

Prolog als Datenbanksprache - Idee

student

Mnr	Vorname	Name	Semester
1000	'Anna'	'Arm'	'ti2'
1001	'Rita'	'Reich'	'ti2'
1002	'Peter'	'Reich'	'ti2'
1003	'Peter'	'Petersen'	'ti2'

←Tupel=Fakt
=Datensatz

kurs

Semester	Fach
'ti2'	'Mathe2'
'ti2'	'Physik2'

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 1

Prolog als Datenbanksprache - Relation

Prädikat
=Relation
=Prozedur

↓

student(1000 , 'Anna' , 'Arm' , 'ti2'). ←Tupel=Fakt
=Datensatz

student(1001 , 'Rita' , 'Reich' , 'ti2').

student(1002 , 'Peter' , 'Reich' , 'ti2').

student(1003 , 'Peter' , 'Petersen' , 'ti2').

kurs('ti2' , 'Mathe2').

kurs('ti2' , 'Physik2').

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 2

Prolog als Datenbanksprache - Syntax

Prädikat
=Relation
=Prozedur

Konstanten

Integer Atom } klein oder in ''

student(1000 , 'Anna' , 'Arm' , 'ti2'). ←Tupel=Fakt
=Datensatz

student(1001 , 'Rita' , 'Reich' , 'ti2').

student(1002 , 'Peter' , 'Reich' , 'ti2').

student(1003 , 'Peter' , 'Petersen' , 'ti2').

Keine Lücke !

Goal = Subquery

?- student(M, V, _, _).

Variablen anonyme Variablen

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 3

Antworten durch Variablenbindung

Konstanten

Integer Atom

student(1000 , 'Anna' , 'Arm' , 'ti2').

Unifikation
scheitert

?- student(M , 'Peter' , _ , _).

Unifikation
bindet:
M=1002
...

student(1002 , 'Peter' , 'Reich' , 'ti2').

student(1003 , 'Peter' , 'Petersen' , 'ti2').

Anonyme Variablen _ und _ können verschieden gebunden werden

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 4

Select-Project-Join-Queries

Query: In welchem Semester S ist Anna Arm,
und welche Fächer F muss man in Semester S hören?

Goal = Subquery

Subquery

?- student(_, 'Anna' , 'Arm' , S) , kurs(S , F) .

Projektion, Selektion Join

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 5

Join und kartesisches Produkt

Query: Wer (ist in welchem Semester S und) muss (deshalb)
welche Fächer F hören?

Goal = Subquery

Subquery

?- student(M, V, N, S) , kurs(S, F) .

Join

Query: Wer könnte welche Fächer F hören?

?- student(M, V, N, S) , kurs(S2, F) .

Kartesisches Produkt

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 6

Prologeregeln - Syntax

Regel:

Head = Regelkopf = View
Goal = Subquery
Subquery

$\underbrace{\text{pflichten}(M, V, N, S, F)}_{\text{Head}} \text{ :- } \underbrace{\text{student}(M, V, N, S)}_{\text{Goal}} , \underbrace{\text{kurs}(S, F)}_{\text{Subquery}} .$

?- pflichten(1000, _, _, _, F) .

Anfrage = Query

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 7

Deklarative Semantik

Prolog:
pflichten(M, V, N, S, F) :- student(M, V, N, S), kurs(S, F) .

Prädikatenlogik
pflichten(M, V, N, S, F) \Leftarrow student(M, V, N, S) \wedge kurs(S, F) .
falls und

Relationale Algebra
pflichten(M, V, N, S, F) := student $\bowtie_{4=1}$ kurs

SQL
Create view Pflichten as select * from student S, kurs K where S . S = K . S

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 8

Select-Project-Join-Queries (Übung)

1. Welche Fächer sind im 'AM' geplant?
2. Welche Semester hören eine Veranstaltung im 'AM'?
3. Welche Semester hören Mo14-16 eine Veranstaltung im 'AM'?
4. In welchen Räumen muss Anna Arm Fächer hören?
5. Welche weiteren Queries können damit beantwortet werden?

Gegeben:

```
% student(Mnr, Vorname, Name, Semester) .
student(1000, 'Anna', 'Arm', 'ti2' ) .
...
% kurs(Semester, Fach) .
kurs('ti2', 'Mathe2' ) .
...
% plan(Fach, Raum, Zeit) .
plan('Mathe2', 'AM', 'Mo14-16' ) .
...
```

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 9

Prozedurale Semantik: Datenfluss

Regel:

Head = Regelkopf = View
Goal = Subquery
Subquery

$\underbrace{\text{pflichten}(M, V, N, S, F)}_{\text{Head}} \text{ :- } \underbrace{\text{student}(M, V, N, S)}_{\text{Goal}} , \underbrace{\text{kurs}(S, F)}_{\text{Subquery}} .$

?- pflichten(1000, _, _, _, F) .

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 10

Schnitt, Vereinigung, Differenz

Gegeben sind: grundkurs(S,F) und kompaktkurs(S,F)

Schnitt: kompakte Grundkurse:
?- grundkurs(S,F), kompaktkurs(S,F).

Als Regel:
kompakterGrundkurs(S,F) :- grundkurs(S,F), kompaktkurs(S,F).

Vereinigung: Grund- oder Kompaktkurse
grundOderKompaktkurs(S,F) :- grundkurs(S,F) .
grundOderKompaktkurs(S,F) :- kompaktkurs(S,F).

Differenz: Grundkurse ohne Kompaktkurse
gOhneKkurs(S,F) :- grundkurs(S,F), \oplus kompaktkurs(S,F).

↑
Negationsoperator (NOT)

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 11

Negation as failure

?- \+ student(1000, 'Anna', 'Arm', _).
liefert no, weil
?- student(1000, 'Anna', 'Arm', _).
yes liefert.

?- \+ student(123, 'Anna', 'Arm', _).
liefert yes, weil
?- student(123, 'Anna', 'Arm', _).
Fehlschlägt (no liefert). → Negation as failure.

?- \+ student(M, V, N, S).
liefert no, weil
?- student(M, V, N, S).
Ergebnisse liefert.

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Bötcher - S. Prolog / 12

Sichere Anfragen(1)

Ist Anna Arm mit *keiner Mnr* als Studentin eingetragen – und was ist Ihre Mnr?

?- \+ student(Mnr, 'Anna', 'Arm', _).

ist unsicher, denn Mnr ist ungebunden. Was ist Mnr ?

Ist Anna Arm nicht Studentin ?

?- \+ student(_, 'Anna', 'Arm', _).

ist sicher, weil keine ungebundene Variable vorkommt.

Unsichere Anfragen sind in den relationalen Kalkülen verboten ! In Prolog gibt es ungebundene Variablen (_x123) als Antworten

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 13

Sichere Anfragen(2)

Welche Kurse muss Anna Arm nicht hören?

?- kurs(S, F), \+ student(_, 'Anna', 'Arm', S).

ist sicher, weil die Variable S im negierten Prädikat (\+student(...)) vorher in einem nicht-negierten Prädikat gebunden wurde.

Ist Anna Arm in *keinem Semester* Studentin – und welche Kurse müsste man in *diesem Semester* hören?

?- \+ student(_, 'Anna', 'Arm', S), kurs(S, F).

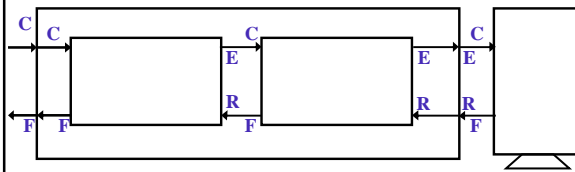
Ist nicht sicher, weil die Variable S im negierten Prädikat (\+student(...)) noch nicht gebunden ist.

In Prolog: Reihenfolge beachten ! (prozedurale Semantik)

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 14

Prozedurale Semantik : 4 Port-Modell

pflichten(M, V, N, S, F) :- student(M, V, N, S), kurs(S, F) .
?- pflichten(M, V, N, S, F).

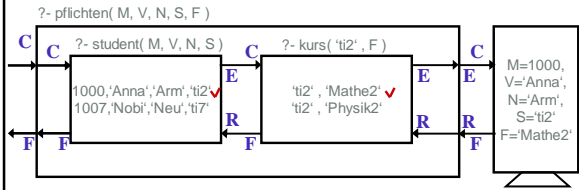


C=Call E=Exit R=Redo F=Fail

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 15

Prozedurale Semantik : 4 Port-Modell

pflichten(M, V, N, S, F) :- student(M, V, N, S), kurs(S, F) .
?- pflichten(M, V, N, S, F).

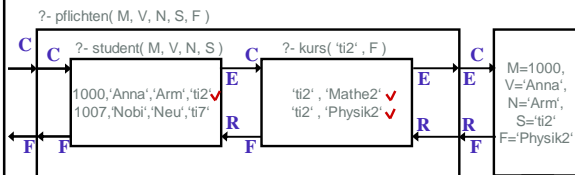


C=Call E=Exit R=Redo F=Fail

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 16

Prozedurale Semantik : 4 Port-Modell

pflichten(M, V, N, S, F) :- student(M, V, N, S), kurs(S, F) .
?- pflichten(M, V, N, S, F).

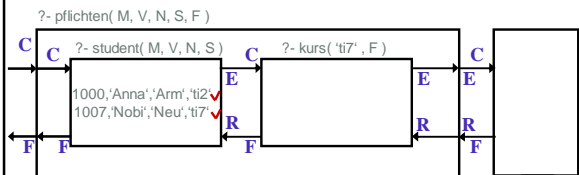


C=Call E=Exit R=Redo F=Fail

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 17

Prozedurale Semantik : 4 Port-Modell

pflichten(M, V, N, S, F) :- student(M, V, N, S), kurs(S, F) .
?- pflichten(M, V, N, S, F).

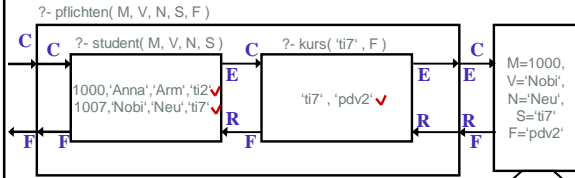


C=Call E=Exit R=Redo F=Fail

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 18

Prozedurale Semantik : 4 Port-Modell

pflichten(M, V, N, S, F) :- **student**(M, V, N, S) , **kurs**(S, F) .
?- **pflichten**(M, V, N, S, F).

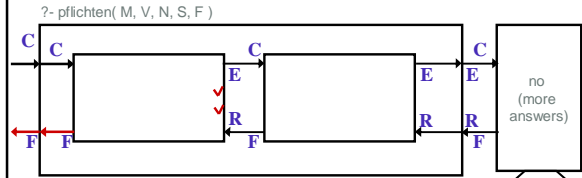


C=Call E=Exit R=Redo F=Fail

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 19

Prozedurale Semantik : 4 Port-Modell

pflichten(M, V, N, S, F) :- **student**(M, V, N, S) , **kurs**(S, F) .
?- **pflichten**(M, V, N, S, F).



C=Call E=Exit R=Redo F=Fail

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 20

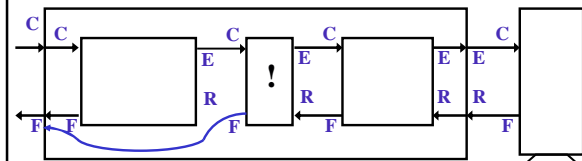
pflichten(M, V, N, S, F) :- **student**(M, V, N, S) , **kurs**(S, F) .

```
void pflichten( M, V, N, S, F )
{ // call-Port von Student
  AS = student . liesAlle( M, V, N, S ) ;
  while ( student(M,V,N,S) = AS. next() )
  { // exit-Port von Student und call-Port von Kurs
    AK = kurs . liesAlle( S, F ) ;
    while ( kurs(S,F) = AK. next() )
    { // exit-Port von Kurs und call-Port von Ausgabe
      Ausgabe( M, V, N, S, F ) ;
      // fail-Port von Ausgabe und redo-Port von Kurs
    }
    // fail-Port von Kurs und redo-Port von Student
  }
  // fail-Port von Student
}
```

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 21

Cut im 4 Port-Modell

pflichten(M, V, N, S, F) :- **student**(M, V, N, S) , ! , **kurs**(S, F) .
?- **pflichten**(M, V, N, S, F).



Cut verlässt auf dem Rückweg die Prozedur-Aufruf-Box (=return)

C=Call E=Exit R=Redo F=Fail

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 22

Jede Lösung nur einmal sehen

Beispiel: Welche Studenten hören mehrere Kurse?

Test-Implementierung :

hörtMehrereKurse(M) :-

hört(M, K1) , hört(M, K2) , \+ K1=K2 , ! .

0 oder 1 Antwort pro M wegen Cut am Ende

Generate-And-Test-Regel :

studentHörtMehrereKurse(M, V, N, S) :-

student(M, V, N, S) , hörtMehrereKurse(M) .

Generator Test

(generiert jeden Student genau 1x) (wählt oder wählt nicht)

Anfrage:

?- **studentHörtMehrereKurse**(M, V, N, S) .

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 23

$\forall x \in R(p(x))$ ersetzen durch $\text{not } \exists x \in R(\text{not } p(x))$

Beispiel: Welche Studenten hören *alle* für 'ti2' angebotenen Kurse ?

$\{ (M,V,N,S) \in \text{Student} \mid \forall ('ti2',F) \in \text{Kurs} (\text{hört}(M,F)) \} \Leftrightarrow$
 $\{ (M,V,N,S) \in \text{Student} \mid \text{not } \exists ('ti2',F) \in \text{Kurs} (\text{not } \text{hört}(M,F)) \}$

Generate-And-Test-Regel :

studentHörtAlleti2Kurse(M, V, N, S) :-

student(M, V, N, S) , \+ hörtMindestens1ti2KursNicht(M) .

Generator Test

(generiert jeden Student genau 1x) (wählt oder wählt nicht)

Test-Implementierung :

hörtMindestens1ti2KursNicht(M) :-

kurs('ti2', K) , \+ hört(M, K) , ! .

Anfrage:

?- **studentHörtAlleti2Kurse**(M, V, N, S) .

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 24

All-quantifizierte Queries (Übung)

1. Welche Studenten hören mindestens einen Kurs nicht?
2. Welche Studenten hören alle Kurse?
3. Welche Kurse werden von mindestens einem Student nicht gehört?
4. Welche 'ti2'-Kurse werden von allen Studenten gehört?
5. Welche Kurse werden von allen 'ti2'-Studenten gehört?
6. Welche weiteren Queries können damit beantwortet werden?

Gegeben: `% student(Mnr, Vorname, Name, Semester) .`
`student(1000 , 'Anna' , 'Arm' , 'ti2') .`
`...`
`% hört(Mnr , Fach) .`
`hört(1000 , 'Mathe2') .`
`...`
`% kurs(Semester, Fach) .`
`kurs('ti2' , 'Mathe2') .`
`...`

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 25

Maximum ersetzen durch "≥alle"

Beispiel: Welcher Student hat die größte Matrikelnummer?

$\{ (M, V, N, S) \in \text{Student} \mid M = \max(\{ M2 \mid (M2, V2, N2, S2) \in \text{Student} \}) \}$ \Leftrightarrow
 $\{ (M, V, N, S) \in \text{Student} \mid \forall (M2, V2, N2, S2) \in \text{Student} (M \geq M2) \}$ \Leftrightarrow
 $\{ (M, V, N, S) \in \text{Student} \mid \text{not } \exists (M2, V2, N2, S2) \in \text{Student} (M < M2) \}$

Generate-And-Test-Regel :

`studentHatGrößteMatrikelnummer(M, V, N, S) :-`
`student(M, V, N, S) , !+ jemandHatGrößereMatrikelnummerAls(M) .`
Generator Test
 (generiert jeden Student genau 1x) (wählt oder wählt nicht)

Test-Implementierung :

`jemandHatGrößereMatrikelnummerAls(M) :-`
`student(M2, _, _, _) , M2 > M .`

Anfrage:

?- studentHatGrößteMatrikelnummer(M, V, N, S) .

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 26

Das Prolog-System

1. Fenster (Editor) für Datenbank

Fakten
und Regeln !!

Editor

```
student( 1000, 'Anna', 'Arm', 'ti2' ) .
...
pflichten( M, V, N, S, F ) :- ... .
aufgabe1( X ) :- ... .
aufgabe2( Y, Z ) :- ... .
```

2. Fenster (Prolog) für Compiler-Anweisungen und Queries

```
DOS_Pfad> bp_w95c uebung1.pl
?- [uebung1].
...
?- aufgabe1( X ).
...
?- aufgabe2( Y, Z ).
```

Datenbank-Grundlagen - SS 2005 - Prof. Dr. Böttcher - S. Prolog / 27