	horst	2000	
1126	heiner	0	
1127			
1128	fritz	17	
		⋮	
4027			Programm
4028		int horst;	
4029		int heiner;	
4030		int fritz;	
4029		horst = 101;	
4030		heiner = 2;	
		⋮	