Datenbanksysteme

Wintersemester 2015/16

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

Wiederholung Infosys: So ging es los ...

Abbilden eines Teilaspekts der realen Welt

- Was wollen wir Abbilden?
- Welcher Grad an Details?
- Welche Entitäten spielen eine Rolle?
- Und welche Rolle spielen sie?
- Wie sind Entitäten miteinander verbunden?

Anforderungsanalyse

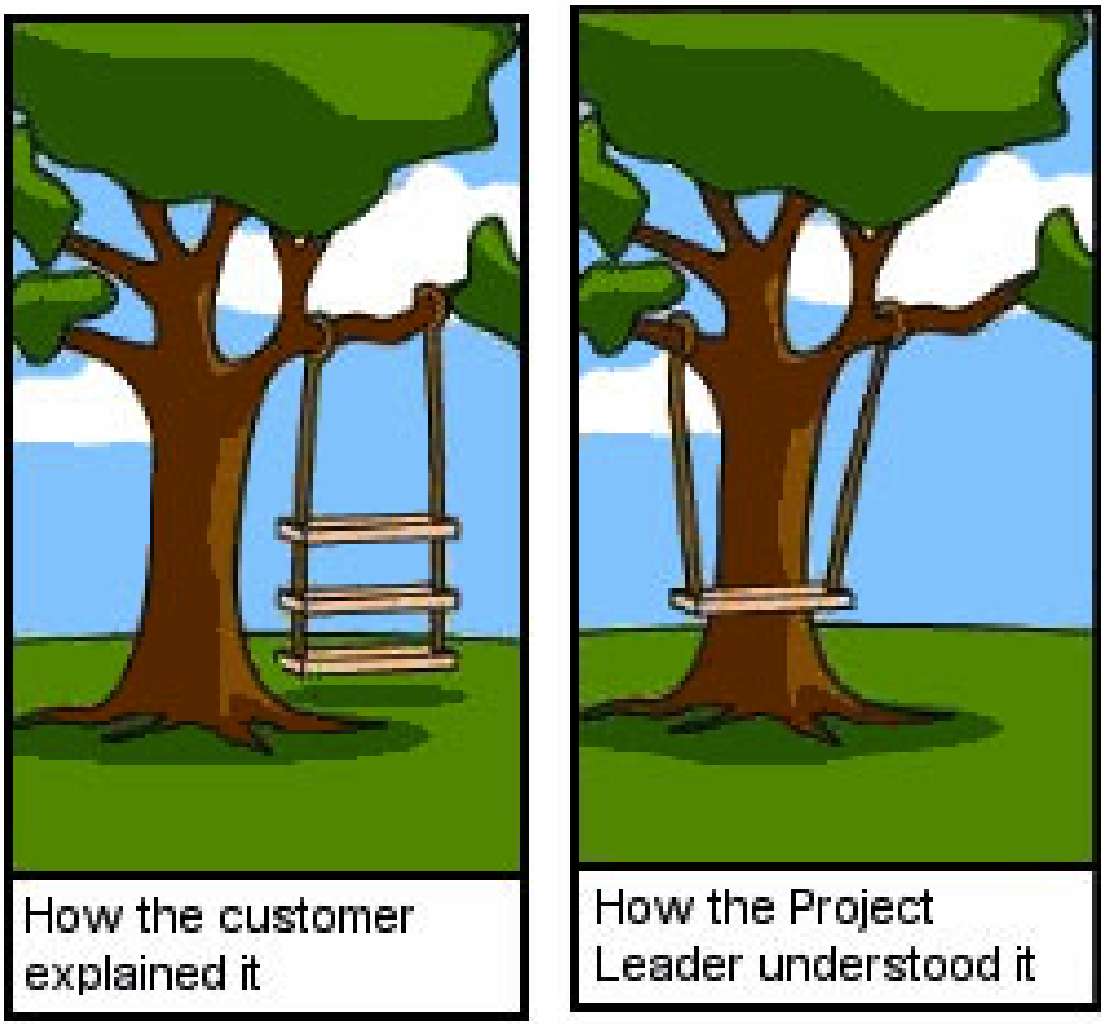
Reale Welt: Universität

→ Anforderungsanalyse

Pflichtenheft

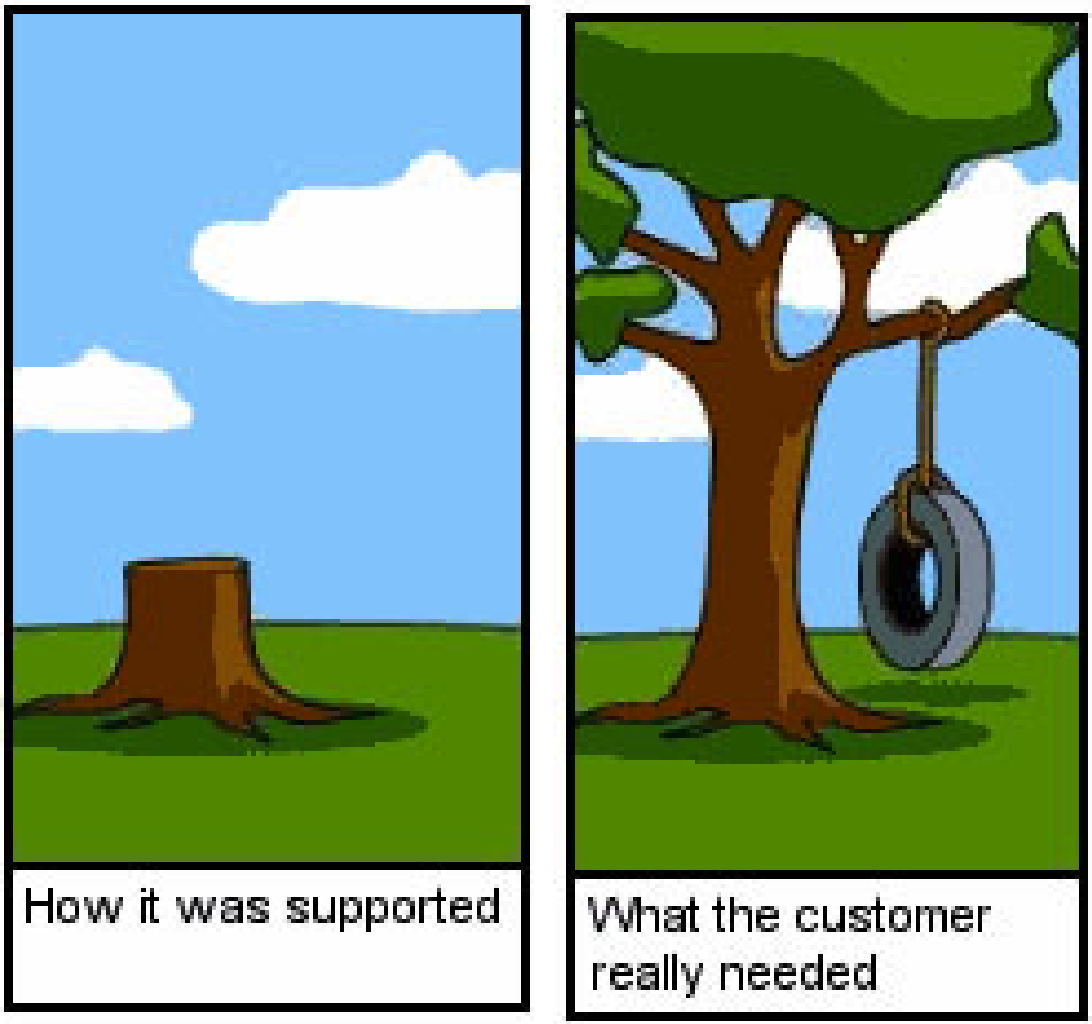
- „Studenten hören Vorlesungen"
- „Professoren halten Vorlesungen"
- „Studenten könnten anhand ihrer Matrikelnummer eindeutig identifiziert werden"
- ...

Entity/Relationship-Modellierung



Quelle:
https://astheqaworldturns.files.wordpress.com/2011/03/requirements.jpg

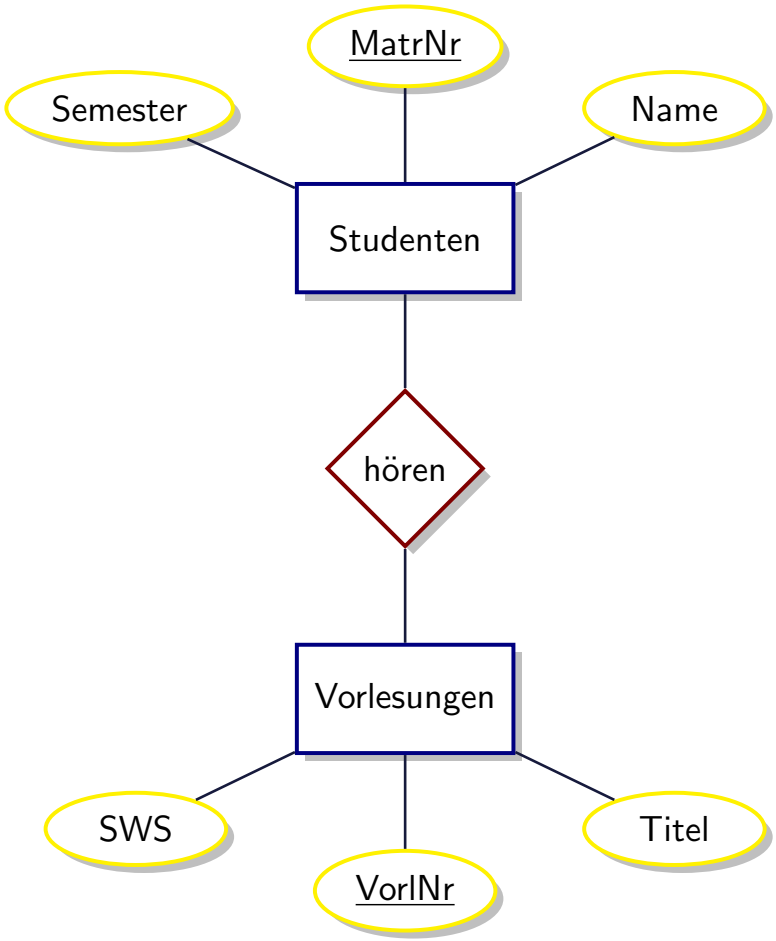
Entity/Relationship-Modellierung



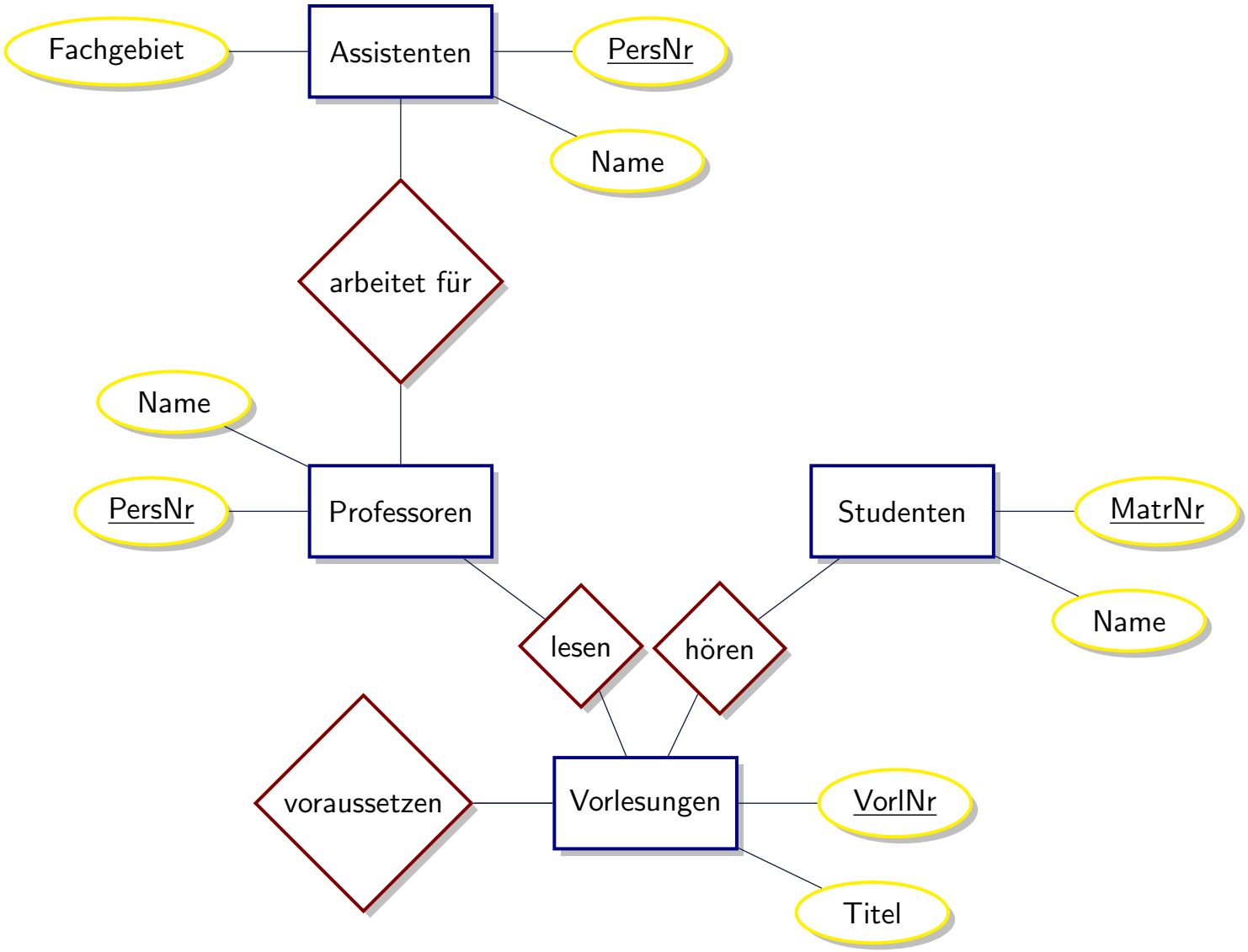
Quelle: https://astheqaworldturns.files.wordpress.com/2011/03/requirements.jpg

Entity/Relationship-Modellierung

1. Entität → Entitätstyp
2. Beziehung → Beziehungstyp
3. Attribut (Eigenschaft)
4. Schlüssel (Identifikation)
5. Rolle



Beispiel: Universitätsschema



Relationales Modell

Abbilden des ER-Modells in Relationen:

Zum Beispiel:

- **Kunden**(ID, Telefon, Adresse, Name)
- **Kaufen**(Wert, Datum, Preis, Verkäufer, Auto, Kunde)
- **Verkaufen**(Datum, Wert, Kommission, Kunde, Verkäufer, Auto)
- ...

oder ...

- **Studenten** (MatrNr, Name)
- **Hören** (MatrNr, VorlNr)
- **Vorlesungen** (VorlNr, Titel)

Beispiel: Die relationale Uni-DB

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen			
VorlNr	Titel	SWS	gelesen von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Assistenten			
PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Die Operatoren der relationalen Algebra

- Selektion σ
- Projektion π
- Kreuzprodukt $\times$
- Join (Verbund) $\bowtie$
- Umbenennung ρ
- Differenz $-$
- Division $\div$
- Aggregation γ
- Vereinigung $\cup$
- Schnitt $\cap$
- Semi-Join (linker) $\ltimes$
- Semi-Join (rechter) $\rtimes$
- linker äußerer Join $⟕$
- rechter äußerer Join $⟖$
- äußerer Join $⟗$

Grundlagen der relationalen Algebra

Achtung:

Es gibt Relationen (Ausprägungen: $R \subseteq D_1 \times D_2 \times ... \times D_n$)
und Relationenschema:{[attributname1: datentyp1, attributname2: datetyp2, ...]}.

Das Schema einer Relation wird auch mit $sch(R)$ oder $\mathcal{R}$ beschrieben.

Die Operatoren und ihre Verwendung:

- Eingabe eines Operators ist eine oder mehrere Relationen.
- Ausgabe ist auch wieder eine Relation.
- D.h., Operatoren sind kombinierbar (mit gewissen Regeln).

Z.B. Selektion

Selektion

$\sigma_{Semester>10}(Studenten)$

$\sigma_{Semester>10}(Studenten)$		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12

- Allgemein: σ_F
- Selektionsprädikat F besteht aus
  - Operatoren: $\vee$ (oder) $\wedge$ (und) $\neg$ (nicht)
  - Arithmetischen Vergleichsoperatoren: $<, \leq, =, >, \geq, \neq$
  - ... und natürlich aus Attributnamen der Argumentrelation oder Konstanten als Operanden

Kartesisches Produkt (Kreuzprodukt)

Gegeben zwei Relationen R und S:

$$R \times S$$

beinhaltet **ALLE!** $|R| * |S|$ möglichen Paare von Tupeln aus R und S.

Enthält viele (oft auch viele unsinnige) Kombinationen!

Das resultierende Schema $sch(R \times S) = sch(R) \cup sch(S) = \mathcal{R} \cup \mathcal{S}$.

Beim Referenzieren der Attribute des resultierenden Schemas wird R.X und S.Y verwendet, insbesondere bei Überlappungen der einzelnen Schemata (=keine Unklarheiten was gemeint ist).

Der natürliche Verbund (Join)

Gegeben zwei Relationen (+ Schemata)

- $R(A_1, ..., A_m, \mathbf{B_1}, ..., \mathbf{B_k})$
- $S(\mathbf{B_1}, ..., \mathbf{B_k}, C_1, ..., C_n)$

$$R \bowtie S = \pi_{A_1,...,A_m,R.B_1,...,R.B_k,C_1,...,C_n}(\sigma_{R.B_1=S.B_1 \wedge ... \wedge R.B_k=S.B_k}(R \times S))$$

$R \bowtie S$											
$\mathcal{R} - \mathcal{S}$				$\mathcal{R} \cap \mathcal{S}$				$\mathcal{S} - \mathcal{R}$			
A_1	A_2	...	A_m	B_1	B_2	...	B_k	C_1	C_2	...	C_n
...	...	...	...	...	...	...	...	...	...	...	...

Mehrere Joins zusammen

- Es können beliebig viele Relationen durch Join Operatoren verknüpft werden.
- Die Reihenfolge spielt dabei keine Rolle (Joins sind kommutativ und assoziativ).

$$(Studenten \bowtie hoeren) \bowtie Vorlesungen$$

$(Studenten \bowtie hoeren) \bowtie Vorlesungen$						
MatrNr	Name	Semester	VorlNr	Titel	SWS	gelesenVon
26120	Fichte	10	5001	Grundzüge	4	2137
27550	Jonas	12	5022	Glaube und Wissen	2	2134
28106	Carnap	3	4052	Wissenschaftstheorie	3	2126
...	...	...	...	...	...	...

Allgemeiner Join (Theta-Join)

Gegeben zwei Relationen (+ Schemata)

- $R(A_1, ..., A_n)$
- $S(B_1, ..., B_m)$

θ ist ein beliebiges Prädikat über den beteiligten Attributen.

$$R \bowtie_\theta S = \sigma_\theta(R \times S)$$

$R \bowtie S$							
$\mathcal{R}$				$\mathcal{S}$			
A_1	A_2	...	A_n	B_1	B_2	...	B_m
...	...	...	...	...	...	...	...

Equi-Join: Join-Prädikat θ darf nur auf Gleichheit ($=$) prüfen.

Weitere Join Typen: Äußere Joins

Diese Join Typen spezifizieren wie mit Tupeln umgegangen wird, die keinen Joinpartner gefunden haben.

- **linker äußerer Join** ($\overset{\frown}{\bowtie}$): auch "partnerlose" Tupel der linken Relation bleiben erhalten
- **rechter äußerer Join** ($\overset{\smile}{\bowtie}$): auch "partnerlose" Tupel der rechten Relation bleiben erhalten
- **vollständiger äußerer Join** ($\overset{\frown\smile}{\bowtie}$): die "partnerlosen" Tupel beider Relationen bleiben erhalten

Natürlicher Join und linker äußerer Join

Natürlicher Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1

Linker äußerer Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie\!\!\!\!-$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	$-$	$-$

Rechter äußerer und äußerer Join

Rechter äußerer Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
$-$	$-$	c_3	d_2	e_2

Äußerer Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\supset\!\bowtie\!\subset$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	$-$	$-$
$-$	$-$	c_3	d_2	e_2

Weitere Join Typen: Semi Joins

Idee: Finde alle Tupel der linken Relation die Joinpartner in der rechten Relation haben.

$$L \ltimes R = \pi_{\mathcal{L}}(L \bowtie R)$$

$L \rtimes R$ ist analog definiert.

Es gilt:

$$L \rtimes R = R \ltimes L$$

allerdings sind Semi-Joins sowie die linken und rechten äußeren Joins **nicht kommutativ!**

Semi-Joins

Semi-Join von L mit R

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\ltimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat		
A	B	C
a_1	b_1	c_1

Semi-Join von R und L

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\rtimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

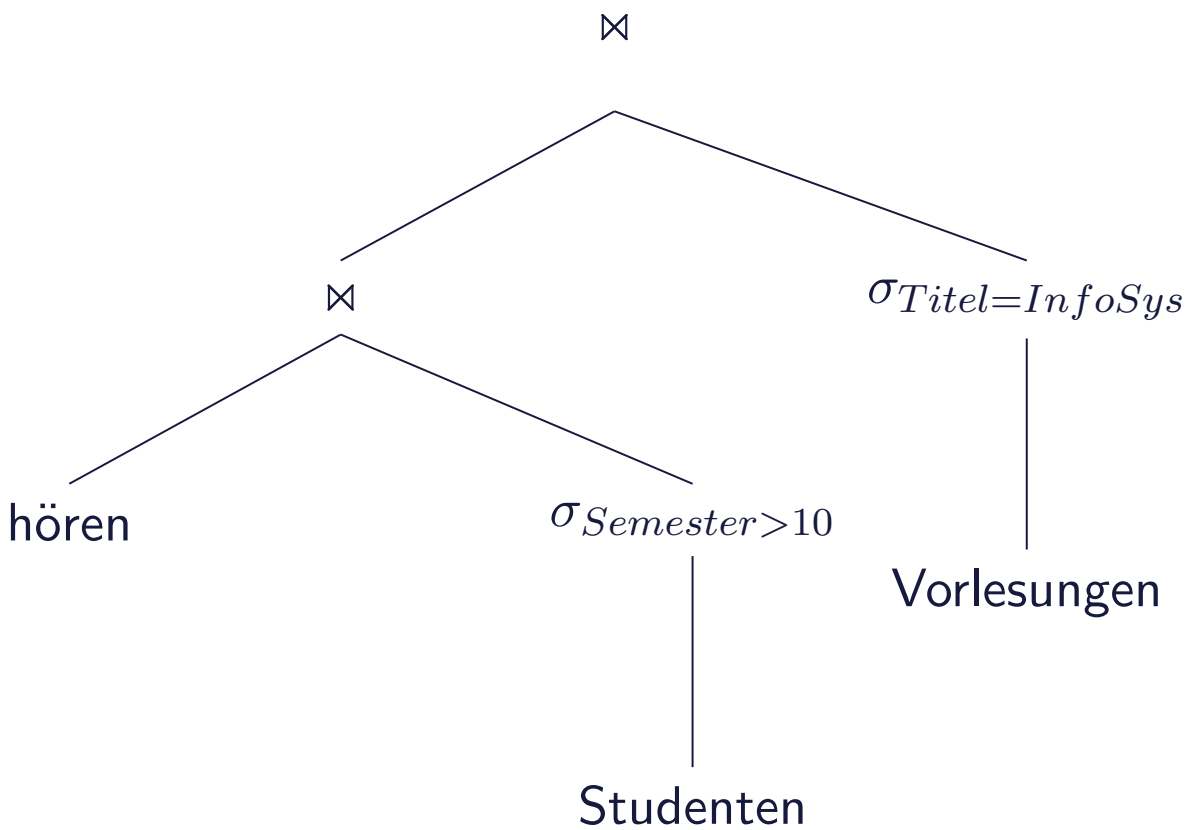
$=$

Resultat		
C	D	E
c_1	d_1	e_1

Operatorbaum-Darstellung

Alternative Darstellung von Ausdrücken der relationalen Algebra. Gerade für größere Ausdrücke viel übersichtlicher.

Beispiel:



SQL

- **deklarative** Anfragesprache ("was" nicht "wie")
- setzte sich aus drei Sprachen zusammen:
- Datendefinitions (DDL)-Sprache:
  - erstellt/ändert das Schema
  - create, alter, drop
- Datenmanipulations (DML)-Sprache
  - ändert Ausprägungen
  - insert, update, delete
- Anfragesprache:
  - berechnet Anfragen auf den Ausprägungen
  - select * from where ...

Relationen vs. Tabellen

- bisher hatten wir Relationen (=Mengen) betrachtet
- in SQL betrachten wir aber Tabellen (=Bags)
- was ist der Unterschied?
- Relationen versus Tabellen:
  - Tabellen können Duplikate enthalten
  - Tabellen können eine Ordnung haben
  - Tabellen haben nicht unbedingt einen Schlüssel

Einfache Datendefinitionen in SQL

Datentypen

- **character** (n), **char** (n)
- **character varying** (n), **varchar** (n)
- **numeric** (p,s), **integer**
- **blob** oder **raw** für sehr große binäre Daten
- **clob** für sehr große String-Attribute
- **date** für Datumsangaben
- **xml** für XML-Dokumente

Datendefinitions (DDL) Sprache

Tabellen erstellen:

```
create table Professoren(
        PersNr      integer not null,
        Name        varchar (10) not null
        Rang        character (2) );
```

Tabellen verändern:

```
alter table Professoren
        add (Raum integer);

alter table Professoren
        modify (Name varchar(30));
```

Datendefinitions (DDL) Sprache

Tabellen löschen:

drop table Professoren;

Datenmanipulationssprache: insert

```
insert into Studenten (MatrNr, Name)
       values (28121, 'Archimedes');

insert into Studenten (Name)
       values ('Meier');

insert into hören
       select MatrNr, VorlNr
       from Studenten, Vorlesungen
       where Titel = 'Logik';
```

Anfragesprache

select <Liste von Spalten>
from <Liste von Tabellen>
where <Bedingung>;

,

select * wählt alle verfügbaren Spalten aus!

,

Relationale Algebra $\rightarrow$ SQL

- Projektion $\pi \rightarrow$ select
- Kreuzprodukt $\times \rightarrow$ from
- Selektion $\sigma \rightarrow$ where

Select from where ...

select PersNr, Name
from Professoren
where Rang = 'C4';

Professoren	
PersNr	**Name**
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Professoren			
PersNr	**Name**	**Rang**	**Raum**
7 2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Anmerkung:

SQL legt nicht fest in welcher Reihenfolge Selektion, Projektion oder Join ausgeführt werden.

Anfragen über mehrere Relationen

Welcher Professor liest "Mäeutik"?

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

In der relationalen Algebra sieht das wie folgt aus:

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

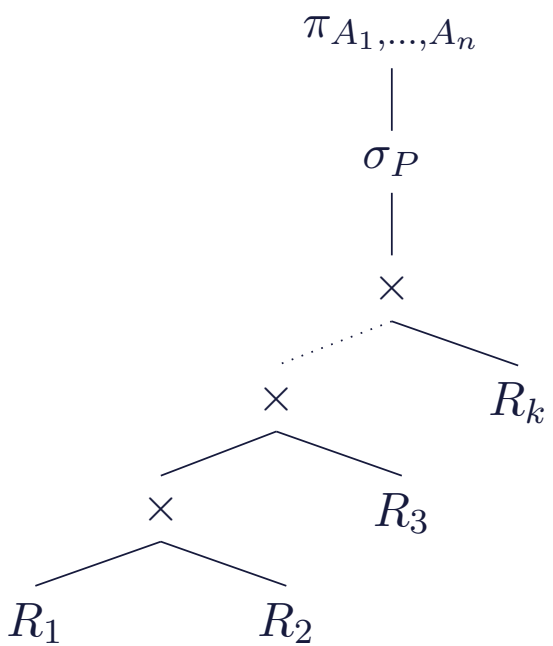
Übersetzung von SQL in die relationale Algebra

Allgemeine Form einer (ungeschachtelten) SQL-Anfrage:

select A_1, ..., A_n
from R_1, ..., R_k
where P;

Übersetzung in die relationale Algebra:

$$\pi_{A_1,...,A_n}(\sigma_P(R_1 \times ... \times R_k))$$



Anfragen über mehrere Relationen

Welche Studenten hören welche Vorlesungen?

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

Alternativ:

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

Vorlesung Datenbanksysteme

Übersicht

Planung

Übersicht, Motivation, Wiederholung

- Motivation
- Organisation der VL und Regeln zum Übungsbetrieb
- Wiederholung aus Infosys (Tabellen, relationale Algebra, SQL)

Übersicht

Anfrageverarbeitung

- Aufbau Festplatte, Kostenmodell, Break Even Point
- Datenbankpuffer Verwaltung (LRU, FIFO, ...)
- Implementierung von Operatoren (Joins, externes Sortieren, ...)
- Kostenschätzung
- Histogramme, Sketches

Übersicht

Anfrageoptimierung

- Prinzip und Regelbasierte Optimierung (Wiederholung Infosys)
- Kostenmodelle für Joins
- Dynamische Programmierung, Greedy Algorithmen

Weitere SQL Konzepte, PL/pgSQL, Trigger

- SQL Fensteranfragen und Rekursion
- User Defined Functions, PL/pgSQL
- Trigger

Übersicht

Indexstrukturen und Hashing

- Mehrdimensionale Objekte
- Hohe Dimensionen
- Quad-Tree, ...

Skylines und Top-k Algorithmen

- Fagin's Algorithmus, Threshold Algorithmus
- Skyline (Operator)

Übersicht

Transaktionsverwaltung

- Transaktionskonzept, Ablauf von Transaktionen
- Commit-Protokolle

Serialisierbarkeit

- Anomalien im Mehrbenutzerbetrieb
- Theorie der Serialisierbarkeit
- Klassen von Historien

Synchronisation und Sperrverfahren

- Sperrprotokolle
- Nicht sperrende Protokolle
- Zweiphasen-Sperrprotokolle

Übersicht

Logging und Recovery

- Fehlermodelle und Recovery-Arten
- Logging-Strategien
- ...

Literaturliste

- A. Kemper und A. Eickler. Datenbanksysteme. Oldenbourg-Verlag, 2012.
- T. Härder und E. Rahm. Datenbanksysteme. Springer. 2011.
- H. Samet. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers. 2006.
- R. Ramakrishnan und J. Gehrke. Database Management Systems. 2003.
- G. Weikum und G. Vossen. Transactional Information Systems. Morgan Kaufmann Publishers. 2002.

Organisatorisches

Vorlesung

- 4 SWS Vorlesung: Wöchentlich, Dienstags und Donnerstags.
  - Dienstags, Raum 42-110, 11:45 bis 13:15 Uhr
  - Donnerstags, Raum 52-207, 15:30 bis 17:00 Uhr
- 2 SWS Übung: Wöchentlich, Mittwochs, 15:30 bis 17:00. Raum 42-110. **Übungsgruppenleiter Manuel Hoffmann**.

Klausur

- Abschlussklausur am 18. März 2016 (Stand der Planung)
- Für die Zulassung zur Klausur bedarf es einer erfolgreichen Teilnahme am Übungsbetrieb.

Regeln zum Übungsbetrieb

- Es gibt **13 Übungsblätter**, die in der wöchentlich stattfindenden Übung besprochen werden.
- Auf diesen Übungsblättern gibt es 40 Aufgaben, die Sie in Vorbereitung auf die Übung lösen müssen.
- **Auf diese 40 Aufgaben gibt es jeweils einen Punkt.**
- **Sie müssen insgesamt 20 Punkte erreichen**, um für die Klausur zugelassen zu werden. Die Punktevergabe funktioniert wie folgt:
- Zu Beginn der Übung markieren Sie auf einer Teilnehmerliste, welche der verbindlichen Aufgaben Sie gelöst haben und in der Übungsstunde präsentieren können.
- In der Übungsstunde wählt der Übungsleiter unter allen markierten Lösungen einen oder mehrere Studierende aus, die entsprechende Aufgabe zu präsentieren. Die Auswahl erfolgt auf Gutdünken des Übungsgruppenleiters.

Regeln zum Übungsbetrieb (2)

- Falls eine Aufgabe durch einen Studierenden als gelöst markiert wird, dieser bei der Aufforderung zur Präsentation passt oder die Präsentation der Aufgabe zeigt, dass sich der Studierende nicht ausreichend mit dem Thema befasst hat, erhält der Studierende **3 Minuspunkte**.
- **Ist die Lösung korrekt oder hat nur leichte Mängel, ist aber auf dem richtigen Weg zur korrekten Lösung, gibt es den vollen Punkt.**
- Die Vergabe der Punkte erfolgt durch subjektive Einschätzung des Übungsleiters und muss nicht begründet werden.

Regeln zum Übungsbetrieb (3)

- Gruppenarbeit generell sinnvoll. Aber: Trotzdem ist die "Abgabe" der Lösungen nicht als Gruppe möglich, sondern pro Studierendem. **Jeder Teilnehmer einer Gruppe muss in der Lage sein, die als gelöst markierten Aufgaben zu präsentieren.**
- Neben den 40 verbindlichen Aufgaben gibt es weitere Aufgaben, deren Bearbeitung freiwillig ist. Wir wünschen und natürlich trotzdem rege Beteiligung bei der Präsentation der Lösungen. **Alle Aufgaben sind verbindlich (1 Punkt) sofern nicht als freiwillig markiert (0 Punkte).**

Vorlesungsfolien und Übungsblätter

Vorlesungsfolien

- Ein in der Regel recht vollständiger Draft wird einen Tag vor der Vorlesung online gestellt.
- Stabile Version folgt dann nach der VL (bzgl. Typos, Anmerkungen, ...) .

Übungsblätter

- Ausgabe Dienstags in der Vorlesung.
- Besprechung in Übung der folgenden Woche (auf Übungsblatt angegeben)

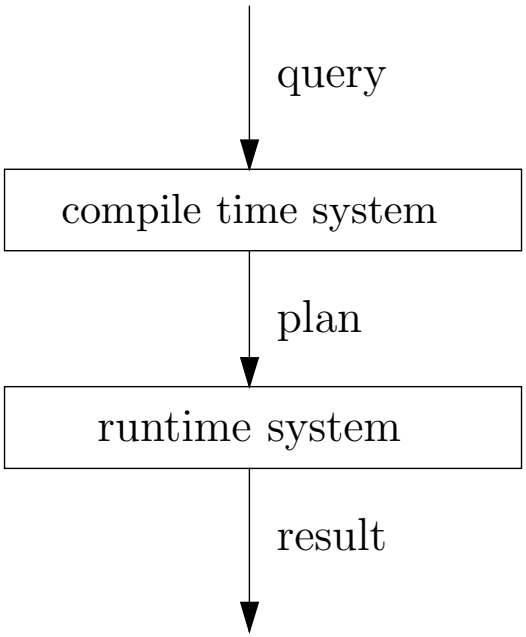
Übersicht und Ausblick

- Bislang bekannt, aus VL Informationssysteme: Optimierung anhand von Regeln auf logischer Ebene
- D.h. gegeben ein Anfrageplan, wie kann dieser optimiert werden (durch "Verschieben" von Operatoren)

Ausblick auf kommende Vorlesungen

- **Physische Datenorganisation**
- **Zugriffskosten**
- **Implementierung der Operatoren**
- **Kostenschätzung**
- **Kostenbasierte Anfrageoptimierung**

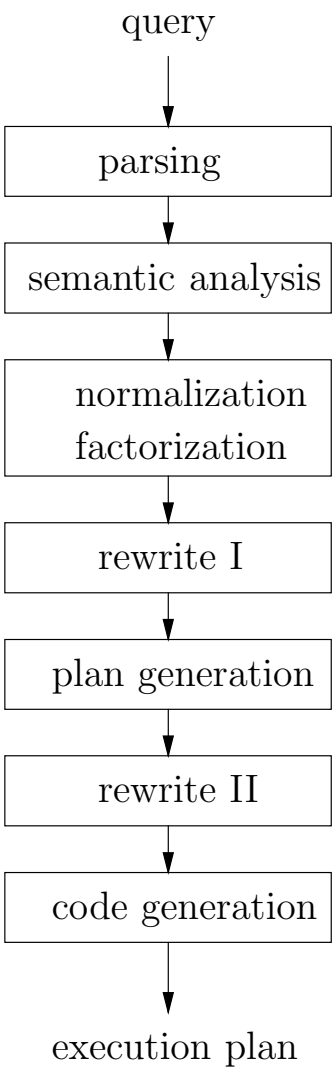
Übersicht Anfrageverarbeitung



- Eingabe: Anfrage als Text (String)
- Kompilierzeitsystem (compile time system) übersetzt und optimiert die Anfrage
- Zwischenprodukt: Anfrage als Anfrageplan (query plan)
- Laufzeitsystem (runtime system) führt die Anfrage aus
- Ausgabe: Ergebnisse der Anfrage

Übersicht, Buch "Working Draft" von Guido Moerkotte (Uni Mannheim):
`http://pi3.informatik.uni-mannheim.de/~moer/querycompiler.pdf`
Vielen Dank an Thomas Neumann (TU München) für die Bereitstellung einiger Folien zu Anfrageoptimierung.

Übersicht Compile Time System



1. Parsen: Erzeugung Abstract Syntax Tree (AST)
2. Schema lookup, Typinferenz
3. Normalisierung, Faktorisierung etc.
4. Auflösen von Sichten, Entschachteln etc.
5. Erzeugung des Ausführungsplans (execution plan)
6. Weitere Optimierung des Plans
7. Erzeugung von ausführbarem Codes

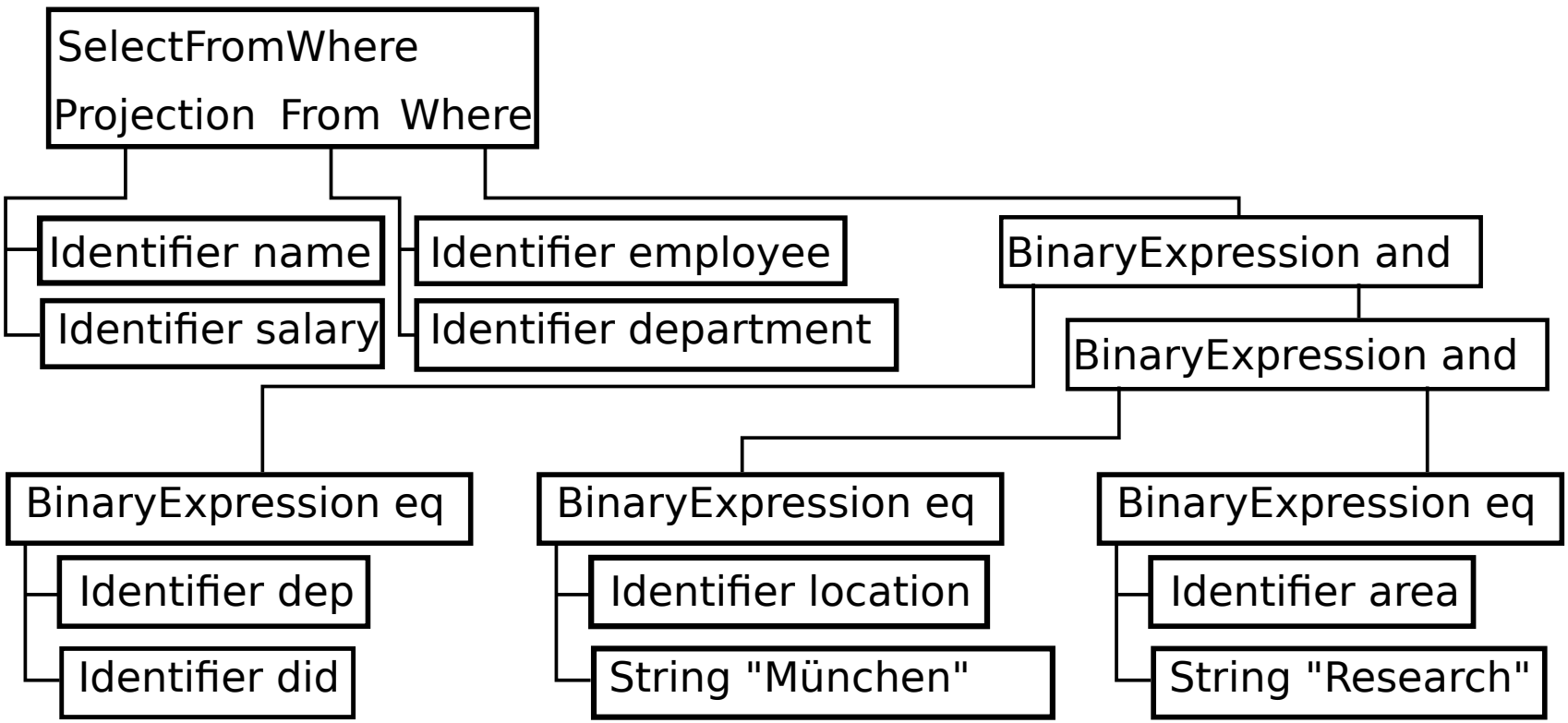
rewrite I, plan generation und rewrite II sind Teile des Anfrageoptimierers

Einfaches Beispiel: Eingabe

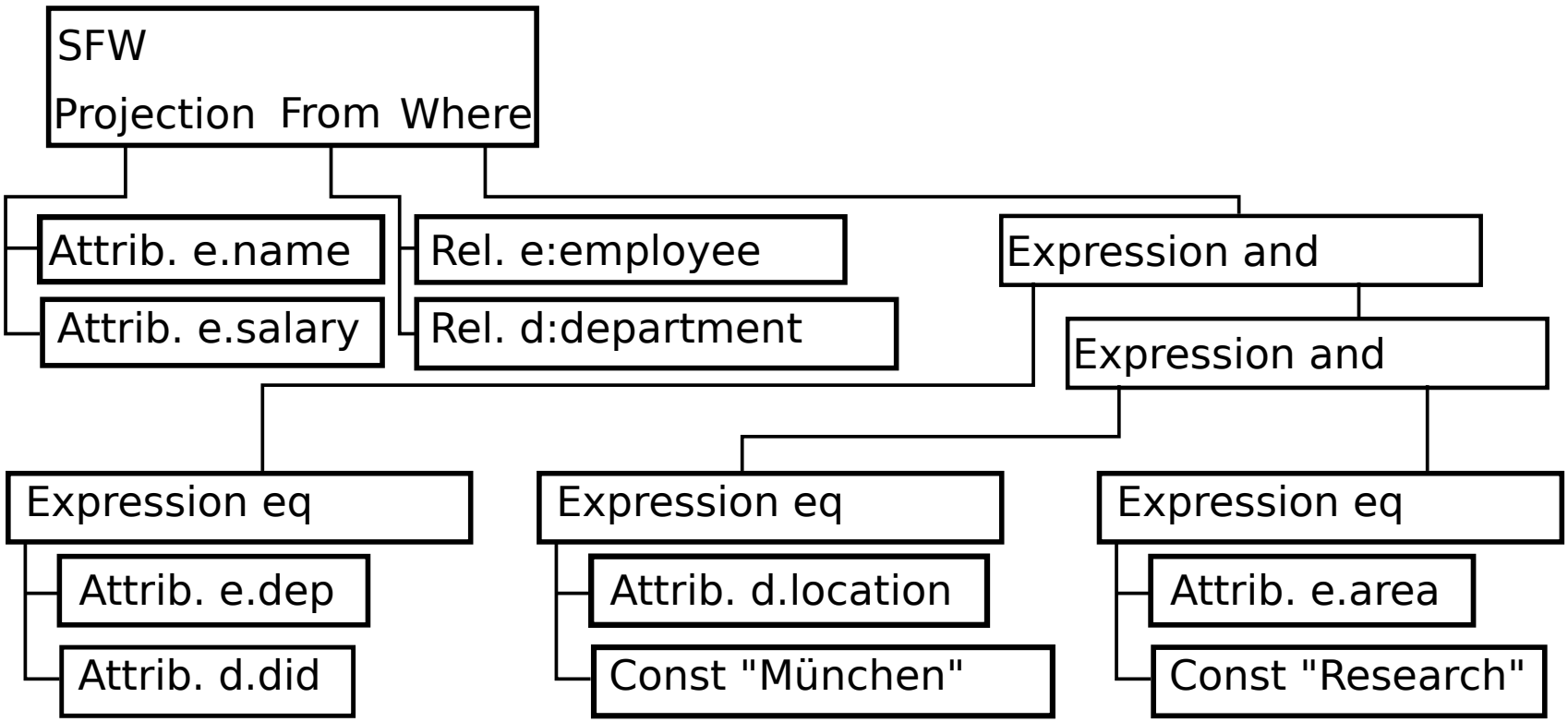
```
select name, salary
from employee, department
where dep=did
      and location="München"
      and area="Research"
```

Beispiel: Parsen der Eingabe

Erstellt einen Abstract Syntax Tree (AST)



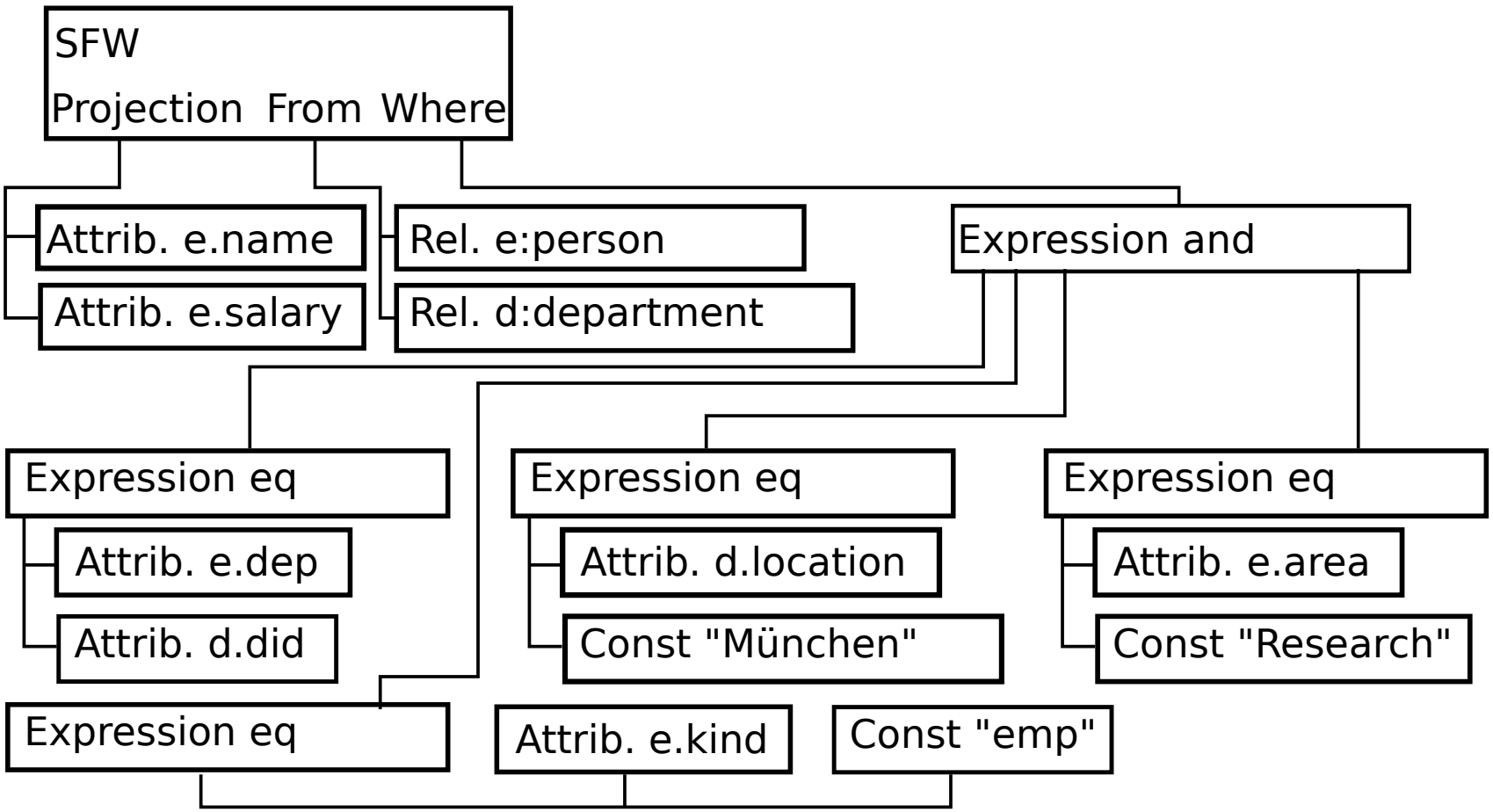
Beispiel: Semantische Analyse



Typen sind hier im Beispiel nicht erwähnt. Ergebnis ist vom Typ $bag < string,number >$

Beispiel: Rewrite I

Auflösen von Sichten, Entschachteln, Optimierung



Beispiel: Code Generierung

Produziert einen ausführbaren Plan

```
<
   @c1 string 0
   @c2 string 0
   @c3 string 0
   @kind string 0
   @name string 0
   @salary float64
   @dep int32
   @area string 0
   @did int32
   @location string 0
   @t1 uint32 local
   @t2 string 0 local
   @t3 bool local
>
[main
   load_string "emp" @c1
   load_string "M\u00fcnchen" @c2
   load_string "Research" @c3
   first_notnull_bool
   <#1 BlockwiseNestedLoopJoin
      memSize 1048576
      [combiner
         unpack_int32 @dep
         eq_int32 @dep @did @t3
         return_if_ne_bool @t3
         unpack_string @name
         unpack_float64 @salary
      ]
[storer
   check_pack 4
   pack_int32 @dep
   pack_string @name
   check_pack 8
   pack_float64 @salary
   load_uint32 0 @t1
   hash_int32 @dep @t1 @t1
   return_uint32 @t1
]
[hasher
   load_uint32 0 @t1
   hash_int32 @did @t1 @t1
   return_uint32 @t1
]
<#2 Tablescan
   segment 1 0 4
   [loader
      unpack_string @kind
      unpack_string @name
      unpack_float64 @salary
      unpack_int32 @dep
      unpack_string @area
      eq_string @kind @c1 @t3
      return_if_ne_bool @t3
      eq_string @area @c3 @t3
      return_if_ne_bool @t3
   ]
>
      <#3 Tablescan
         segment 1 0 5
         [loader
            unpack_int32 @did
            unpack_string @location
            eq_string @location @c2 @t3
            return_if_ne_bool @t3
         ]
      >
   > @t3
   jf_bool 6 @t3
   print_string 0 @name
   cast_float64_string @salary @t2
   print_string 10 @t2
   println
   next_notnull_bool #1 @t3
   jt_bool -6 @t3
]
```

In dieser Vorlesung wird Postgresql verwendet/betrachtet

- "The world's most advanced open source database"
- Einfach zu Installieren
- Gute SQL Unterstützung
- Transaktionen
- Gute Dokumentation (es gibt auch Bücher dazu)

http://www.postgresql.org/

Beispieldatenbanken zum "Üben"

Installieren Sie eine aktuelle Version Postgresql. Folgende Daten stellen wir Ihnen als Postgresql-Dump bereit:

1. Die "Universitäts Datenbank"
2. Eine bereits generierte Version des TPC-H Benchmark-Datensatzes `www.tpc.org/tpch/`

können Sie hier herunterladen:

- `http://dbis.informatik.uni-kl.de/files/teaching/ws1516/dbs/protected/tpch.dmp.gz`
- `http://dbis.informatik.uni-kl.de/files/teaching/ws1516/dbs/protected/uni_db.dmp`

Login wird benötigt (außerhalb der Uni):

- Benutzername: dbs1516
- Passwort[1]:

[1] Wird/Wurde in der VL bekannt gegeben

PSQL: Command Line Interface

- Anfragen ausführen, Schema anlegen, Daten aus CVS Dateien laden, Skripte ausführen, ... siehe Befehlsübersicht via \?

```
postgres@      :~$ psql university
psql (9.3.5, server 9.1.14)
Type "help" for help.

university=# select *
university-# from studenten NATURAL JOIN hoeren
university-# where semester > 6;
 matrnr |  name  | semester | vorlnr
--------+--------+----------+--------
  26120 | Fichte |       10 |   5001
  25403 | Jonas  |       12 |   5022
(2 rows)

university=#
```

Beispiel: Datenbank erzeugen und Dump einspielen

Datenbank erzeugen

- In GUI oder via command line:

  `createdb university`

Daten einspielen

- Mit GUI
- oder via command line (und psql Kommando):

  `psql university < uni_db.dmp`
- Achtung: es gibt auch backup/restore. "Dumps" sind reine SQL Statements und Daten, werden erzeugt mit `pg_dump databasename`

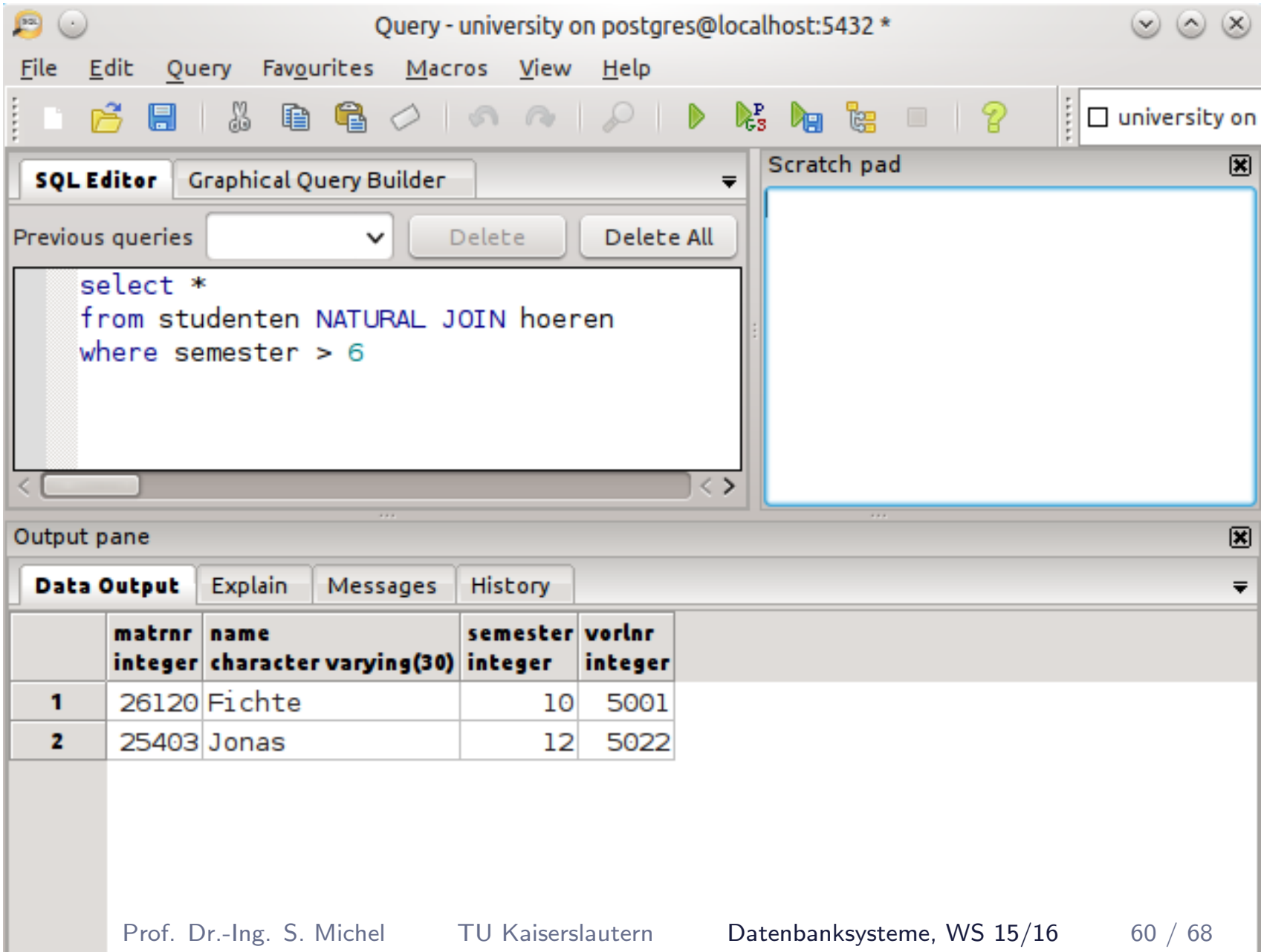
PGAdmin3

User Interface: Download unter `http://www.pgadmin.org/`

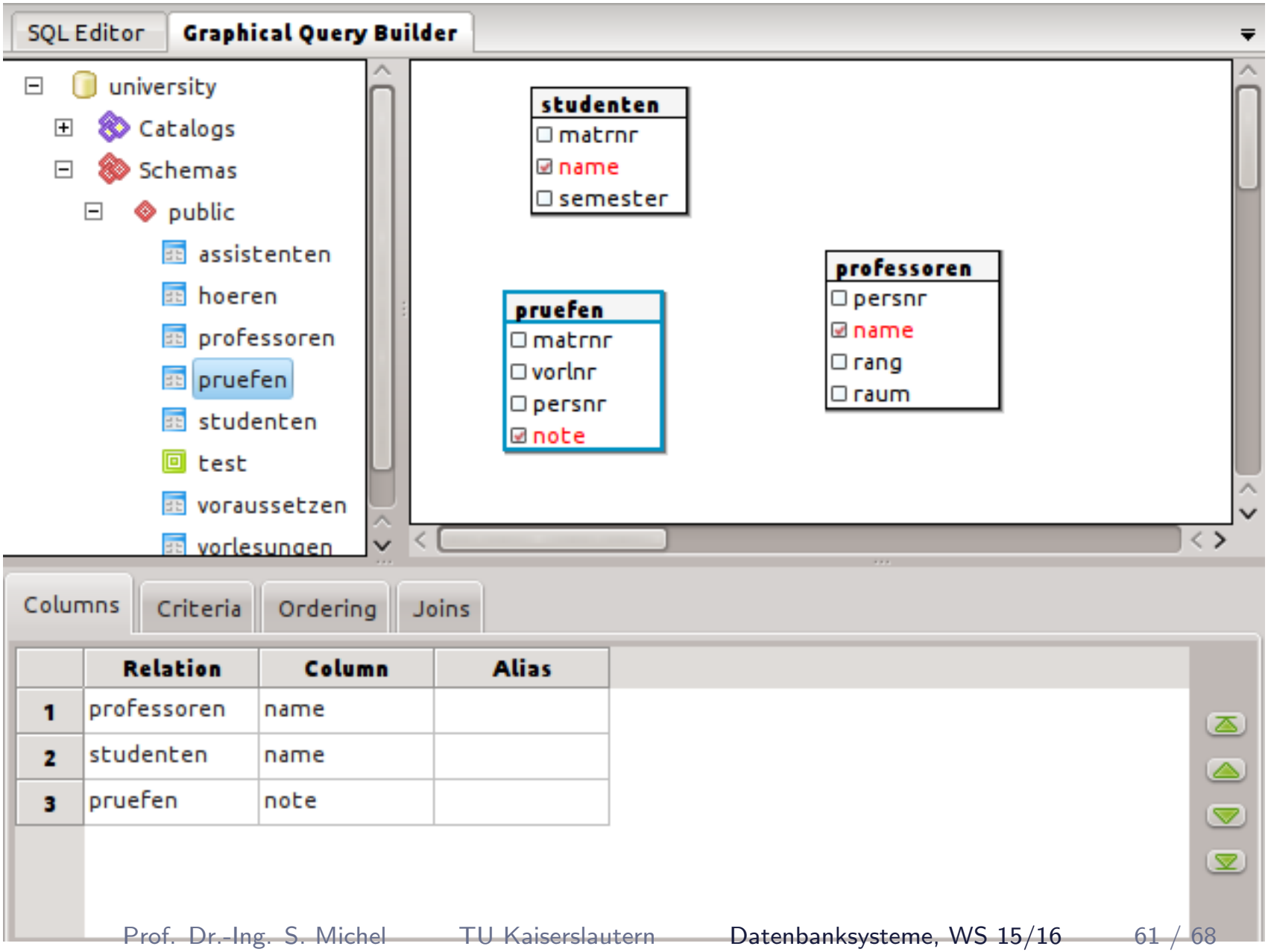
Features

- Query Builder
- Sehr anschauliche Darstellung von Ausführungplänen (explain plan)
- Komfortable Administration (Benutzer, Rechteverwaltung, ...)

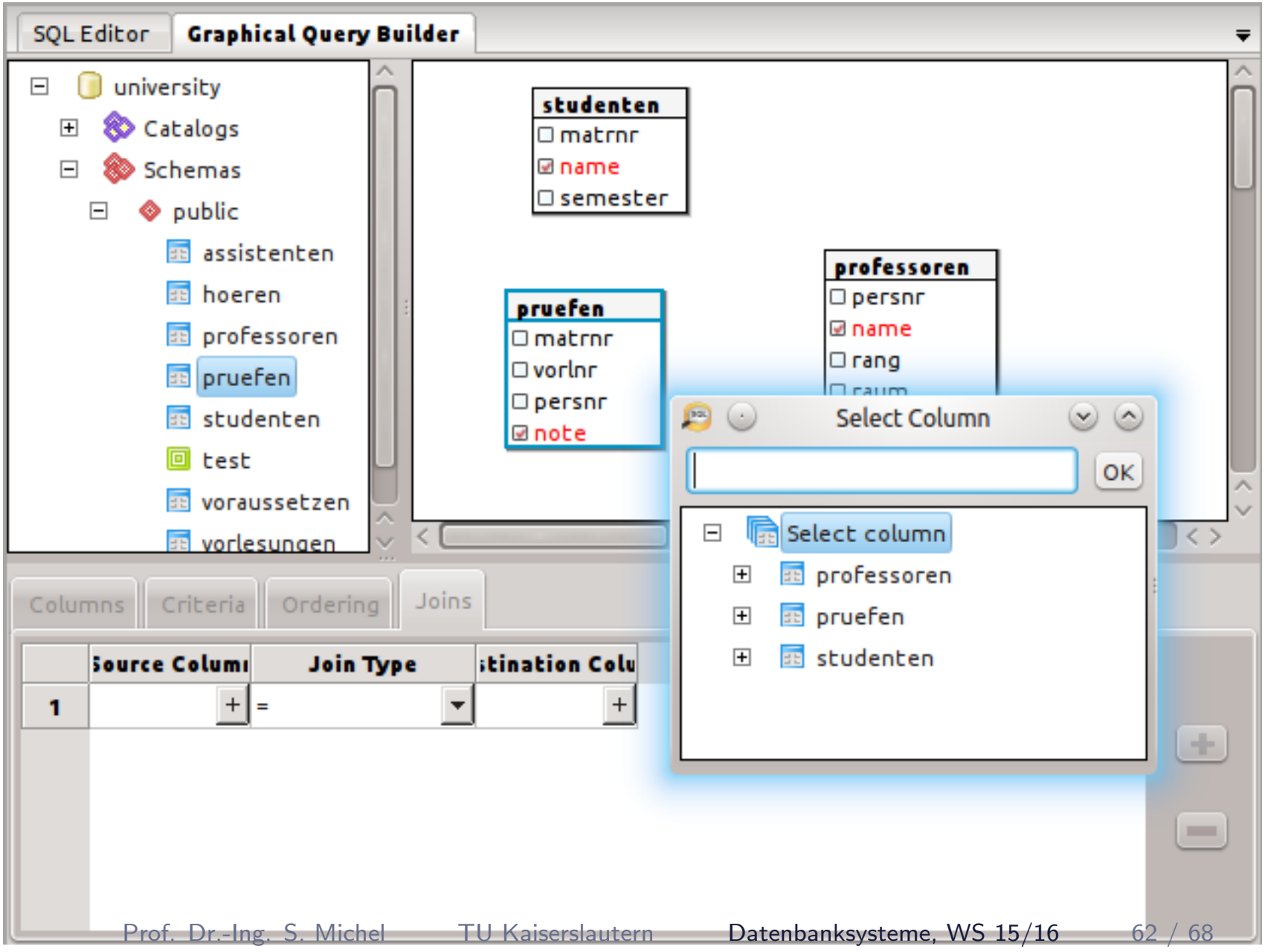
PGAdmin3 (2)


Query - university on postgres@localhost:5432 *
File Edit Query Favourites Macros View Help
university on
SQL Editor
Graphical Query Builder
Previous queries
Delete
Delete All
select *
from studenten NATURAL JOIN hoeren
where semester > 6
Scratch pad
Output pane
Data Output
Explain
Messages
History
matrnr integer
name character varying(30)
semester integer
vorlnr integer
1 26120 Fichte 10 5001
2 25403 Jonas 12 5022

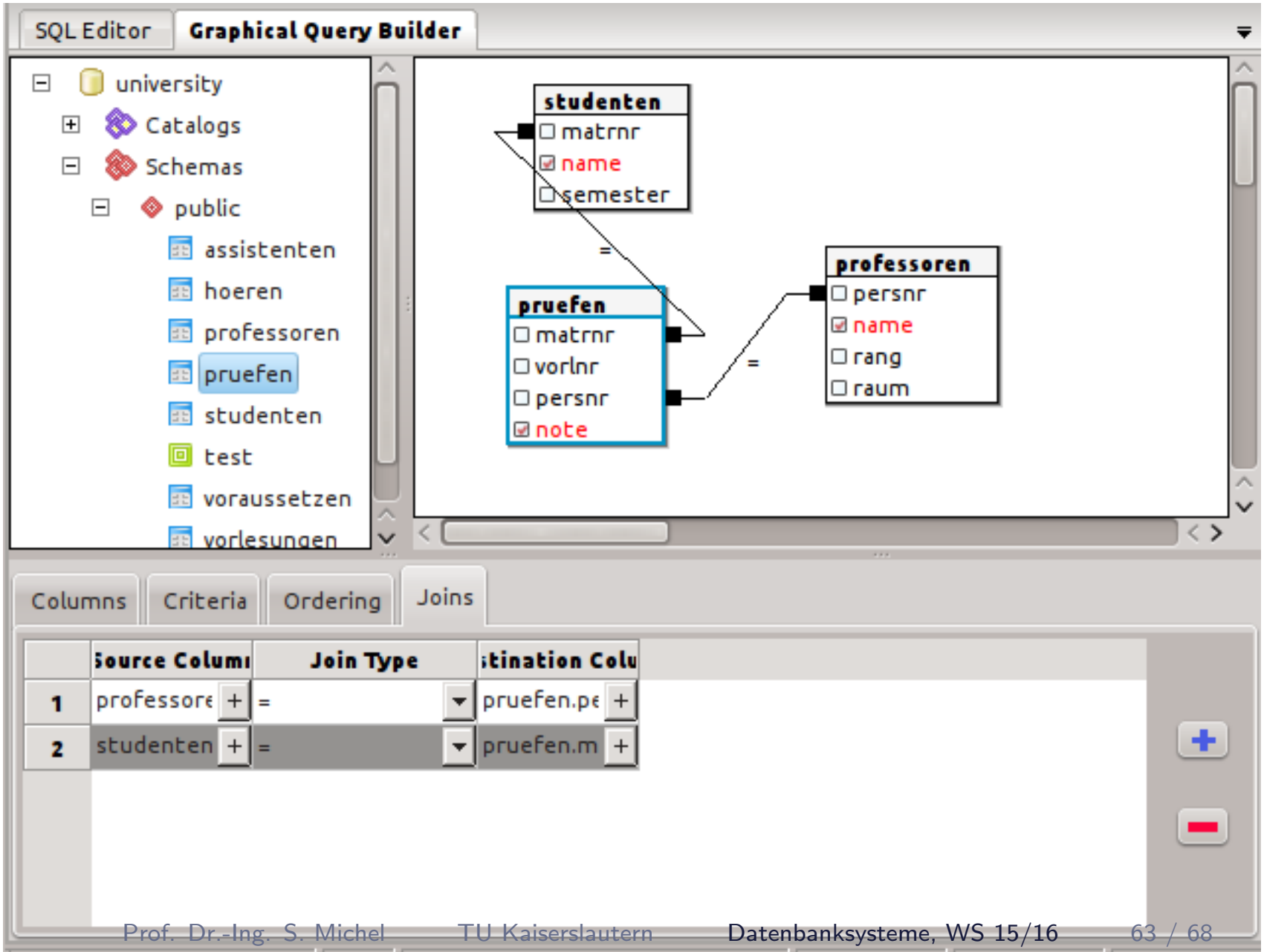
PGAdmin3: query builder (1)



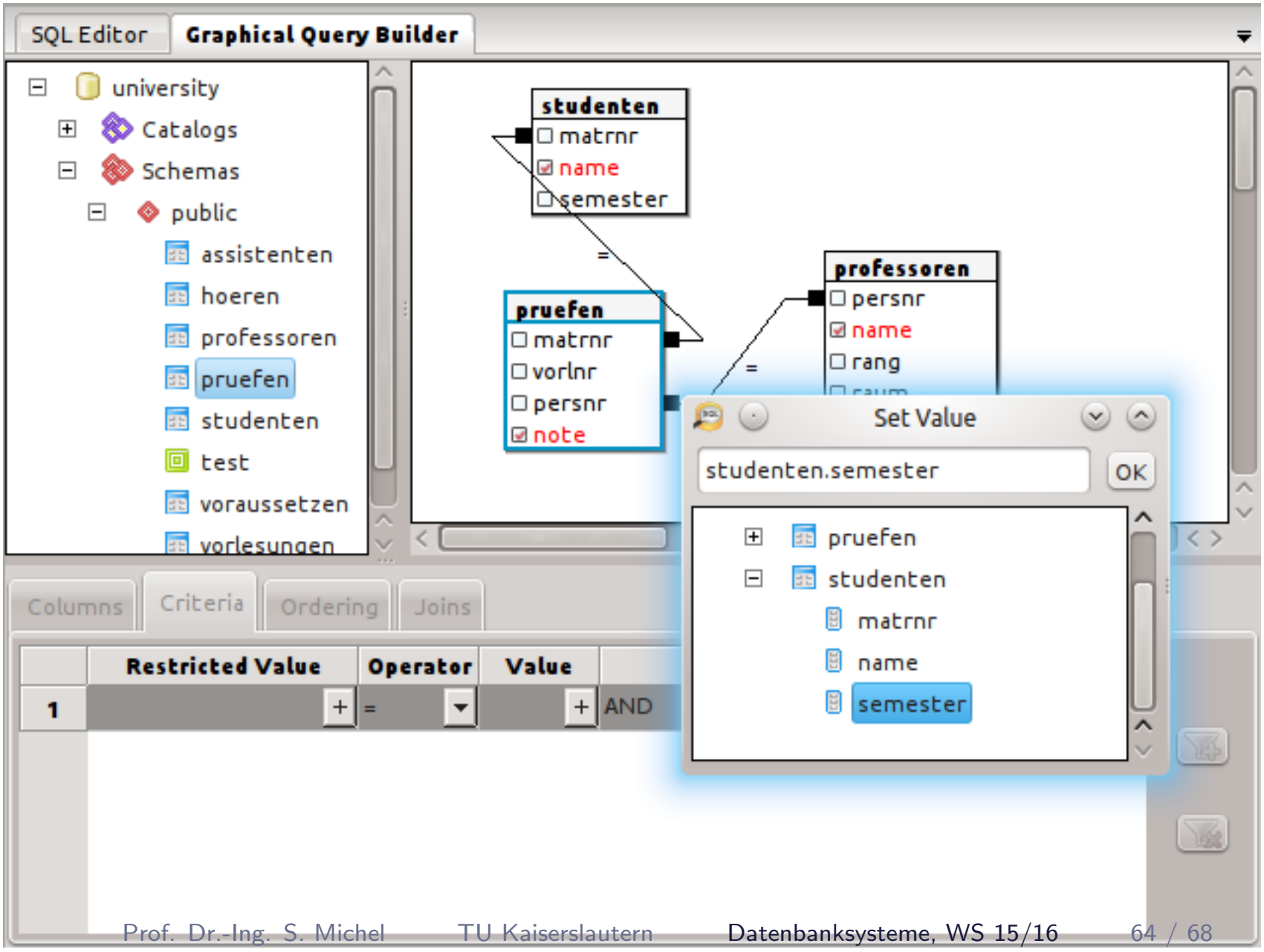
PGAdmin3: query builder (2)



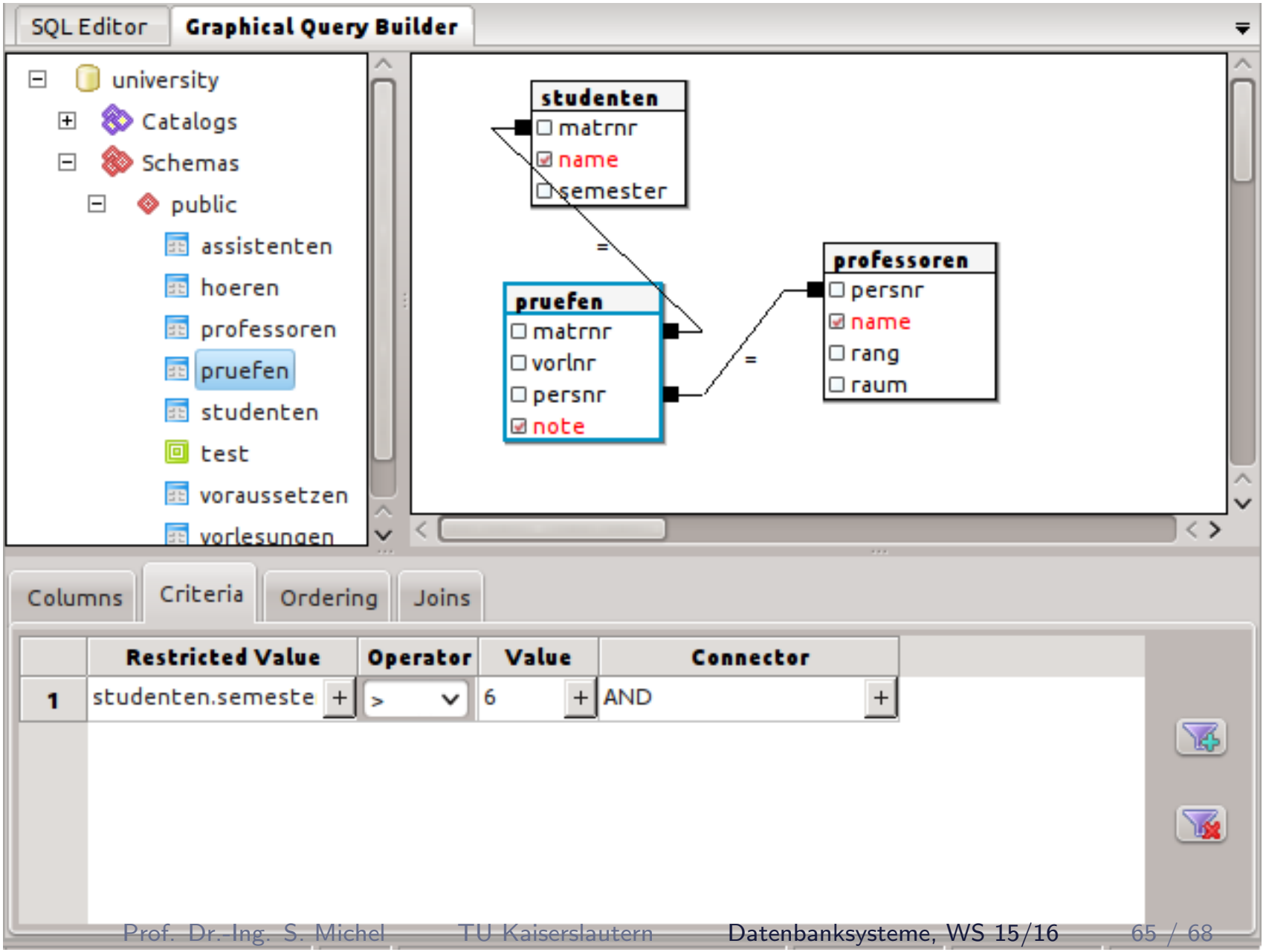
PGAdmin3: query builder (3)



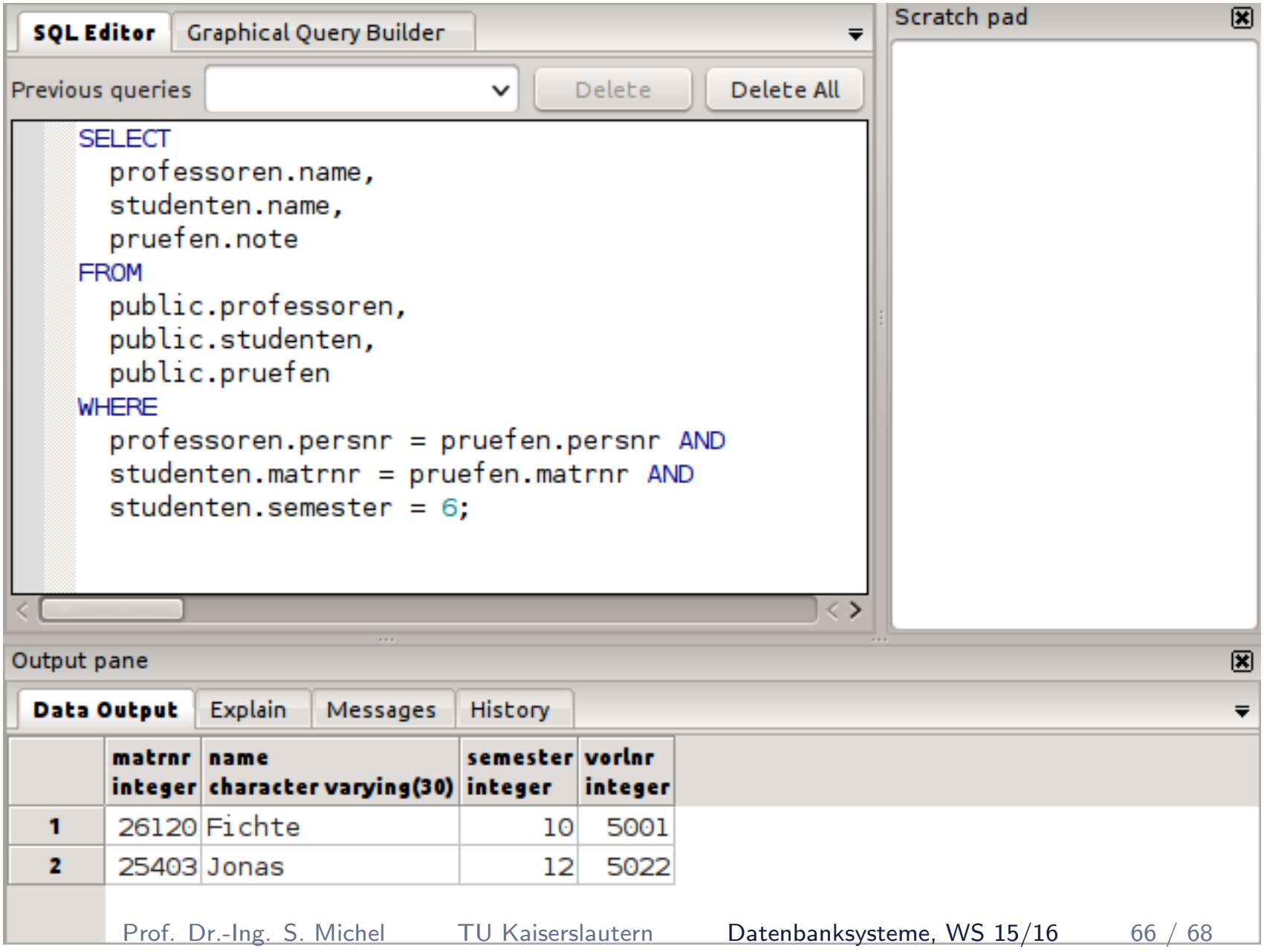
PGAdmin3: query builder (4)



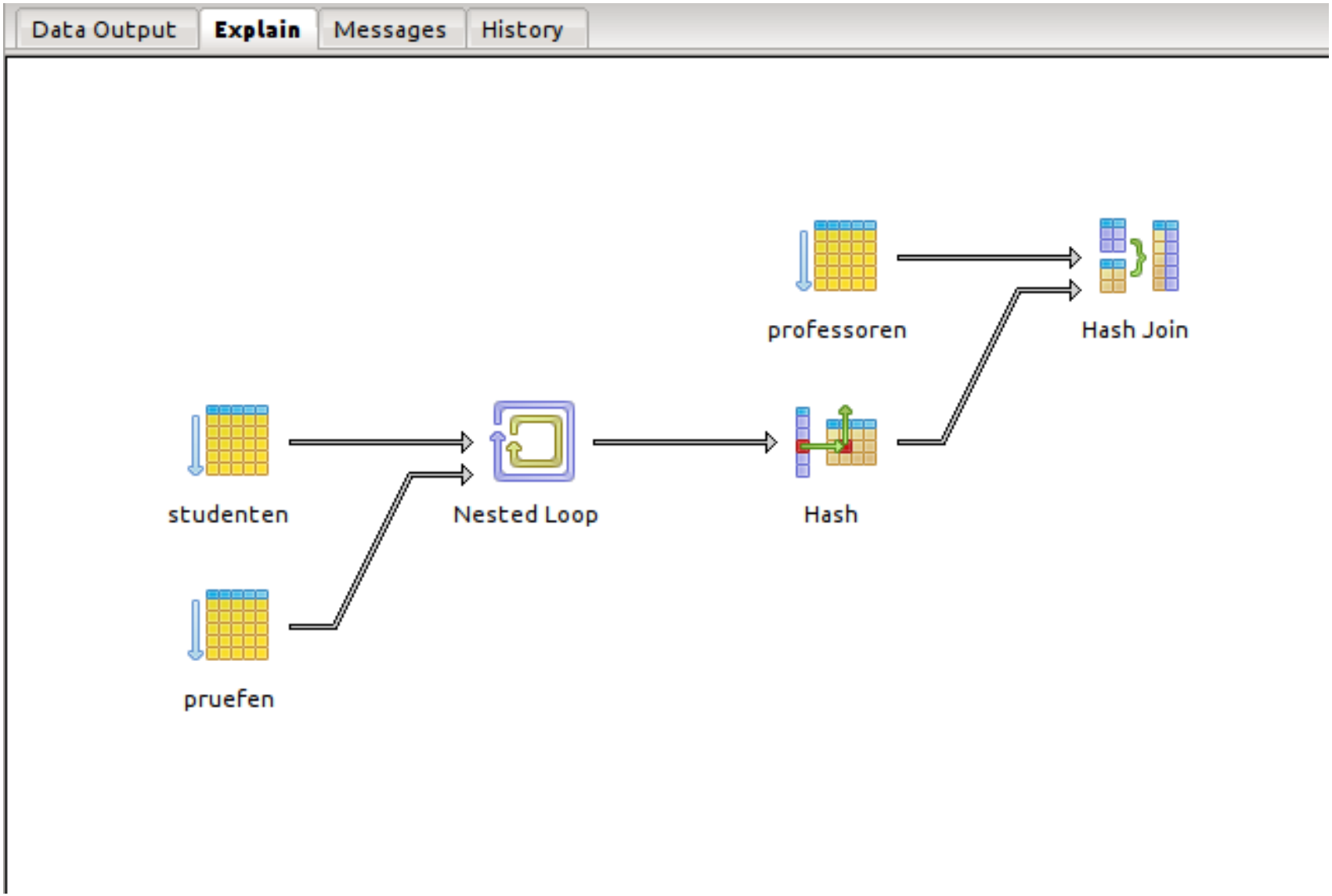
PGAdmin3: query builder (5)



PGAdmin3: query builder (6)



Postgresql/PGAdmin3: Explain Plan (1)



Postgresql/PGAdmin3: Explain Plan (2)

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Hash Join (cost=2.18..3.29 rows=1 width=168) (actual time=21.784..21.785 rows=1 loops=1)
2	Hash Cond: (professoren.persnr = pruefen.persnr)
3	-> Seq Scan on professoren (cost=0.00..1.07 rows=7 width=82) (actual time=11.201..11.203 rows=7 loops=1)
4	-> Hash (cost=2.17..2.17 rows=1 width=94) (actual time=10.563..10.563 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 1kB
6	-> Nested Loop (cost=0.00..2.17 rows=1 width=94) (actual time=10.556..10.559 rows=1 loops=1)
7	Join Filter: (studenten.matrnr = pruefen.matrnr)
8	-> Seq Scan on studenten (cost=0.00..1.10 rows=1 width=82) (actual time=0.009..0.011 rows=1 loops=1)
9	Filter: (semester = 6)
10	-> Seq Scan on pruefen (cost=0.00..1.03 rows=3 width=20) (actual time=10.533..10.535 rows=3 loops=1)
11	Total runtime: 21.841 ms