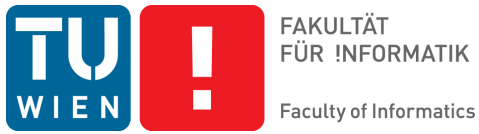
Data Modelling/Data Base Systems

VU 184.685/VU 184.686, WS 2020

Relational Query Languages – SQL

Anela Lolić

Institute of Logic and Computation, TU Wien



Acknowledgements

The slides are based on the slides (in German) of Sebastian Skritek.

The content is based on Chapter 4 of
(Kemper, Eickler: Datenbanksysteme – Eine Einführung).
(Chapters 4.1 – 4.9, 4.11 – 4.15, 4.17)

For related literature in English see Chapter 5 of
(Ramakrishnan, Gehrke: Database Management Systems).

Overview

1. SQL then and now
2. Data Query Language
3. Data Definition Language
4. Data Manipulation Language

SQL then and now

SQL was developed based on relational algebra and relational calculus

declarative language

processing translation of the query through parser in a relational algebra and optimization through query optimizer of the DBMS (VU "Advanced Database Systems")

relations are represented by tables

SQL offers a standardized

- data definition languages (DDL)
- data manipulation language (DML)
- query language

SQL then and now

SQL 99: extended SQL 92 with object relational constructs, recursive queries and trigger

SQL 2003: offers extended support for nested tables, merge operations and XML based features

SQL 2006: bridge to XML, XQuery

SQL 2008, 2011: 2011 introduces temporal data

SQL 2016: a.o. JSON, current standard

System R (IBM): first DBMS prototype, language: Structured English Query Language $\Rightarrow$ SQL

DBMS: Oracle (Oracle Corporation), Informix (Informix), SQL-Server (Microsoft), DB2 (IBM), PostgreSQL, MySQL

Query Language

1. simple SQL queries
2. queries over several relations
3. set operations
4. aggregate functions
5. aggregate and grouping
6. nested queries
7. existentially quantified queries
8. universally quantified queries
9. null values
10. special language construct

Simple SQL Queries

Example

```
select *
from professor;
```

Simple SQL Queries

Example

```
select *
from professor;
```

professor

persNr	name	rank	room
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

Simple SQL Queries

Example

```
select name, room
from professor;
```

$$\pi_{\text{name,room}}(\text{professor})$$

Simple SQL Queries

Example

```
select name, room
from professor;
```

$\pi_{\text{name,room}}(\text{professor})$

professor	
name	room
Sokrates	226
Russel	232
Kopernikus	310
Popper	52
Augustinus	309
Curie	36
Kant	7

Simple SQL Queries – Duplicates

Example (eliminating duplicates)

all ranks for professors (rel. algebra)

$$\pi_{\text{rank}}(\text{professor})$$

Simple SQL Queries – Duplicates

Example (eliminating duplicates)

all ranks for professors (rel. algebra)

$$\pi_{\text{rank}}(\text{professor})$$

result
rank
C4
C3

Simple SQL Queries – Duplicates

Example (eliminating duplicates)

all ranks for professors (rel. algebra)

$$\pi_{\text{rank}}(\text{professor})$$

all ranks for professors (SQL)

```
select rank
from professor;
```

Simple SQL Queries – Duplicates

Example (eliminating duplicates)

all ranks for professors (rel. algebra)

$$\pi_{\text{rank}}(\text{professor})$$

all ranks for professors (SQL)

```
select rank
from professor;
```

result
rank
C4
C4
C4
C4
C3
C3
C3

Simple SQL Queries – duplicates

Example (eliminating duplicates)

all ranks for professors without duplciates

```
select distinct rank
from professor;
```

Simple SQL Queries – duplicates

Example (eliminating duplicates)

all ranks for professors without duplciates

```
select distinct rank
from professor;
```

result
rank
C4
C3

Simple SQL Queries – duplicates

Example (eliminating duplicates)

all ranks for professors without duplciates

```
select distinct rank
from professor;
```

result
rank
C4
C3

`distinct` . . . eliminates duplicates from the result

Simple SQL Queries

Example

find the room number of Professor Popper

Simple SQL Queries

Example

find the room number of Professor Popper

```
select name, room
from professor
where name='Popper';
```

$$\pi_{\text{name,room}}\left(\sigma_{name='Popper'}(\text{professor})\right)$$

Simple SQL Queries

Example

find the room number of Professor Popper

```
select name, room
from professor
where name='Popper';
```

$$\pi_{\text{name,room}}\left(\sigma_{name='Popper'}(\text{professor})\right)$$

professor	
name	room
Popper	52

Simple SQL Queries

structure of a simple query:

```
select attributes
from table
where condition;
```

Simple SQL Queries

structure of a simple query:

```
select attributes
from table
where condition;
```

attributes: list of attributes that will be displayed

table: name of the table that will be searched

condition: condition that every displayed tuple has to satisfy

Simple SQL Queries

Example

persNr and name of all C4-professors

professor

persNr	name	rank	room
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

result

persNr	name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Simple SQL Queries

Example

persNr and name of all C4-professors

professor

persNr	name	rank	room
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

result

persNr	name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

```
select   persNr, name
from professor
where  rank='C4';
```

Simple SQL Queries – Sorting

Example (sorting)

persNr name and rank of all professors, sorted by rank in descending order and by name in ascending order

result		
persNr	name	rank
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Simple SQL Queries – Sorting

Example (sorting)

persNr name and rank of all professors, sorted by rank in descending order and by name in ascending order

```
select persNr, name, rank
from   professor
order by rank desc, name asc;
```

result

persNr	name	rank
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Simple SQL Queries – Sorting

Example (sorting)

persNr and name of all professors, sorted by rank in descending order and by name in ascending order

```
select persNr, name
from   professor
order by rank desc, name asc;
```

result

persNr	name
2136	Curie
2137	Kant
2126	Russel
2125	Sokrates
2134	Augustinus
2127	Kopernikus
2133	Popper

Simple SQL Queries

Example

Which professors give the course Mäeutik?

Simple SQL Queries

Example

Which professors give the course Mäeutik?
information from table: `professor, lecture`

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

Simple SQL Queries

Example

Which professors give the course Mäeutik?
information from table: `professor, lecture`

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select name, title
from professor, lecture
where persNr = givenBy and
      title = 'Mäeutik';
```

Queries over Several Relations

Queries over Several Relations

structure of a query over several tables:

```
select attributes
from table1, table2, ..., tableN
where conditions;
```

Queries over Several Relations

structure of a query over several tables:

```
select attributes
from table1, table2, ..., tableN
where conditions;
```

conditions: over attributes in the various tables

Queries over Several Relations

structure of a query over several tables:

```
select attributes
from table1, table2, ..., tableN
where conditions;
```

conditions: over attributes in the various tables

evaluation:

1. construct the cross product of the `from` tables
2. check for every row the `where` conditions
3. project to the `select` attributes

Queries over Several Relations

Example

Which professors give the course Mäeutik?
information from tables `professor, lecture`

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select name, title
from professor, lecture
where persNr = givenBy and
      title = 'Mäeutik';
```

Queries over Several Relations

Example

Which professors give the course Mäeutik?
information from tables `professor, lecture`

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select name, title
from professor, lecture
where persNr = givenBy and
      title = 'Mäeutik';
```

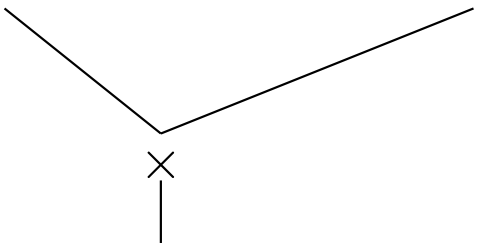
$$\pi_{\text{name,title}}\left(\sigma_{\text{persNr=givenBy}\wedge\text{title='Mäeutik'}}(\text{professor} \times \text{lecture})\right)$$

Queries over Several Relations – Processing

1. construct the cross product of the `from` tables

professor

persNr	name	rank	room
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



lecture

lecNr	title	SWS	persNr
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 drei Kritiken	4	2137

Queries over Several Relations – Processing

professor × *lecture*							
persNr	name	rank	room	lecNr	title	SWS	lPersNr
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

Queries over Several Relations – Processing

$professor \times lecture$

persNr	name	rank	room	lecNr	title	SWS	lPersNr
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

2. check for every row the `where` conditions

$\sigma_{persNr=VPersNr \wedge title='Mäeutik'}(professor \times lecture)$

persNr	name	rank	room	lecNr	title	SWS	lPersNr
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

Queries over Several Relations – Processing

$\sigma_{persNr=VPersNr \wedge title='Mäeutik'}(professor \times lecture)$

persNr	name	rank	room	lecNr	title	SWS	lPersNr
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

Queries over Several Relations – Processing

$\sigma_{persNr=VPersNr \wedge title='Mäeutik'}(professor \times lecture)$

persNr	name	rank	room	lecNr	title	SWS	lPersNr
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

3. project to the `select` attributes

Queries over Several Relations – Processing

$\sigma_{persNr=VPersNr \wedge title='Mäeutik'}(professor \times lecture)$							
persNr	name	rank	room	lecNr	title	SWS	lPersNr
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

3. project to the `select` attributes

$\pi_{name,title}(\ldots)$	
name	title
Sokrates	Mäeutik

Queries over Several Relations

Example

name and matrNr of students and the title of the lectures they are attending

```
student(matrNr, name, sem)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

Queries over Several Relations

Example

name and matrNr of students and the title of the lectures they are attending

```
student(matrNr, name, sem)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

```
select student.matrNr, name, title
from student, attend, lecture
where student.matrNr = attend.matrNr and
      attend.lecNr = lecture.lecNr;
```

Place Holder for Tables and Renaming of Attributes

Example

same query for placing pace holders for table names:

```
select s.matrNr, s.name, l.title
from student s, attend a, lecture l
where s.matrNr = a.matrNr and
      a.lecNr = l.lecNr;
```

Place Holder for Tables and Renaming of Attributes

Example

same query for placing pace holders for table names:

```
select s.matrNr, s.name, l.title
from student s, attend a, lecture l
where s.matrNr = a.matrNr and
      a.lecNr = l.lecNr;
```

Example

find the lecture number of the lectures that are second-level predecessors of the lecture 5216 (= predecessor of the predecessor of 5216)

```
presuppose(predNr, sucNr)
```

```
select v1.predNr as pred_order_2
from presuppose v1, presuppose v2
where v1.sucNr=v2.predNr and v2.sucNr=5216;
```

Queries over Several Relations

structure of a query over several relations

```
select  attribute1 [ as ] a1,
        attribute2 [ as ] a2,
        ...,
        attributeM [ as ] aM
from  table1 [ as ] t1,
      table2 [ as ] t2,
      ...,
      tableN [ as ] tN
where  conditions;
```

Translation to Relational Algebra

general structure of a (non-nested) SQL query

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

Translation to Relational Algebra

general structure of a (non-nested) SQL query

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

turns into:

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

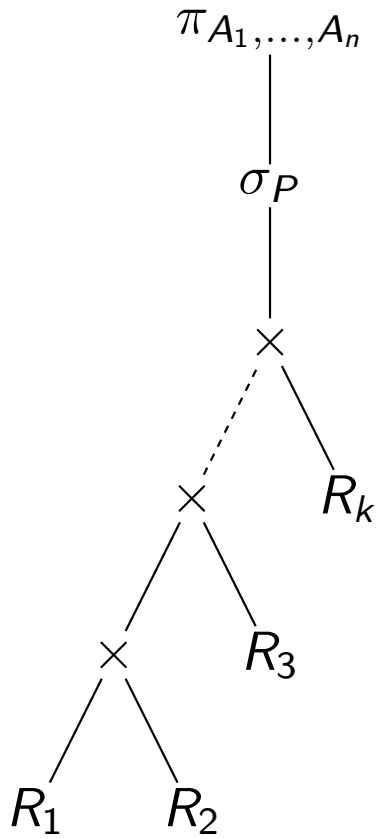
Translation to Relational Algebra

general structure of a (non-nested) SQL query

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

turns into:

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



Set Operations

Set Operations

queries with type compatible attributes in the result can be connected using set operations

without duplicates: `union`, `intersect`, `except` (`minus`)

with duplicates: `union all`, `intersect all`, `except all`

Set Operations

queries with type compatible attributes in the result can be connected using set operations

without duplicates: `union`, `intersect`, `except` (`minus`)

with duplicates: `union all`, `intersect all`, `except all`

Example

names of all assistants or professors

```
( select name from assistant )
union
( select name from professor );
```

Aggregate Functions

Aggregate Functions

aggregate functions are operations that rather than operating on single tuples, operate on a set of tuples

Aggregate Functions

aggregate functions are operations that rather than operating on single tuples, operate on a set of tuples

- `avg()`, `max()`, `min()`, `sum()` calculate the value of a set of tuples

Aggregate Functions

aggregate functions are operations that rather than operating on single tuples, operate on a set of tuples

- `avg()`, `max()`, `min()`, `sum()` calculate the value of a set of tuples
- `count()` counts the number of tuples in a set

Aggregate Functions

Example

average/maximal/minimal number of semesters a student is enrolled in; number of students

Aggregate Functions

Example

average/maximal/minimal number of semesters a student is enrolled in; number of students

student		
matrNr	name	sem
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Aggregate Functions

Example

average/maximal/minimal number of semesters a student is enrolled in; number of students

```
select avg(semester)
from student;

select max(semester)
from student;

select min(semester)
from student;

select count(matrNr)
from student;
```

student

matrNr	name	sem
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Aggregate Functions

Example

What is the sum of hours of lectures given by C4-professors?

lecture

lecNr	title	SWS	givenBy
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5259	Der Wiener Kreis	2	2133
5216	Bioethik	2	2126
⋮	⋮	⋮	⋮

professor

persNr	name	rank
2125	Sokrates	C4
2126	Russel	C4
2133	Popper	C3
⋮	⋮	⋮

Aggregate Functions

Example

What is the sum of hours of lectures given by C4-professors?

```
select sum (SWS)
from   lecture, professor
where  givenBy = persNr and rank = 'C4';
```

lecture

lecNr	title	SWS	givenBy
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5259	Der Wiener Kreis	2	2133
5216	Bioethik	2	2126
⋮	⋮	⋮	⋮

professor

persNr	name	rank
2125	Sokrates	C4
2126	Russel	C4
2133	Popper	C3
⋮	⋮	⋮

Aggregates and Grouping

Aggregates and Grouping

problem: We do not want to calculate the sum of all hours of lectures, but for each lecturer the sum of hours of lectures given by him/her.

Aggregates and Grouping

problem: We do not want to calculate the sum of all hours of lectures, but for each lecturer the sum of hours of lectures given by him/her.

solution: construct for each lecturer a group (with `group by`) and calculate the sum within this group

Aggregates and Grouping

problem: We do not want to calculate the sum of all hours of lectures, but for each lecturer the sum of hours of lectures given by him/her.

solution: construct for each lecturer a group (with `group by`) and calculate the sum within this group

in general: all rows in a table, that take the same value at the attributes of `group by` are combined to a group and the aggregate functions are evaluated corresponding to this group

Aggregates and Grouping

problem: We do not want to calculate the sum of all hours of lectures, but for each lecturer the sum of hours of lectures given by him/her.

solution: construct for each lecturer a group (with `group by`) and calculate the sum within this group

in general: all rows in a table, that take the same value at the attributes of `group by` are combined to a group and the aggregate functions are evaluated corresponding to this group

conditions for the aggregated values are listed using having

Ex: Aggregates and Grouping

Example

return for every C4 professor the sum of hours of lectures given by him/her

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

Ex: Aggregates and Grouping

Example

return for every C4 professor the sum of hours of lectures given by him/her

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select givenBy, sum (SWS)
from lecture, professor
where givenBy = persNr and rank = 'C4'
group by givenBy;
```

Ex: Aggregates and Grouping

Example

return for every C4 professor lecturing long lectures (average ≥ 3) the sum of hours of lectures given by him/her

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

Ex: Aggregates and Grouping

Example

return for every C4 professor lecturing long lectures (average ≥ 3) the sum of hours of lectures given by him/her

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select givenBy, name, sum (SWS)
from lecture, professor
where givenBy = persNr and rank = 'C4'
group by givenBy, name
having avg (SWS) >= 3;
```

Aggregates and Grouping – Processing

`from lecture, professor`

professor × *lecture*

persNr	name	rank	room	lecNr	title	SWS	givenBy
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2133	Popper	C3	52	5001	Grundzüge	4	2137
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

where-condition

Aggregates and Grouping

where-condition

```
where givenBy = persNr and rank = 'C4'
```

persNr	name	rank	room	lecNr	title	SWS	givenBy
2137	Kant	C4	7	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
2126	Russel	C4	232	5043	Erkenntnistheorie	3	2126
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
2125	Sokrates	C4	226	4052	Logik	4	2125
2126	Russel	C4	232	5052	Wissenschaftstheorie	3	2126
2126	Russel	C4	232	5216	Bioethik	2	2126
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

grouping

Aggregates and Grouping

grouping

```
group by givenBy, name
```

persNr	name	rank	room	lecNr	title	SWS	givenBy
2125	Sokrates	C4	226	5041	Ethik	4	2125
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
2125	Sokrates	C4	226	4052	Logik	4	2125
2126	Russel	C4	232	5043	Erkenntnistheorie	3	2126
2126	Russel	C4	232	5052	Wissenschaftstheorie	3	2126
2126	Russel	C4	232	5216	Bioethik	2	2126
2137	Kant	C4	7	5001	Grundzüge	4	2137
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

having-condition

Aggregates and Grouping

having-condition

```
having avg(SWS)>= 3
```

persNr	name	rank	room	lecNr	title	SWS	givenBy
2125	Sokrates	C4	226	5041	Ethik	4	2125
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
2125	Sokrates	C4	226	4052	Logik	4	2125
2137	Kant	C4	7	5001	Grundzüge	4	2137
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

Aggregates and Grouping

having-condition

```
having avg(SWS)>= 3
```

persNr	name	rank	room	lecNr	title	SWS	givenBy
2125	Sokrates	C4	226	5041	Ethik	4	2125
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
2125	Sokrates	C4	226	4052	Logik	4	2125
2137	Kant	C4	7	5001	Grundzüge	4	2137
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

aggregation (sum) and projection

```
select givenBy, name, sum(SWS)
```

givenBy	name	sum(SWS)
2125	Sokrates	10
2137	Kant	8

Aggregates and Grouping

attention: SQL constructs a resulting tuple per group

$\Rightarrow$ all `select` listed attributes – besides the aggregated ones – have to be listed in `group by` as well
this way we ensure that the displayed attributes do not change within the group

Aggregates and Grouping

attention: SQL constructs a resulting tuple per group

$\Rightarrow$ all `select` listed attributes – besides the aggregated ones – have to be listed in `group by` as well
this way we ensure that the displayed attributes do not change within the group

```
select givenBy, name, sum (SWS)
.........
group by givenBy, name;
```

Ex: Aggregates and Grouping

Example

names of students that have studied for the longest period of time

```
student(matrNr, name, semester)
```

```
select name, max(semester)
from student;
```

Ex: Aggregates and Grouping

Example

names of students that have studied for the longest period of time

```
student(matrNr, name, semester)
```

```
select name, max(semester)
from student;
```

$\Rightarrow$ ORA-00937: not a single-group group function

Ex: Aggregates and Grouping

Example

names of students that have studied for the longest period of time

```
student(matrNr, name, semester)
```

```
select name, max(semester)
from student;
```

⇒ ORA-00937: not a single-group group function

⇒ ERROR: column ‘‘studenten.name’’ must appear in the GROUP BY clause or be used in an aggregate function

Ex: Aggregates and Grouping

Example (second attempt)

names of students that have studied for the longest period of time

```
student(matrNr, name, semester)
```

```
select name, max(semester)
from student
group by name;
```

Ex: Aggregates and Grouping

Example (second attempt)

names of students that have studied for the longest period of time

```
student(matrNr, name, semester)
```

```
select name, max(semester)
from student
group by name;
```

$\Rightarrow$ all tuples ?!?

Ex: Aggregates and Grouping

Example (second attempt)

names of students that have studied for the longest period of time

```
student(matrNr, name, semester)
```

```
select name, max(semester)
from student
group by name;
```

$\Rightarrow$ all tuples ?!?

solution: nested query

Nested Queries

Nested Queries

there are several possibilities to connect `select` expressions
the following classification depends on the result of the sub query (a value or a relation)

- result of the sub query consists of a tuple with an attribute (=value)
  - sub query instead of a scalar value in `select` resp. `where`

Nested Queries

there are several possibilities to connect `select` expressions
the following classification depends on the result of the sub query (a value or a relation)

- result of the sub query consists of a tuple with an attribute (=value)
  - sub query instead of a scalar value in `select` resp. `where`
- result of the sub query is a relation
  - sub query in `from`
  - set comparisons: `in`, `not in`, `any`, `all`
  - connection via quantifiers: `exists`

Nested Queries

there are several possibilities to connect `select` expressions
the following classification depends on the result of the sub query (a value or a relation)

- result of the sub query consists of a tuple with an attribute (=value)
  - sub query instead of a scalar value in `select` resp. `where`
- result of the sub query is a relation
  - sub query in `from`
  - set comparisons: `in`, `not in`, `any`, `all`
  - connection via quantifiers: `exists`

when working with nested queries it is essential for run time whether we work with correlated or uncorrelated queries

Nested Queries – Scalar Values as Result

sub query in `where`

Nested Queries – Scalar Values as Result

sub query in `where`

Example

names of students that have studied for the longest period of time

```
student(matrNr, name, semester)
```

Nested Queries – Scalar Values as Result

sub query in `where`

Example

names of students that have studied for the longest period of time

```
student(matrNr, name, semester)
```

```
select name
from student
where semester = (select max (semester)
                  from student );
```

Nested Queries – Scalar Values as Result

sub query in `select`

Nested Queries – Scalar Values as Result

sub query in `select`

Example

for each lecturer return the number of hours he or she is teaching

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select p.name, (select sum (SWS)
                from lecture
                where givenBy=p.persNr)
from professor p;
```

Nested Queries – Scalar Values as Result

sub query in `select`

Example

for each lecturer return the number of hours he or she is teaching

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select p.name, (select sum (SWS)
                from lecture
                where givenBy=p.persNr)
from professor p;
```

attention: the sub query is correlating (=it works with attributes of the (super) query), for each resulting tuple the sub query is executed once

Nested Queries – Scalar Values as Result

Example (de-nesting via `group by`)

for each lecturer return the number of hours he or she is teaching

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

Nested Queries – Scalar Values as Result

Example (de-nesting via group by)

for each lecturer return the number of hours he or she is teaching

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select p.persNr, p.name, sum (SWS)
from professor p, lecture v
where v.givenBy=p.persNr
group by p.persNr, p.name
```

Nested Queries – Tables as Result

sub query in `from`

Nested Queries – Tables as Result

sub query in `from`

Example

find all students that attend more than 2 lectures

```
student(matrNr, name, semester)
attend(matrNr, lecNr)
```

Nested Queries – Tables as Result

sub query in `from`

Example

find all students that attend more than 2 lectures

```
student(matrNr, name, semester)
attend(matrNr, lecNr)
```

```
select tmp.matrNr, tmp.name, tmp.lecAmount
from (select s.matrNr, s.name,
             count(*) as lecAmount
      from student s, attend h
      where s.matrNr=h.matrNr
      group by s.matrNr, s.name) tmp
where tmp.lecAmount > 2;
```

Nested Queries – Tables as Result

Example (de-nesting via having)

find all students that attend more than 2 lectures

```
student(matrNr, name, semester)
attend(matrNr, lecNr)
```

```
select s.matrNr, s.name, count(*) as lecAmount
from student s, attend h
where s.matrNr = h.matrNr
group by s.matrNr, s.name
having count (*) > 2;
```

Nested Queries – Set Comparisons

sub query in `where`

Nested Queries – Set Comparisons

sub query in `where`
set comparison with `in`/`not in`

Nested Queries – Set Comparisons

sub query in `where`
set comparison with `in`/`not in`

Example

find all professors that do not give any lectures

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

Nested Queries – Set Comparisons

sub query in `where`
set comparison with `in`/`not in`

Example

find all professors that do not give any lectures

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select name
from professor
where persNr not in (select givenBy
                     from lecture);
```

Nested Queries – Set Comparisons

set comparison with `in`/`not in`

Example

find for all students the lectures that they have attended and where they achieved a positive grade

```
attend(matrNr, lecNr)
examine(matrNr, lecNr, persNr, grade)
```

Nested Queries – Set Comparisons

set comparison with `in`/`not in`

Example

find for all students the lectures that they have attended and where they achieved a positive grade

```
attend(matrNr, lecNr)
examine(matrNr, lecNr, persNr, grade)
```

```
select matrNr, lecNr
from attend
where (matrNr, lecNr) in (select matrNr, lecNr
                          from examine
                          where grade < 5);
```

Nested Queries – Set Comparisons

Example (de-nesting via `intersect`)

find for all students the lectures that they have attended and where they achieved a positive grade

```
attend(matrNr, lecNr)
examine(matrNr, lecNr, persNr, grade)
```

Nested Queries – Set Comparisons

Example (de-nesting via `intersect`)

find for all students the lectures that they have attended and where they achieved a positive grade

```
attend(matrNr, lecNr)
examine(matrNr, lecNr, persNr, grade)
```

```
(select matrNr, lecNr
 from attend)
intersect
(select matrNr, lecNr
 from examine
 where grade < 5);
```

Nested Queries – Set Comparisons

set comparisons with `any`/`all` (attention: no universal quantifier, only the comparison of a value with a set of values)

Nested Queries – Set Comparisons

set comparisons with `any`/`all` (attention: no universal quantifier, only the comparison of a value with a set of values)

Example

find all students that have studied for the longest period of time

```
select name
from student
where semester >= all (select semester
                       from student);
```

Nested Queries – Set Comparisons

set comparisons with `any`/`all` (attention: no universal quantifier, only the comparison of a value with a set of values)

Example

find all students that have studied for the longest period of time

```
select name
from student
where semester >= all (select semester
                       from student);
```

equivalent to

```
select name
from student
where semester = (select max(semester)
                 from student);
```

Nested Queries – Set Comparisons

Example

find all students that are not studying for the longest period of time

```
select name
from student
where semester < any (select semester
                      from student);
```

Nested Queries – Set Comparisons

Example

find all students that are not studying for the longest period of time

```
select name
from student
where semester < any (select semester
                      from student);
```

equivalent to:

```
select name
from student
where semester < (select max(semester)
                  from student);
```

Nested Queries – Aggregate Functions

nested aggregate functions are not allowed
$\Rightarrow$ use sub queries

Nested Queries – Aggregate Functions

nested aggregate functions are not allowed
$\Rightarrow$ use sub queries

Example

find C4 professors with the highest number of lecturing hours

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

Nested Queries – Aggregate Functions

nested aggregate functions are not allowed
$\Rightarrow$ use sub queries

Example

find C4 professors with the highest number of lecturing hours

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select givenBy, max(sum(SWS))
from lecture, professor
where givenBy = persNr and rank = 'C4'
group by givenBy;
```

Nested Queries – Aggregate Functions

nested aggregate functions are not allowed
$\Rightarrow$ use sub queries

Example

find C4 professors with the highest number of lecturing hours

Nested Queries – Aggregate Functions

nested aggregate functions are not allowed
$\Rightarrow$ use sub queries

Example

find C4 professors with the highest number of lecturing hours

```
select givenBy, sum(SWS)
from lecture, professor
where givenBy=persNr and rank='C4'
group by givenBy
having sum(SWS) >= all(
            select sum(SWS)
            from lecture, professor
            where givenBy=persNr and rank='C4'
            group by givenBy);
```

Nested Queries – Aggregate Functions

nested aggregate functions are not allowed – even when the queries are correct ones

Example

find the highest number of lecturing hours that is given by a C4 professor

Nested Queries – Aggregate Functions

nested aggregate functions are not allowed – even when the queries are correct ones

Example

find the highest number of lecturing hours that is given by a C4 professor

```
select max(sum(SWS))
from lecture, professor
where givenBy=persNr and rank='C4'
group by givenBy;
```

Existentially Quantified Queries

Existentially Quantified Queries

connection with a sub query via `exists`

`exists` checks whether the sub query contains tuples or not

Existentially Quantified Queries

connection with a sub query via `exists`

`exists` checks whether the sub query contains tuples or not

Example

find all students that are older than the youngest professor

Existentially Quantified Queries

connection with a sub query via `exists`

`exists` checks whether the sub query contains tuples or not

Example

find all students that are older than the youngest professor

```
select s.*
from student s
where exists (select p.*
              from professor p
              where p.birthDate > s.birthDate);
```

Existentially Quantified Queries

connection with a sub query via `exists`

`exists` checks whether the sub query contains tuples or not

Example

find all students that are older than the youngest professor

```
select s.*
from student s
where exists (select p.*
              from professor p
              where p.birthDate > s.birthDate);
```

correlated sub query (`s.birthDate` and `student s`)!

Existentially Quantified Queries

a non correlating solution is

Example (non correlating)

find all students that are older than the youngest professor

```
select s.*
from student s
where s.birthDate < (select max(p.birthDate)
                     from professor p);
```

Existentially Quantified Queries

a non correlating solution is

Example (non correlating)

find all students that are older than the youngest professor

```
select s.*
from student s
where s.birthDate < (select max(p.birthDate)
                     from professor p);
```

or

```
select s.* from student s
where s.birthDate < any (select p.birthDate
                         from professor p);
```

or ...

Existentially Quantified Queries

Example

find all professors that do not lecture

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

Existentially Quantified Queries

Example

find all professors that do not lecture

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select name
from professor
where not exists (select *
                  from lecture
                  where givenBy = persNr);
```

Existentially Quantified Queries

Example

find all professors that do not lecture

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)
```

```
select name
from professor
where not exists (select *
                  from lecture
                  where givenBy = persNr);
```

correlating sub query (persNr and professor)!

Existentially Quantified Queries

a non-correlating solution is:

```
select name
from professor
where persNr not in (select givenBy
                     from lecture);
```

or

Existentially Quantified Queries

a non-correlating solution is:

```
select name
from professor
where persNr not in (select givenBy
                     from lecture);
```

or

```
(select persNr
 from professor)
except
(select givenBy
 from lecture);
```

Existentially Quantified Queries

Example

find all assistants that work for a professor who is younger than themselves

```
professor(persNr, name, rank, room, birthDate)
assistant(persNr, name, birthDate, boss)
```

Existentially Quantified Queries

Example

find all assistants that work for a professor who is younger than themselves

```
professor(persNr, name, rank, room, birthDate)
assistant(persNr, name, birthDate, boss)
```

```
select a.*
from assistant a
where exists (select p.*
              from professor p
              where a.boss = p.persNr and
              p.birthDate > a.birthDate);
```

Existentially Quantified Queries

Example

find all assistants that work for a professor who is younger than themselves

```
professor(persNr, name, rank, room, birthDate)
assistant(persNr, name, birthDate, boss)
```

```
select a.*
from assistant a
where exists (select p.*
              from professor p
              where a.boss = p.persNr and
              p.birthDate > a.birthDate);
```

correlating sub query (`a.birthDate` and `assistant a`)!

Existentially Quantified Queries

Example

find all assistants that work for a professor who is younger than themselves

```
professor(persNr, name, rank, room, birthDate)
assistant(persNr, name, birthDate, boss);
```

Existentially Quantified Queries

Example

find all assistants that work for a professor who is younger than themselves

```
professor(persNr, name, rank, room, birthDate)
assistant(persNr, name, birthDate, boss);
```

a non-correlating solution is:

```
select a.*
from assistant a, professor p
where a.boss = p.persNr and
      p.birthDate > a.birthDate;
```

Universally Quantified Queries

Universally Quantified Queries

Example

find students that have attended all 4h lectures

```
student(matrNr, name, semester)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

Universally Quantified Queries

Example

find students that have attended all 4h lectures

```
student(matrNr, name, semester)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

$$\text{attend} \div \pi_{\text{lecNr}}\left(\sigma_{\text{SWS}=4}(\text{lecture})\right)$$

Universally Quantified Queries

Example

find students that have attended all 4h lectures

```
student(matrNr, name, semester)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

$$\text{attend} \div \pi_{\text{lecNr}}\left(\sigma_{\text{SWS}=4}(\text{lecture})\right)$$

no universal quantifiers in SQL

Universally Quantified Queries – Not Directly in SQL

Example

find students that have attended all 4h lectures

no universal quantifiers in SQL
realizable through

Universally Quantified Queries – Not Directly in SQL

Example

find students that have attended all 4h lectures

no universal quantifiers in SQL
realizable through

1. logical equivalence (via 2 $\times$ `not exists`)

Universally Quantified Queries – Not Directly in SQL

Example

find students that have attended all 4h lectures

no universal quantifiers in SQL
realizable through

1. logical equivalence (via 2 $\times$ `not exists`)
2. subsets (via `exists` and `except`)

Universally Quantified Queries – Not Directly in SQL

Example

find students that have attended all 4h lectures

no universal quantifiers in SQL
realizable through

1. logical equivalence (via 2 $\times$ `not exists`)
2. subsets (via `exists` and `except`)
3. counting (via `count`)

Universally Quantified Queries – Not Directly in SQL

Example

find students that have attended all 4h lectures

no universal quantifiers in SQL
realizable through

1. logical equivalence (via 2 × `not exists`)
2. subsets (via `exists` and `except`)
3. counting (via `count`)
4. division (via `except`)

Universally Quantified Queries – 1. "Logical Transformation"

translation in equivalent query with negation and existential quantifier (predicate logic)

Universally Quantified Queries – 1. "Logical Transformation"

translation in equivalent query with negation and existential quantifier (predicate logic)

$$\forall x F(x) \Leftrightarrow \neg \exists x (\neg F(x))$$

Universally Quantified Queries – 1. "Logical Transformation"

translation in equivalent query with negation and existential quantifier (predicate logic)

$$\forall x F(x) \Leftrightarrow \neg \exists x (\neg F(x))$$

Example

find students that have attended all 4h lectures

$\Leftrightarrow$ find students for which it holds that:
attended all 4h lectures

Universally Quantified Queries – 1. "Logical Transformation"

translation in equivalent query with negation and existential quantifier (predicate logic)

$$\forall x F(x) \Leftrightarrow \neg \exists x (\neg F(x))$$

Example

find students that have attended all 4h lectures

$\Leftrightarrow$ find students for which it holds that: attended all 4h lectures

$\Leftrightarrow$ find all students for which it does not hold that: have not attended one 4h lecture

Universally Quantified Queries – 1. "Logical Transformation"

translation in equivalent query with negation and existential quantifier (predicate logic)

$$\forall x F(x) \Leftrightarrow \neg \exists x (\neg F(x))$$

Example

find students that have attended all 4h lectures

$\Leftrightarrow$ find students for which it holds that:
attended all 4h lectures

$\Leftrightarrow$ find all students for which it does not hold that:
have not attended one 4h lecture

$\Leftrightarrow$ find students for which it does not hold that:
there exists a 4h lecture
that the student has not attended

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:
there is a 4h lecture for which it does not hold:

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:
there is a 4h lecture for which it does not hold:
the student attended this lecture

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:
there is a 4h lecture for which it does not hold:
the student attended this lecture

Example

```
select s.*
from student s
where not exists
```

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:
there is a 4h lecture for which it does not hold:
the student attended this lecture

Example

```
select s.*
from student s
where not exists
          (select *
          from lecture v
          where v.SWS = 4 and
          not exists
```

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:
there is a 4h lecture for which it does not hold:
the student attended this lecture

Example

```
select s.*
from student s
where not exists
          (select *
          from lecture v
          where v.SWS = 4 and
          not exists (select *
                      from attend h
                      where h.lecNr = v.lecNr
                      and s.matrNr = h.matrNr));
```

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:
there is a 4h lecture for which it does not hold:

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:
there is a 4h lecture for which it does not hold:
the student attended this lecture

Universally Quantified Queries – 1. "Logical Transformation"

SQL translation directly follows from:

find students for which it does not hold:
there is a 4h lecture for which it does not hold:
the student attended this lecture

Example

```
select s.*
from student s
where not exists
          (select *
          from lecture v
          where v.SWS = 4 and
          s.matrNr not in (select h.matrNr
                           from attend h
                           where h.lecNr = v.lecNr));
```

Universally Quantified Queries – 1. "Logical Transformation"

transformation to tuple relational calculus

Example

```
student(matrNr, name, sem)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

Universally Quantified Queries – 1. "Logical Transformation"

transformation to tuple relational calculus

Example

```
student(matrNr, name, sem)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

$$\{s \mid s \in student \wedge \forall l \in lecture(l.SWS = 4 \rightarrow \exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr))\}$$

Universally Quantified Queries – 1. "Logical Transformation"

Example

$$\{s \mid s \in student \wedge \forall l \in lecture(l.SWS = 4 \rightarrow \exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr))\}$$

Universally Quantified Queries – 1. "Logical Transformation"

Example

$$\{s \mid s \in student \wedge \forall l \in lecture(l.SWS = 4 \rightarrow \exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr))\}$$

$$\Leftrightarrow$$

$$\{s \mid s \in student \wedge \neg\exists l \in lecture\neg(l.SWS = 4 \rightarrow \exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr))\}$$

Universally Quantified Queries – 1. "Logical Transformation"

Example

$$\{s \mid s \in student \wedge \forall l \in lecture(l.SWS = 4 \rightarrow \exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr))\}$$

$$\Leftrightarrow$$

$$\{s \mid s \in student \wedge \neg\exists l \in lecture\neg(l.SWS = 4 \rightarrow \exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr))\}$$

$$\Leftrightarrow$$

$$\{s \mid s \in student \wedge \neg\exists l \in lecture\neg(\neg(l.SWS = 4) \vee (\exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr)))\}$$

Universally Quantified Queries – 1. "Logical Transformation"

Example

$$\{s \mid s \in student \wedge \forall l \in lecture(l.SWS = 4 \rightarrow \exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr))\}$$

$$\Leftrightarrow$$

$$\{s \mid s \in student \wedge \neg\exists l \in lecture\neg(l.SWS = 4 \rightarrow \exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr))\}$$

$$\Leftrightarrow$$

$$\{s \mid s \in student \wedge \neg\exists l \in lecture\neg(\neg(l.SWS = 4) \vee (\exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr)))\}$$

$$\Leftrightarrow$$

$$\{s \mid s \in student \wedge \neg\exists l \in lecture((l.SWS = 4) \wedge (\neg\exists a \in attend(a.lecNr = l.lecNr \wedge a.matrNr = s.matrNr)))\}$$

Universally Quantified Queries – 1. "Logical Transformation"

Example

$$\{s \mid s \in student \wedge \neg\exists l \in lecture((l.SWS = 4) \wedge (\neg\exists a \in attend(l.lecNr = a.lecNr \wedge s.matrNr = a.matrNr)))\}$$

```
select s.*
from student s
where not exists
        (select *
        from lecture l
        where l.SWS = 4 and
         not exists
```

Universally Quantified Queries – 1. "Logical Transformation"

Example

$$\{s \mid s \in student \wedge \neg\exists l \in lecture((l.SWS = 4) \wedge (\neg\exists a \in attend(l.lecNr = a.lecNr \wedge s.matrNr = a.matrNr)))\}$$

```
select s.*
from student s
where not exists
        (select *
        from lecture l
        where l.SWS = 4 and
         not exists (select *
                     from attend a
                     where a.lecNr = l.lecNr
                     and s.matrNr = a.matrNr));
```

Universally Quantified Queries – 2. "Subsets"

find all students M for which it holds:
set of all 4h long lectures $\subseteq$ set of lectures attended by M

Universally Quantified Queries – 2. "Subsets"

find all students M for which it holds:
set of all 4h long lectures $\subseteq$ set of lectures attended by M

"$\subseteq$" no operator in SQL, but

Universally Quantified Queries – 2. "Subsets"

find all students M for which it holds:
set of all 4h long lectures $\subseteq$ set of lectures attended by M

"$\subseteq$" no operator in SQL, but

set of all 4h long lectures $\subseteq$ set of lectures attended by M

$$\Leftrightarrow$$

set of all 4h long lectures $-$ set of lectures attended by $M = \emptyset$

Universally Quantified Queries – 2. "Subsets"

find all students M for which it holds:
set of all 4h long lectures $\subseteq$ set of lectures attended by M

"$\subseteq$" no operator in SQL, but

set of all 4h long lectures $\subseteq$ set of lectures attended by M

$$\Leftrightarrow$$

set of all 4h long lectures $-$ set of lectures attended by $M = \emptyset$

$\emptyset$ . . . there is no lecture in the set difference

Universally Quantified Queries – 2. "Subsets"

find all students M for which it does not hold:
there is a lecture in the difference
set of all 4h long lectures − set of all lectures attended by M

Example

```
select s.*
from student s
where not exists
          ((select lecNr
          from lecture l
          where l.SWS = 4)
          except
          (select lecNr
          from attend a
          where a.matrNr = s.matrNr))
```

Universally Quantified Queries – 3. "Counting"

universal quantification can be expressed by a `count` aggregation

Example

find students that have attended all 4h lectures

Universally Quantified Queries – 3. "Counting"

universal quantification can be expressed by a `count` aggregation

Example

find students that have attended all 4h lectures

1. count how many 4h lectures each student has attended

Universally Quantified Queries – 3. "Counting"

universal quantification can be expressed by a `count` aggregation

Example

find students that have attended all 4h lectures

1. count how many 4h lectures each student has attended
2. count how many 4h lectures there are.

Universally Quantified Queries – 3. "Counting"

universal quantification can be expressed by a `count` aggregation

Example

find students that have attended all 4h lectures

1. count how many 4h lectures each student has attended
2. count how many 4h lectures there are.

```
select s.matrNr, s.name
from student s, attend a, lecture l
where s.matrNr = a.matrNr and
      a.lecNr = l.lecNr and
      l.SWS = 4
group by s.matrNr, s.name
having count (*) =
```

Universally Quantified Queries – 3. "Counting"

universal quantification can be expressed by a `count` aggregation

Example

find students that have attended all 4h lectures

1. count how many 4h lectures each student has attended
2. count how many 4h lectures there are.

```
select s.matrNr, s.name
from student s, attend a, lecture l
where s.matrNr = a.matrNr and
      a.lecNr = l.lecNr and
      l.SWS = 4
group by s.matrNr, s.name
having count (*) = (select count (*) from lecture
                    where SWS = 4);
```

Universally Quantified Queries – 4. Relational Algebra?

Universally Quantified Queries – 4. Relational Algebra?

Example

students that have attended all 4h lectures

attend	
matrNr	lecNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5001
28106	4052
28106	4630
29120	5001
29120	5041
29120	5049

$\div$

attend $\div \pi_{lecNr}\sigma_{SWS=4}(lecture)$

$\pi_{lecNr}\sigma_{SWS=4}(lecture)$
lecNr
5001
5041
4052
4630

$=$

$R \div S$
matrNr
28106

Universally Quantified Queries – 4. Relational Algebra?

Example

students that have attended all 4h lectures

attend	
matrNr	lecNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5001
28106	4052
28106	4630
29120	5001
29120	5041
29120	5049

find students x, such that

$(x, 5001) \in \text{attend}, (x, 5041) \in \text{attend},$
$(x, 4052) \in \text{attend}, (x, 4630) \in \text{attend}$

Universally Quantified Queries – 4. Relational Algebra?

Example

students that have attended all 4h lectures

attend	
matrNr	lecNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5001
28106	4052
28106	4630
29120	5001
29120	5041
29120	5049

find students x, such that

$(x, 5001) \in \text{attend}, (x, 5041) \in \text{attend},$
$(x, 4052) \in \text{attend}, (x, 4630) \in \text{attend}$

1. find all students such that this does not hold

Universally Quantified Queries – 4. Relational Algebra?

Example

students that have attended all 4h lectures

attend	
matrNr	lecNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5001
28106	4052
28106	4630
29120	5001
29120	5041
29120	5049

find students x, such that
$(x, 5001) \in \text{attend}$, $(x, 5041) \in \text{attend}$,
$(x, 4052) \in \text{attend}$, $(x, 4630) \in \text{attend}$

1. find all students such that this does not hold

2. all "other" students are the ones we are looking for

Universally Quantified Queries – 4. Relational Algebra?

Example

students that have attended all 4h lectures

attend	
matrNr	lecNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5001
28106	4052
28106	4630
29120	5001
29120	5041
29120	5049

find students x, such that
$(x, 5001) \in \text{attend}$, $(x, 5041) \in \text{attend}$,
$(x, 4052) \in \text{attend}$, $(x, 4630) \in \text{attend}$

1. find all students such that this does not hold
   1. "maximum:" all students have attended all 4h lectures
2. all "other" students are the ones we are looking for

Universally Quantified Queries – 4. Relational Algebra?

Example

students that have attended all 4h lectures

attend	
matrNr	lecNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5001
28106	4052
28106	4630
29120	5001
29120	5041
29120	5049

find students x, such that
$(x, 5001) \in \text{attend}$, $(x, 5041) \in \text{attend}$,
$(x, 4052) \in \text{attend}$, $(x, 4630) \in \text{attend}$

1. find all students such that this does not hold
   1. "maximum:" all students have attended all 4h lectures
   2. Which of those pairs are not in attend?
2. all "other" students are the ones we are looking for

Universally Quantified Queries – 4. Relational Algebra?

Example

students that have attended all 4h lectures

attend	
matrNr	lecNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5001
28106	4052
28106	4630
29120	5001
29120	5041
29120	5049

find students x, such that
$(x, 5001) \in \text{attend}$, $(x, 5041) \in \text{attend}$,
$(x, 4052) \in \text{attend}$, $(x, 4630) \in \text{attend}$

1. find all students such that this does not hold
   1. "maximum:" all students have attended all 4h lectures
   2. Which of those pairs are not in attend?
   3. Which students occur in at least one of these pairs?
2. all "other" students are the ones we are looking for

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

1. construct all pairs of students and 4h lectures

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

1. construct all pairs of students and 4h lectures
   content of the DB when all students have attended all 4h lectures
   $\rightarrow$ search for deviations

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

1. construct all pairs of students and 4h lectures
   content of the DB when all students have attended all 4h lectures
   $\rightarrow$ search for deviations
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

1. construct all pairs of students and 4h lectures
   content of the DB when all students have attended all 4h lectures
   $\rightarrow$ search for deviations
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
   these pairs of students and 4h lectures are "missing" in the DB

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

1. construct all pairs of students and 4h lectures
   content of the DB when all students have attended all 4h lectures
   $\rightarrow$ search for deviations
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
   these pairs of students and 4h lectures are "missing" in the DB
3. collect the students from the remaining pairs

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

1. construct all pairs of students and 4h lectures
   content of the DB when all students have attended all 4h lectures
   $\rightarrow$ search for deviations
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
   these pairs of students and 4h lectures are "missing" in the DB
3. collect the students from the remaining pairs
   students that have not attended at least one 4h lecture

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

1. construct all pairs of students and 4h lectures
   content of the DB when all students have attended all 4h lectures
   $\rightarrow$ search for deviations
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
   these pairs of students and 4h lectures are "missing" in the DB
3. collect the students from the remaining pairs
   students that have not attended at least one 4h lecture
4. subtract these students from the set of all students

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

1. construct all pairs of students and 4h lectures
   content of the DB when all students have attended all 4h lectures
   $\rightarrow$ search for deviations
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
   these pairs of students and 4h lectures are "missing" in the DB
3. collect the students from the remaining pairs
   students that have not attended at least one 4h lecture
4. subtract these students from the set of all students
   students that have attended all 4h lectures

Universally Quantified Queries – 4. Relational Algebra?

division can be expressed by primitive operators

1. construct all pairs of students and 4h lectures
   content of the DB when all students have attended all 4h lectures
   $\rightarrow$ search for deviations
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
   these pairs of students and 4h lectures are "missing" in the DB
3. collect the students from the remaining pairs
   students that have not attended at least one 4h lecture
4. subtract these students from the set of all students
   students that have attended all 4h lectures

$$R \div S = \pi_{\mathcal{R}-\mathcal{S}}(R) - \pi_{\mathcal{R}-\mathcal{S}}((\pi_{\mathcal{R}-\mathcal{S}}(R) \times S) - R)$$

Universally Quantified Queries – 4. Relational Algebra?

1. construct all pairs of students and 4h lectures
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
3. collect the students from the remaining pairs
4. subtract these students from the set of all students

Example

;

Universally Quantified Queries – 4. Relational Algebra?

1. construct all pairs of students and 4h lectures
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
3. collect the students from the remaining pairs
4. subtract these students from the set of all students

Example

```
(select s.matrNr, l.lecNr
 from student s, lecture l
 where l.SWS = 4)

                                    ;
```

Universally Quantified Queries – 4. Relational Algebra?

1. construct all pairs of students and 4h lectures
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
3. collect the students from the remaining pairs
4. subtract these students from the set of all students

Example

```
(select s.matrNr, l.lecNr
 from student s, lecture l
 where l.SWS = 4)
except
(select * from attend)      ;
```

Universally Quantified Queries – 4. Relational Algebra?

1. construct all pairs of students and 4h lectures
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
3. collect the students from the remaining pairs
4. subtract these students from the set of all students

Example

```
(select sno.matrNr
 from ((select s.matrNr, l.lecNr
        from student s, lecture l
        where l.SWS = 4)
       except
       (select * from attend)) sno);
```

Universally Quantified Queries – 4. Relational Algebra?

1. construct all pairs of students and 4h lectures
2. delete all pairs (*student*, *lecture*) of lectures, that have been attended by a student (i.e. the content of `attend`)
3. collect the students from the remaining pairs
4. subtract these students from the set of all students

Example

```
(select syes.matrNr from student syes)
except
(select sno.matrNr
 from ((select s.matrNr, l.lecNr
        from student s, lecture l
        where l.SWS = 4)
       except
       (select * from attend)) sno);
```

Null Values

Null Values

Example

student		
matrNr	name	semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3

find the number of students

Null Values

Example

find the number of students:

Null Values

Example

find the number of students:

```
select count (*) from student;
```

Null Values

Example

find the number of students:

```
select count (*) from student;
```

```
select count (matrNr) from student;
```

Null Values

Example

find the number of students:

```
select count (*) from student;
```

```
select count (matrNr) from student;
```

```
select count (semester) from student;
```

Null Values

Example

find the number of students:

```
select count (*) from student;
```

```
select count (matrNr) from student;
```

```
select count (semester) from student;
```

```
select count (*) from student
where semester < 13 or semester >= 13;
```

Null Values

Example

find the number of students:

```
select count (*) from student;                    (6)

select count (matrNr) from student;               (6)

select count (semester) from student;             (6)

select count (*) from student
where semester < 13 or semester >= 13;            (6)
```

Null Values

we get null values

- when there is no value stored in the data base,
- in the course of query evaluation (ex. outer joins)

Null Values

we get null values

- when there is no value stored in the data base,
- in the course of query evaluation (ex. outer joins)

Example

student		
matrNr	name	semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	NULL
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3

find the number of students

Null Values

Example

find the number of students:

```
select count (*) from student;
```

```
select count (matrNr) from student;
```

```
select count (semester) from student;
```

```
select count (*) from student
where semester < 13 or semester >= 13;
```

Null Values

Example

find the number of students:

```
select count (*) from student;                                    (6)
```

```
select count (matrNr) from student;
```

```
select count (semester) from student;
```

```
select count (*) from student
where semester < 13 or semester >= 13;
```

Null Values

Example

find the number of students:

```
select count (*) from student;                    (6)
```

```
select count (matrNr) from student;               (6)
```

```
select count (semester) from student;
```

```
select count (*) from student
where semester < 13 or semester >= 13;
```

Null Values

Example

find the number of students:

```
select count (*) from student;                    (6)

select count (matrNr) from student;               (6)

select count (semester) from student;             (5)

select count (*) from student
where semester < 13 or semester >= 13;
```

Null Values

Example

find the number of students:

```
select count (*) from student;                    (6)

select count (matrNr) from student;               (6)

select count (semester) from student;             (5)

select count (*) from student
where semester < 13 or semester >= 13;            (5)
```

`count` Revisited

behaviour of `count()`:

- `count(*)`:
  counts all tuples (including duplicates)

`count` Revisited

behaviour of `count()`:

- `count(*)`:
  counts all tuples (including duplicates)
- `count(attributes)`:
  counts the number of values different from NULL in the given column (including duplicates)

`count` Revisited

behaviour of `count()`:

- `count(*)`:
  counts all tuples (including duplicates)
- `count(attributes)`:
  counts the number of values different from NULL in the given column (including duplicates)
- `count(distinct attributes)`:
  counts the number of distinct values different from NULL in the given column (i.e. without duplicates)

Null Values

Example

in case there are tuples where the value for semester is unknown we have:

```
select count(*)
from student
where semester < 13 or semester >= 13;
```

$$\neq$$

```
select count(*) from student;
```

Null Values: Handling in Expressions

arithmetic expressions: null values are propagated: if an operand is null then the result is null

Example

null + 1 = null; null * 0 = null; . . .

Null Values: Handling in Expressions

arithmetic expressions: null values are propagated: if an operand is null then the result is null

Example

null + 1 = null; null * 0 = null; ...

comparison operators: SQL has a three-valued logic: true, false, unknown. The result is unknown if at least one of the arguments is null.

Example

(semester > 13) results in unknown if the semester is unknown, i.e. takes the value null.

attention! for example, also (semester = null)

Null Values – Evaluation of Logical Expressions

logical expressions: are based on the following tables:

not	
true	false
unknown	unknown
false	true

Null Values – Evaluation of Logical Expressions

logical expressions: are based on the following tables:

not	
true	false
unknown	unknown
false	true

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

Null Values – Evaluation of Logical Expressions

logical expressions: are based on the following tables:

not	
true	false
unknown	unknown
false	true

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

Null Values – Evaluation

where-condition: only those tuples are passed on for which the condition evaluates to true. Tuples for which the condition evaluates to unknown are not passed to the result.

Null Values – Evaluation

where-condition: only those tuples are passed on for which the condition evaluates to true. Tuples for which the condition evaluates to unknown are not passed to the result.

grouping: null is interpreted as a value and classified as an own group

null values are queried using: `is null` resp. `is not null`.

Example

```
select *
from Student
where semester is null;
```

Special Language Constructs

More Language Constructs

- `between`
- `like`
- `case`
- Joins
  - `cross join`
  - `natural join`
  - `join`
  - `left`, `right`, `full outer join`

between

Example

find all students studying between the first and the fourth semester

```
select * from student
where semester >= 1 and semester <= 4;
```

```
select * from student
where semester between 1 and 4;
```

```
select * from student
where semester in (1,2,3,4);
```

like

comparison of strings: place holder '%' and '_'

- %: arbitrarily many characters (also none)
- _: exactly one character

like

comparison of strings: place holder '%' and '_'

- %: arbitrarily many characters (also none)
- _: exactly one character

Example

find the matrNr of Theophrastos, where you do not know whether the name is spelled with an 'h':

```
select * from student
where name like 'T%eophrastos';
```

like

comparison of strings: place holder '%' and '_'

- %: arbitrarily many characters (also none)
- _: exactly one character

Example

find the matrNr of Theophrastos, where you do not know whether the name is spelled with an 'h':

```
select * from student
where name like 'T%eophrastos';
```

find the matrNr of all students that have attended at least one lecture about ethics:

```
select distinct matrNr
from lecture v, attend h
where h.lecNr = v.lecNr and
      v.title like '%thics%';
```

case

Example

translate grades into values for the study department:

```
select matrNr, (case when grade < 1.5 then 'S1'
                     when grade < 2.5 then 'U2'
                     when grade < 3.5 then 'B3'
                     when grade < 4.0 then 'G4'
                     else 'N5' end)
from examine;
```

case

Example

translate grades into values for the study department:

```
select matrNr, (case when grade < 1.5 then 'S1'
                     when grade < 2.5 then 'U2'
                     when grade < 3.5 then 'B3'
                     when grade < 4.0 then 'G4'
                     else 'N5' end)
from examine;
```

the first qualifying `when`-clause will be executed

Joins

SQL supports the following join keywords:

`cross join`	(cross product)
`natural join`	
`[inner] join`	(theta-join)
`(left|right|full) outer join`	

Joins

SQL supports the following join keywords:

```
                    cross join    (cross product)
                  natural join
                  [inner] join    (theta-join)
(left|right|full) outer join
```

`cross join`:

$$R_1 \times R_2$$

```
select * from professor, lecture;
```

```
select * from professor cross join lecture;
```

`natural join`:

$$R_1 \bowtie R_2$$

the values of columns with same attribute names are set to equal

natural join:

$$R_1 \bowtie R_2$$

the values of columns with same attribute names are set to equal

Example

find name and matrNr of students and the titles of lectures attended by them

```
student(matrNr, name, sem)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

natural join:

Example

find name and matrNr of students and the titles of lectures attended by them

```
student(matrNr, name, sem)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

natural join:

Example

find name and matrNr of students and the titles of lectures attended by them

```
student(matrNr, name, sem)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

```
select student.matrNr, name, title
from student, attend, lecture
where student.matrNr=attend.matrNr and
      attend.lecNr=lecture.lecNr;
```

natural join:

Example

find name and matrNr of students and the titles of lectures attended by them

```
student(matrNr, name, sem)
attend(matrNr, lecNr)
lecture(lecNr, title, SWS, givenBy)
```

```
select student.matrNr, name, title
from student, attend, lecture
where student.matrNr=attend.matrNr and
      attend.lecNr=lecture.lecNr;
```

```
select matrNr, name, title
from student natural join attend
             natural join lecture;
```

[inner] join:

$$R_1 \bowtie_\theta R_2$$

[inner] join:

$$R_1 \bowtie_\theta R_2$$

Example

find professors that teach the lecture Mäeutik

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)

select name, title
from professor, lecture
where persNr = givenBy and
      title = 'Mäeutik';
```

[inner] join:

$$R_1 \bowtie_\theta R_2$$

Example

find professors that teach the lecture Mäeutik

```
professor(persNr, name, rank, room)
lecture(lecNr, title, SWS, givenBy)

select name, title
from professor, lecture
where persNr = givenBy and
      title = 'Mäeutik';
```

```
select name, title
from professor join
     lecture on persNr=givenBy
where title='Mäeutik';
```

(left|right|full) outer join:

$$R_1 (\mathbb{X}| \mathbb{X} |\mathbb{X}) R_2$$

(left|right|full) outer join:

$$R_1 (\mathbb{X}| \mathbb{X} |\mathbb{X}) R_2$$

Example

find all students and their grades obtained at corresponding exams of lectures

(left|right|full) outer join:

$$R_1 (\bowtie | \ltimes | \rtimes) R_2$$

Example

find all students and their grades obtained at corresponding exams of lectures

```
select s.matrNr, s.name, e.lecNr, e.grade
from student s left outer join examine e
                    on s.matrNr=e.matrNr;
```

(left|right|full) outer join:

$$R_1 (\bowtie | \ltimes | \rtimes) R_2$$

Example

find all students and their grades obtained at corresponding exams of lectures

```
select s.matrNr, s.name, e.lecNr, e.grade
from student s left outer join examine e
                    on s.matrNr=e.matrNr;
```

```
select s.matrNr, s.name, e.lecNr, e.grade
from examine e right outer join student s
                  on s.matrNr=e.matrNr;
```

student ⟗ *examine*

matrNr	name	lecNr	grade
24002	Xenokrates		
25403	Jonas	5041	2
26120	Fichte		
26830	Aristoxenos		
27550	Schopenhauer	4630	2
28106	Carnap	5001	1
29120	Theophrastos		
29555	Feuerbach		

student ⟗ *examine*			
matrNr	name	lecNr	grade
24002	Xenokrates		
25403	Jonas	5041	2
26120	Fichte		
26830	Aristoxenos		
27550	Schopenhauer	4630	2
28106	Carnap	5001	1
29120	Theophrastos		
29555	Feuerbach		

result without using outer join

student ⋈ *examine*			
matrNr	name	lecNr	grade
25403	Jonas	5041	2
27550	Schopenhauer	4630	2
28106	Carnap	5001	1

coalesce()

Example

find all students and their grades obtained at corresponding exams of lectures; for students that have not attended any lectures set lecNr and grade to the value 0

coalesce()

Example

find all students and their grades obtained at corresponding exams of lectures; for students that have not attended any lectures set lecNr and grade to the value 0

```
select s.matrNr, s.name,
       coalesce(e.lecNr,0),
       coalesce(e.grade,0)
from student s left outer join examine e
                    on s.matrNr=e.matrNr;
```

attention: types have to be compatible

concat()

Example

```
student(matrNr, name, surname, semester);
```

concat()

Example

```
student(matrNr, name, surname, semester);
```

find the names of all students

concat()

Example

```
student(matrNr, name, surname, semester);
```

find the names of all students

```
select name || surname student;
```

or

concat()

Example

```
student(matrNr, name, surname, semester);
```

find the names of all students

```
select name || surname student;
```

or

```
select concat(name, surname) from student;
```

(sometimes "+" instead of "||")

Overview

1. SQL then and now
2. Data Query Language
3. Data Definition Language
4. Data Manipulation Language

Data Definition Language

data types: constructs for strings, digits, dates, ...

```
character(n),           char(n)
character varying(n), varchar(n)

numeric(p,s)
integer, int

date

blob, raw     %for big binary resp. text data
clob          %for big text data
```

Data Definition Language

schema definition and modification

```
create table professor
   (persNr   integer primary key,
    name     varchar(10) not null,
    rank     character(2));
```

Data Definition Language

schema definition and modification

```
create table professor
    (persNr   integer primary key,
     name     varchar(10) not null,
     rank     character(2));
```

```
create table presuppose
    (predNr  integer,
     sucNr integer,
     primary key (predNr, sucNr)
    );
```

Data Definition Language

schema definition and modification

```
drop table professor;
```

Data Definition Language

schema definition and modification

```
drop table professor;
```

```
alter table professor rename to lecturer;

alter table professor add room integer;

alter table professor drop room;

alter table professor alter
                          name type varchar(30);
```

Data Definition Language

schema definition and modification

```
drop table professor;
```

```
alter table professor rename to lecturer;

alter table professor add room integer;

alter table professor drop room;

alter table professor alter
                          name type varchar(30);
```

```
alter table professor alter
          column name set data type varchar(30);
```

Data Manipulation Language – Insert Tuples

insert tuples in a constructed table

```
professor(persNr:integer, name:varchar(30),
          rank:character(2), room:integer)
```

Data Manipulation Language – Insert Tuples

insert tuples in a constructed table

```
professor(persNr:integer, name:varchar(30),
          rank:character(2), room:integer)
```

Example

insert professor Curie:

```
insert into professor
values(2136, 'Curie', 'C4', 36);
```

Data Manipulation Language – Insert Tuples

insert tuples in a constructed table

```
professor(persNr:integer, name:varchar(30),
          rank:character(2), room:integer)
```

Example

insert professor Curie:

```
insert into professor
values(2136, 'Curie', 'C4', 36);
```

insert several professors:

```
insert into professor
values (2125, 'Sokrates', 'C4', 226),
       (2126, 'Russel', 'C4', 232);
```

Data Manipulation Language – Insert Tuples

insert the result of a query into a table

Example

insert all students to the lecture 'Logik':

```
attend (matrNr, lecNr)
student (matrNr, name, sem)
lecture (lecNr, title, SWS, persNr)
```

```
insert into attend
```

Data Manipulation Language – Insert Tuples

insert the result of a query into a table

Example

insert all students to the lecture 'Logik':

```
attend (matrNr, lecNr)
student (matrNr, name, sem)
lecture (lecNr, title, SWS, persNr)
```

```
insert into attend
select matrNr, lecNr
from student, lecture
where title='Logik';
```

Data Manipulation Language – Delete Tuples

list the tuples to be deleted:

Example

delete Kant from the professor table:

```
delete from professor
values (2137, 'Kant', 'C4', 7);
```

delete all tuples satisfying a condition:

Example

delete Kant from the professor table:

```
delete from professor where persNr=2137;
```

Data Manipulation Language – Delete Tuples

list the tuples to be deleted:

Example

delete Kant from the professor table:

```
delete from professor
values (2137, 'Kant', 'C4', 7);
```

delete all tuples satisfying a condition:

Example

delete Kant from the professor table:

```
delete from professor where persNr=2137;
```

```
delete from professor where persNr < 2137;
```

Data Manipulation Language – Delete Tuples

Example

delete students attending the lecture 'Logik'

```
attend (matrNr, lecNr)
student (matrNr, name, sem)
lecture (lecNr, title, SWS, persNr)
```

Data Manipulation Language – Delete Tuples

Example

delete students attending the lecture 'Logik'

```
attend (matrNr, lecNr)
student (matrNr, name, sem)
lecture (lecNr, title, SWS, persNr)
```

```
delete from student
where matrNr in (
  select matrNr
  from attend, lecture
  where attend.lecNr = lecture.lecNr and
        title = 'Logik');
```

Data Manipulation Language – Manipulate Tuples

Data Manipulation Language – Manipulate Tuples

modify tuples

Example

increase the semester number of all students by 1:

```
update student
set semester = semester + 1;
```

Example

change the rank of C3 professors with a persNr bigger than 2500 to C2

```
update professor
set rank = 'C2'
where rank = 'C3' and persNr > 2500;
```

Data Manipulation Language – Manipulate Tuples

Data Manipulation Language – Manipulate Tuples

modify tuples

Example

```
professor(persNr:integer, name:varchar(30),
          rank:character(2), room:integer,
          teaching load: integer)
lecture (lecNr, title, SWS, persNr)
```

insert for all professors their corresponding teaching load (=sum of SWS)

Data Manipulation Language – Manipulate Tuples

modify tuples

Example

```
professor(persNr:integer, name:varchar(30),
          rank:character(2), room:integer,
          teaching load: integer)
lecture (lecNr, title, SWS, persNr)
```

insert for all professors their corresponding teaching load (=sum of SWS)

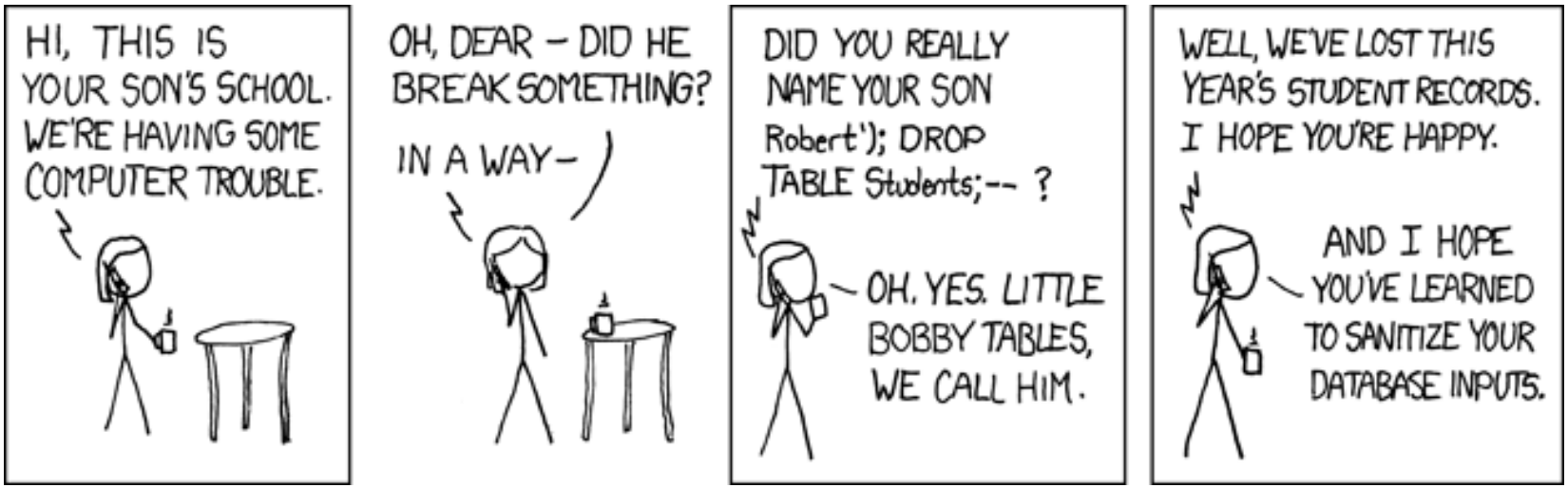
```
update professor
set teaching load = (
  select sum(SWS)
  from lecture v
  where v.persNr = professor.persNr)
```

(remark: this is not how the teaching load should be stored)

Learning Objectives

- simple SQL queries
- queries over several relations
- set operations
- aggregate operations
- aggregate and grouping
- nested queries
- existentially quantified queries
- universally quantified queries
- null values
- special language constructs

Security – SQL Injections



(c) xkcd.com