

Datenbanken:

Tutorium 6

Marvin Jahn

27.11.2019

Wildcards in Strings

“Bestimme alle Studenten, deren Name mit einem 'M' beginnt.”

Wildcards in Strings

“Bestimme alle Studenten, deren Name mit einem 'M' beginnt.”

```
select *  
from Studenten s  
where s.Name like 'M%'
```

Wildcards in Strings

“Bestimme alle Studenten, deren Name mit einem 'M' beginnt.”

```
select *  
from Studenten s  
where s.Name like 'M%'
```

“Bestimme alle Professoren, deren Name als zweiten Buchstaben ein 'o' enthält.”

Wildcards in Strings

“Bestimme alle Studenten, deren Name mit einem 'M' beginnt.”

```
select *  
from Studenten s  
where s.Name like 'M%'
```

“Bestimme alle Professoren, deren Name als zweiten Buchstaben ein 'o' enthält.”

```
select *  
from Professoren p  
where p.Name like '_o%'
```

Aggregatfunktionen

Aggregatfunktionen reduzieren alle Werte einer Spalte zu einem *einzigem* Wert.

In SQL gibt es u.a. folgende Aggregatfunktionen:

- Zählen: *count()*
- Aufsummieren: *sum()*
- Durchschnitt bilden: *avg()*
- Maximum finden: *max()*
- Minimum finden: *min()*

Gruppieren mit *having*

- häufig möchte man erst gruppieren und dann eine Aggregatfunktion (z.B. *count*) aufrufen

Gruppieren mit *having*

- häufig möchte man erst gruppieren und dann eine Aggregatfunktion (z.B. *count*) aufrufen
- um nach dem Gruppieren noch weiter zu filtern, benutzt man eine *having*-Klausel

Beispiel

“Finde den Namen aller Studenten, die genau zwei Vorlesungen hören.”

Beispiel

“Finde den Namen aller Studenten, die genau zwei Vorlesungen hören.”

```
select distinct s.Name  
from Studenten s, hoeren h  
where s.MatrNr = h.MatrNr  
group by s.MatrNr, s.Name  
having count(*) = 2
```

Vorgehen bei komplexeren Anfragen

Vorgehen bei komplexeren Anfragen

- Zerlegen in Teilprobleme

Vorgehen bei komplexeren Anfragen

- Zerlegen in Teilprobleme
- Lösen der Teilprobleme

Vorgehen bei komplexeren Anfragen

- Zerlegen in Teilprobleme
- Lösen der Teilprobleme
- Zusammenfügen der Lösung:

Vorgehen bei komplexeren Anfragen

- Zerlegen in Teilprobleme
- Lösen der Teilprobleme
- Zusammenfügen der Lösung:
 - ▶ temporäre Relationen mit *with* erstellen

Vorgehen bei komplexeren Anfragen

- Zerlegen in Teilprobleme
- Lösen der Teilprobleme
- Zusammenfügen der Lösung:
 - ▶ temporäre Relationen mit *with* erstellen
 - ▶ Unteranfragen mit *select* (meistens in der *where*-Klausel)

Vorgehen bei komplexeren Anfragen

- Zerlegen in Teilprobleme
- Lösen der Teilprobleme
- Zusammenfügen der Lösung:
 - ▶ temporäre Relationen mit *with* erstellen
 - ▶ Unteranfragen mit *select* (meistens in der *where*-Klausel)
 - ▶ mehrere Anfragen mit *intersect*, *union*, *union all* oder *except* verbinden

Beispiel 1

“Finde den Namen aller Studenten und Professoren.”

```
( select Name from Studenten )  
union  
( select Name from Professoren )
```

Beispiel 2

“Finde alle Professoren für die Assistenten mit verschiedenen Fachgebieten arbeiten.”

Beispiel 2

“Finde alle Professoren für die Assistenten mit verschiedenen Fachgebieten arbeiten.”

```
select distinct p.Name
from Professoren p, Assistenten a
where a.Boss = p.PersNr and exists
    (select *
     from Assistenten b
     where b.Boss = p.PersNr
        and a.Fachgebiet != b.Fachgebiet)
```

Beispiel 2

“Finde alle Professoren für die Assistenten mit verschiedenen Fachgebieten arbeiten.”

```
select distinct p.Name
from Professoren p, Assistenten a
where a.Boss = p.PersNr and exists
    (select *
     from Assistenten b
     where b.Boss = p.PersNr
        and a.Fachgebiet != b.Fachgebiet)
```

Dies ist eine **korellierte** Unteranfrage, da die Unteranfrage die Relation *p* referenziert.

Beispiel 3

“Generiere eine Tabelle, die angibt, wie beliebt die Professoren sind. Beliebtheit ist definiert als der Prozentsatz der Studenten, die mind. eine Vorlesung bei diesem Professor hören.”

```
with kenntSich(ProfPersNr,StudMatrNr) as
    (select distinct v.gelesenVon,h.MatrNr
    from hoeren h, Vorlesungen v
    where h.VorINr = v.VorINr),
kenntAnzahl (ProfPersNr, AnzStudenten) as
    (select ProfPersNr, count(*)
    from kenntSich
    group by ProfPersNr),
wieviele (GesamtAnz) as
    (select count(*) from Studenten)
```

```

with kenntSich(ProfPersNr,StudMatrNr) as
    (select distinct v.gelesenVon,h.MatrNr
    from hoeren h, Vorlesungen v
    where h.VorINr = v.VorINr),
kenntAnzahl (ProfPersNr, AnzStudenten) as
    (select ProfPersNr, count(*)
    from kenntSich
    group by ProfPersNr),
wieviele (GesamtAnz) as
    (select count(*) from Studenten)

select Name,
        AnzStudenten*1.00/GesamtAnz as Beliebtheit
from Professoren, kenntAnzahl, wieviele
where PersNr = ProfPersNr
order by Beliebtheit desc

```


Achtung: Im Ergebnis sind momentan nur die Professoren, die mind. eine Vorlesung lesen, die von mind. einem Studenten gehört wird.

Deshalb ersetzen wir den vorherigen *select*-Teil durch das Folgende:

Achtung: Im Ergebnis sind momentan nur die Professoren, die mind. eine Vorlesung lesen, die von mind. einem Studenten gehört wird.

Deshalb ersetzen wir den vorherigen *select*-Teil durch das Folgende:

```
(select Name,  
        AnzStudenten*1.00/GesamtAnz as Beliebtheit  
from Professoren, kenntAnzahl, wieviele  
where PersNr = ProfPersNr)  
union  
(select Name, 0  
from Professoren p  
where not exists  
(select * from kenntAnzahl  
  where PersNr = ProfPersNr))  
order by Beliebtheit desc
```

Einfügen von Daten mit *insert into*

“Füge einen Studenten namens 'Tom' hinzu, der Ethik hört.”

Einfügen von Daten mit *insert into*

“Füge einen Studenten namens 'Tom' hinzu, der Ethik hört.”

```
insert into Studenten  
values(2400, 'Tom', 1);  
insert into hoeren  
values(2400,5041)
```

Einfügen von Daten mit *insert into*

“Füge einen Studenten namens 'Tom' hinzu, der Ethik hört.”

```
insert into Studenten  
values(2400, 'Tom', 1);  
insert into hoeren  
values(2400,5041)
```

“Füge einen Studenten namens 'Tom' hinzu, der alle Vorlesungen hört.”

Einfügen von Daten mit *insert into*

“Füge einen Studenten namens 'Tom' hinzu, der Ethik hört.”

```
insert into Studenten  
values (2400, 'Tom', 1);  
insert into hoeren  
values (2400, 5041)
```

“Füge einen Studenten namens 'Tom' hinzu, der alle Vorlesungen hört.”

```
insert into Studenten (MatrNr, Name)  
values (2400, 'Tom');  
insert into hoeren  
select 2400, v.VorlNr  
from Vorlesungen v
```

Updaten von Daten mit *update*

“Update die Datenbank, sodass Sokrates in Raum 1 arbeitet.”

Updaten von Daten mit *update*

“Update die Datenbank, sodass Sokrates in Raum 1 arbeitet.”

```
update Professoren  
set Raum = 1  
where Name = 'Sokrates'
```


Updaten von Daten mit *update*

“Update die Datenbank, sodass Sokrates in Raum 1 arbeitet.”

```
update Professoren  
set Raum = 1  
where Name = 'Sokrates'
```

“Stufe alle Studenten um ein Semester nach oben.”

Updaten von Daten mit *update*

“Update die Datenbank, sodass Sokrates in Raum 1 arbeitet.”

```
update Professoren  
set Raum = 1  
where Name = 'Sokrates'
```

“Stufe alle Studenten um ein Semester nach oben.”

```
update Studenten  
set Semester = Semester + 1
```

Löschen von Daten oder Relationen

“Lösche Sokrates aus der Datenbank.”

Löschen von Daten oder Relationen

“Lösche Sokrates aus der Datenbank.”

```
delete from Professoren  
where Name = 'Sokrates'
```

Löschen von Daten oder Relationen

“Lösche Sokrates aus der Datenbank.”

```
delete from Professoren  
where Name = 'Sokrates'
```

“Lösche die Tabelle 'Studenten'.”

```
drop table Studenten
```

Erstellen einer neuen Relation

“Erstelle die Relation 'Professoren'.”

Erstellen einer neuen Relation

“Erstelle die Relation 'Professoren’.”

```
create table Professoren
(PersNr integer not null primary key ,
Name varchar (30) not null ,
Rang character (2) ,
Raum integer ,
check (Raum > 0 and Raum < 99999))
```

Erstellen einer neuen Relation

“Erstelle die Relation 'Vorlesungen'.”

Erstellen einer neuen Relation

“Erstelle die Relation ‘Vorlesungen’.”

```
create table Vorlesungen
(
  VorlNr integer not null primary key ,
  Titel varchar (30) not null ,
  SWS integer ,
  gelesenVon integer ,
  foreign key (gelesenVon)
  references Professoren(PersNr)
  on delete set null
  on update cascade)
```

Erstellen einer neuen Relation

“Erstelle die Relation ‘Vorlesungen’.”

```
create table Vorlesungen
(
  VorlNr integer not null primary key ,
  Titel varchar (30) not null ,
  SWS integer ,
  gelesenVon integer ,
  foreign key (gelesenVon)
  references Professoren(PersNr)
  on delete set null
  on update cascade)
```

Die *on delete*-Klausel entscheidet, was passiert, wenn ein Tupel in der referenzierten Relation (hier: *Professoren*) gelöscht wird. Analog funktioniert *on update*.