Generated Values

Artificial values, surrogates, no semantic, mostly as keys:

- Directly in the table definition:

```
create table dept (
      deptno   serial primary key,
      deptname varchar(50) not null);
```

insert with:

```
insert into dept values(default, 'I3')or
       insert into dept values('I3');
```

Sequences to share

CREATE [TEMPORARY | TEMP] SEQUENCE name [INCREMENT [BY] increment]

[MINVALUE minvalue | NO MINVALUE] [MAXVALUE maxvalue | NO MAXVALUE]

[START [WITH] start] [CACHE cache]

[[NO] CYCLE]

```
CREATE SEQUENCE artificial_key START 101;
```

```
CREATE TABLE Dept (deptno INT DEFAULT nextval('artificial_key') NOT NULL,...)
```

```
INSERT INTO dept VALUES (nextval('artificial_key'), 'I3');
```

Relational Algebra

σ Selection

π Pojection

X Cartesian Product

⋈ Join

ρ Renaming

⋊ Semi-Join (left)

⋉ Semi-Join (right)

⟕ left outer Join

⟖ right outer Join

General Set Operations:

– (set-theoretic) Difference (Complement)

÷ Division

∪ Union

∩ Intersection

Example Set Intersection

Find the *PersNr* of all C4-Professors, who give at least one lecture.

$$\Pi_{\text{PersNr}}(\rho_{\text{PersNr}\leftarrow\text{Given\_by}}(\text{Lectures})) \cap \Pi_{\text{PersNr}}(\sigma_{\text{Level=C4}}(\text{Professors}))$$

→ procedural !

Relational Tuple Calculus

A query in the relational calculus is of the form

$$\{t \mid P(t)\}$$

with t Tuple variable and P predicate

Simple **example:**

C4-Professors

$$\{p \mid p \in \text{Professors} \wedge p.\text{Level} = \text{'C4'}\}$$

Relational Tuple Calculus: further example

Students who attend at least one lecture of Curie

$$
\begin{aligned}
\{s \mid\ & s \in \text{Students} \\
& \wedge \exists h \in \text{attend}(s.\text{StudNr}=h.\text{StudNr} \\
& \wedge \exists v \in \text{Lectures}(h.\text{LectureNr}=v.\text{LectureNr} \\
& \wedge \exists p \in \text{Professors}(p.\ \text{PersNr}=v.\text{Given\_by} \\
& \wedge p.\text{Name} = \text{'Curie'})))\}
\end{aligned}
$$

The same query in SQL …
… shows the relation

```
select s.*
from Students s
where exists (
        select h.*
        from attend h
        where h.StudNr = s.StudNr and exists (
              select *
              from Lectures v
              where v.LectureNr = h.LectureNr and exists (
                       select *
                        from Professors p
                        where p.Name =,Curie' and
                                                p.PersNr= v.Given_by )))
```

Relational Domain Calculus

Query in the domain calculus is of the form:

$\{[v1, v2, \ldots, vn] \mid P(v1, \ldots, vn)\}$

with v1, . . . , v2 domain variables and P predicate

Example:

StudNr and Name of the testees of Sokrates

$$\{[m, n] \mid \exists\ ([m, n, s] \in Students$$
$$\wedge \exists\ p, v, g\ ([m, p, v, g] \in test$$
$$\wedge \exists\ a, r, b\ ([p, a, r, b] \in Professors$$
$$\wedge\ a = \text{‘Sokrates’}))))\}$$

Expressive Power

The three languages

- relational Algebra
- Tuple Relational Calculus, restricted to safe expressions
- Domain Relational Calculus, restricted to safe expressions

are equal in their expressive power

$\{n \mid \neg(n \in \mathit{Professors})\}$ *e.g. is* not safe, as the result is infinite

SQL - DRL

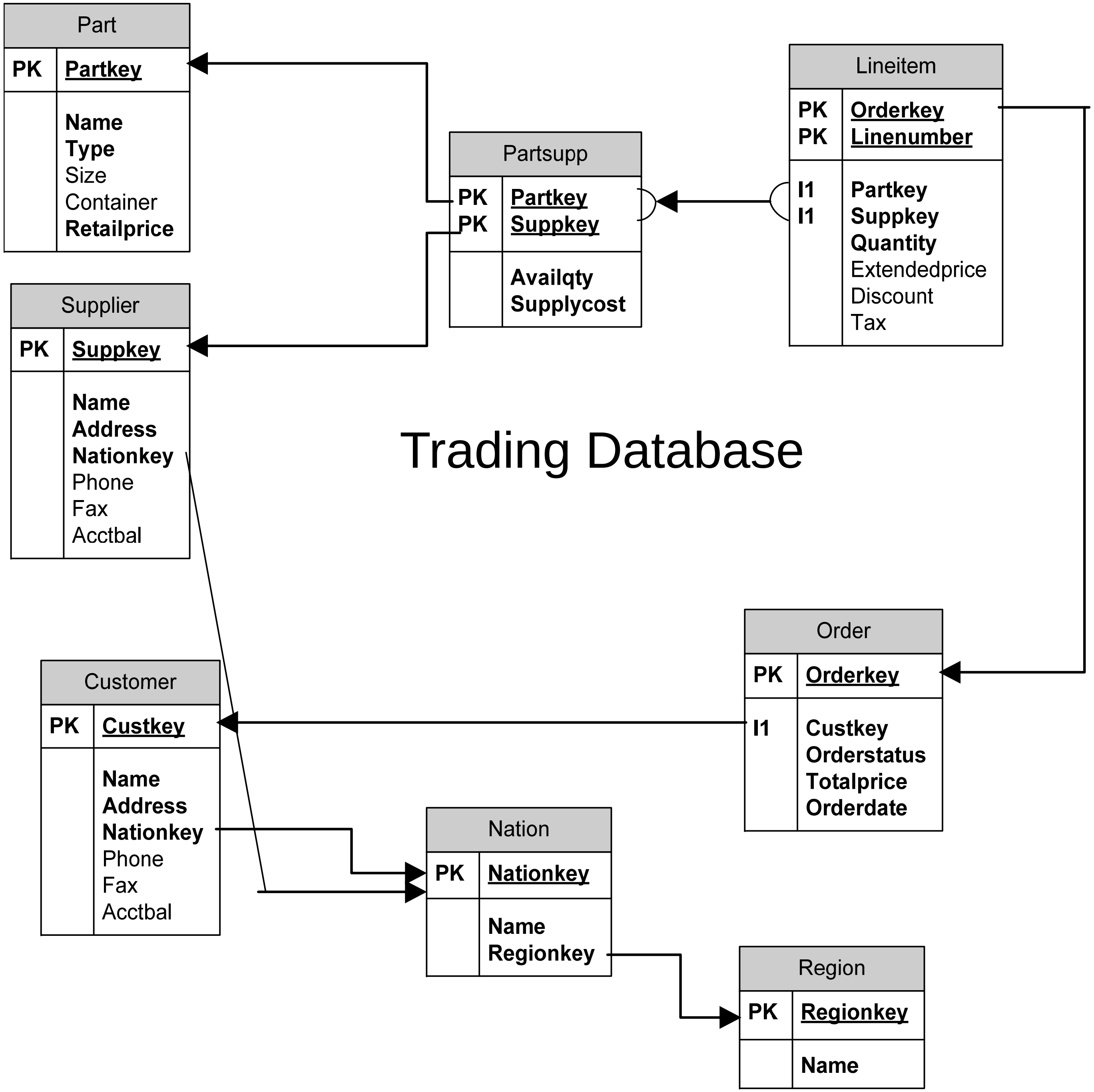
Tutorials for first insights into SQL:

- sql.lernenhoch2.de/lernen/ (German)
- www.w3schools.com/sql

Web interfaces for SQL:

- sqlfiddle.com (MySQL, Oracle, PostgreSQL, SQLite, MS SQL):
  also possible to create tables
- hyper-db.com/interface.html (HyPer):
  University Scema, TPC-H Schema
  Query Execution Plans

Trading Database



Part	
PK	**Partkey**
	Name **Type** Size Container **Retailprice**

Supplier	
PK	**Suppkey**
	Name **Address** **Nationkey** Phone Fax Acctbal

Partsupp	
PK **PK**	**Partkey** **Suppkey**
	Availqty **Supplycost**

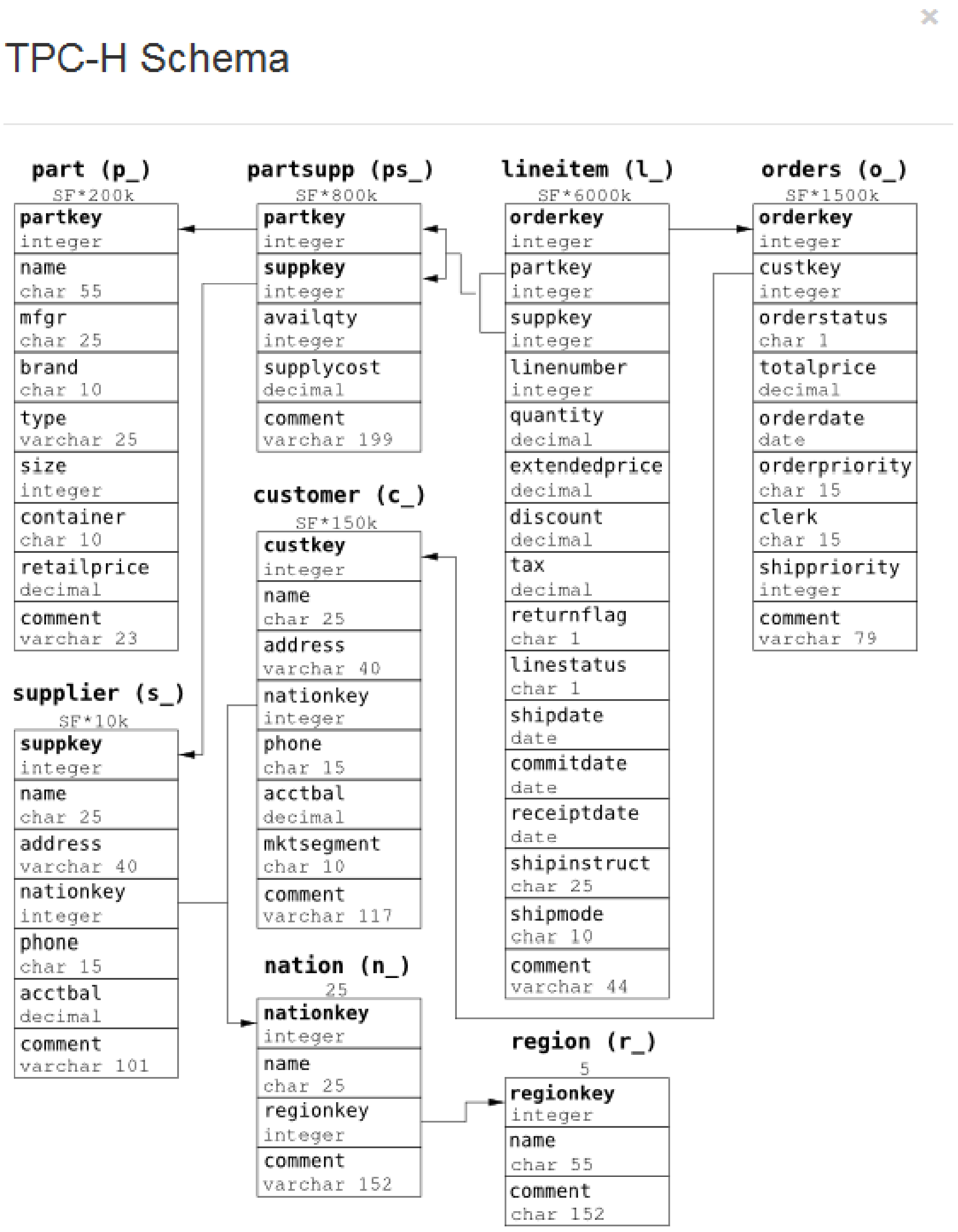
Lineitem	
PK **PK**	**Orderkey** **Linenumber**
I1 **I1**	**Partkey** **Suppkey** **Quantity** Extendedprice Discount Tax

Order	
PK	**Orderkey**
I1	**Custkey** **Orderstatus** **Totalprice** **Orderdate**

Customer	
PK	**Custkey**
	Name **Address** **Nationkey** Phone Fax Acctbal

Nation	
PK	**Nationkey**
	Name **Regionkey**

Region	
PK	**Regionkey**
	Name



TPC-H Schema

part (p_)
SF*200k

partkey integer
name char 55
mfgr char 25
brand char 10
type varchar 25
size integer
container char 10
retailprice decimal
comment varchar 23

supplier (s_)
SF*10k

suppkey integer
name char 25
address varchar 40
nationkey integer
phone char 15
acctbal decimal
comment varchar 101

partsupp (ps_)
SF*800k

partkey integer
suppkey integer
availqty integer
supplycost decimal
comment varchar 199

customer (c_)
SF*150k

custkey integer
name char 25
address varchar 40
nationkey integer
phone char 15
acctbal decimal
mktsegment char 10
comment varchar 117

nation (n_)
25

nationkey integer
name char 25
regionkey integer
comment varchar 152

lineitem (l_)
SF*6000k

orderkey integer
partkey integer
suppkey integer
linenumber integer
quantity decimal
extendedprice decimal
discount decimal
tax decimal
returnflag char 1
linestatus char 1
shipdate date
commitdate date
receiptdate date
shipinstruct char 25
shipmode char 10
comment varchar 44

region (r_)
5

regionkey integer
name char 55
comment char 152

orders (o_)
SF*1500k

orderkey integer
custkey integer
orderstatus char 1
totalprice decimal
orderdate date
orderpriority char 15
clerk char 15
shippriority integer
comment varchar 79

Skeleton SQL Query

select	<Attribute_list>	5
from	<Relation_list>	1
[where	<Predicate_list>	2
group by	<Attribute_list>	3
having	<Predicate_list>	4
order by	<Attribute_list>	6
fetch first	<Number Result Tuples> **]**	7

Simple example

Query:
"Give complete information of all Professors„

select *

from Professors

Professors

PersNr	Name	Level
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Result

PersNr	Name	Level
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Selection of attributes

Query:
"Give PersNr and name of all professors„

```
select   PersNr, Name
from     Professors
```

Professors

PersNr	Name	Level
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Result

PersNr	Name
2136	Curie
2137	Kant
2126	Russel
2125	Sokrates
2134	Augustinus
2127	Kopernikus
2133	Popper

Duplicate elimination

- Contrary to the relational algebra (sets!) SQL does not eliminate duplicates

- If you want duplicate elimination, the key word **distinct** has to be used

- Example:

query: „Which levels professors have?„

```
select distinct Level
from    Professors
```

Result:

Level
C3
C4

Where clause: Select Tuples

Query:
"Give PersNr and name of all professors, who have the level C4„

```
select   PersNr, Name
from     Professors
where    Level= ´C4´;
```

Result:

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Where Clause: Predicates

- Predicates in the where clause can be combined logically with:

AND, OR, NOT

- Comparison operators can be:

=, <,<=, >,>=, between, like

Example für between

query:

"Give the name of all students who were born between 1987-01-01 and 1989-01-01„

select Name
from Students
where birthday **between** 1987-01-01 **and** 1989-01-01**;**

query equivalent to:

select Name
from Students
where birthday **>=** 1987-01-01
and birthday**<=** 1989-01-01***;***

String comparisons

- String constants have to be included in single quotation marks

query:
"Give all information about the professor whose name is Kant„

```
select *
from    Professors
where Name = ’Kant’;
```

Search with wildcards

query:
"Give all information about professors, whose name starts with a K“

select *
from Professors
where Name **like** ’K%’;

Possible wildcards:

- _ arbitrary character
- % arbitrary string (maybe also of length 0)

Null values

- In SQL there is a special value **NULL**
- This value exists for all data types and represents values which are
  - unknown or
  - *not available* or
  - *not applicable*
- Null values can also emerge from query evalaution

- Test for NULL → **is NULL**

Example:

select *

from Professors

where Room **is NULL;**

Null values cont.

- Null values are passed through in arithmetic expressions : at least one operand NULL → result is NULL as well
- Sometimes surprising query results, if Null values occur, e.g.:

```
select count (*)
from Students
where Semester < 13 or Semester > = 13
```

- If there are students whose attribute value semester is a NULL value these are not counted
- The reason is three-valued logic with inclusion of NULL values

Evaluation with Null values

- SQL: three-valued logic with the values **true**, **false** und **unknown**
- **unknown** is result of comparisons if at least one of the arguments is NULL
- In a **where** clause only tuples are passed through for which the predicate is **true**. In particular tuples for which the predicate is **unknown** do not contribute to the result.
- In groupings NULL is a separate value and classified as an own group.
- Logical expressions are computed according to the following tables:

Three valued logic 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

Professors

PersNr	Name	Level	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

attend

StudNr	LectureNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
25403	5022
29555	5022
29555	5001

Students

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

require

Predecessor	Successor
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

test

StudNr	LectureNr	PersNr	Grade
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Lectures

Lecture Nr	Title	Weekly Hours	Given_by
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

Assistants

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

Queries with several relations: Cartesian product

- If several relations are listed in the from clause they are combined with a cartesian product

- Example:

query: "Give all professors and their lectures„

```
select *
from   Vorlesung, Professor;
```

Result???

Queries with several Relations: Joins

- Cartesian products usually do not make sense, more interesting are Joins
- Join predicates are given in the where clause

```
select *
from Lectures, Professors
where Given_by = PersNr;
```

Queries with several Relations: Joins cont.

Which professor gives "Mäeutik"?

```
select Name, Title
from Professors, Lectures
where PersNr = Given_by
        and Title = ‘Mäeutik‘;
```

Example

Professors			
PersNr	Name	Level	Room
2125	Sokrates	C4	226
2126	Russel	C4	232
⋮	⋮	⋮	⋮
2137	Kant	C4	7

Lectures			
LectureNr	Title	WeeklyHours	Given_by
5001	Grundzüge	4	2137
5041	Ethik	4	2125
⋮	⋮	⋮	⋮
5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮
4630	Die 3 Kritiken	4	2137

PersNr	Name	Level	Room	LectureNr	Title	WeeklyHours	Given_by
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮		⋮	⋮	⋮	⋮	⋮
2126	Russel	C4	232	5001	Grundzüge	4	2137
2126	Russel	C4	232	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die 3 Kritiken	4	2137

↓ Selection

PersNr	Name	Level	Room	LectureNr	Title	WeeklyHours	Given_by
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

↓ Projection

Name	Title
Sokrates	Mäeutik

Name collision

- Attributes with the same names have to be identified uniquely in the corresponding relations

Example:
Which students attend which lectures?

select Name, Title

from Students, attend, Lectures

where Students.StudNr = attend.StudNr **and**
  attend.LectureNr = Lectures.LectureNr;

Name collision cont.

Which students attend which lectures?

Alternative:

```
select s.Name, l.Title
from Students s, attend a, Lectures l
where s.StudNr = a.StudNr and
    a.LectureNr = l.LectureNr
```

Set operations

- In SQL you also have the common operations on sets:
  union, intersection, and (set-theoretic) difference
- Require – like in the relational algebra – the same schema of the resulting relations

```
( select Name
 from Assistants )
union
( select Name
 from Professors);
```

Duplicate elimination

- In contrary to **select** the **union** operator automatically eliminates duplicates

- If duplicates are desired in the result the **union all** operator has to be used

Intersection , Difference

Professors **and** Assistants

```
select Name  from Professors
intersect
select Name from Assistants;
```

Professors, but **not** Assistants

```
select Name from Professors
except
select Name from Assistants;
```

Sorting

- Tuples in a relation are not (automatically) sorted
- Result of a query can be sorted via the **order by** clause
- It can be sorted ascending or descending
- Default sorting: ascending

Example

```
select *
from Students
order by Name, Semester desc;
```

Nested queries

- Queries can be nested within other queries, i.e. there is more than one select clause
- Nested select can be in the where clause, in the from clause, and even in a select clause itself
- In principal an intermediate result is computed in the "inner" query which is then used in the „outer" one

Select in Where clause

- Two different sorts of subqueries: correlated and uncorrelated
- uncorrelated: subquery only refers to „own" attributes
- correlated: subquery also refers to attributes of the outer query

Professors

PersNr	Name	Level	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

Students

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

Lectures

Lecture Nr	Title	Weekly Hours	Given_by
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

attend

StudNr	LectureNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
25403	5022
29555	5022
29555	5001

require

Predecessor	Successor
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

Assistants

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

test

StudNr	LectureNr	PersNr	Grade
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Uncorrelated subquery

Name of all students, who attend LectureNr 5041

```
select S.Name
from Students S
where S.StudNr in
(select a.StudNr
from attend a
where a.LectureNr = 5041);
```

- subquery is evaluated once
- for every tuple of the outer query is checked whether StudNr is in the result of the subquery

Correlated subquery

Find those professors and their assistants where the assistants work in different areas

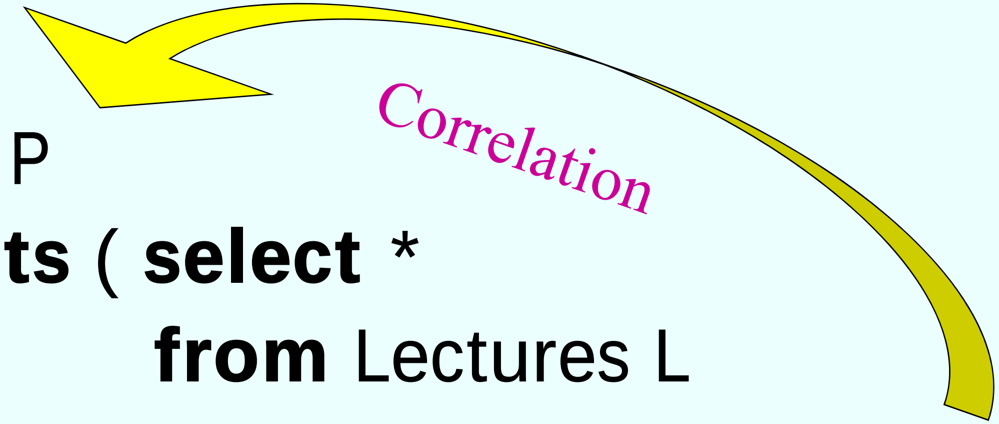
```
select distinct  P.Name
from Professors P, Assistants A
where A.Boss = P.PersNr
and exists
(select *
from Assistent B
where B.Boss = P.PersNr and A.Area <> B.Area);
```

← Correlation

- For every tuple of the outer query the inner query has different values
- The exists-predicate is true, if the subquery contains at least one tuple

Existential Quantification: exists

```
select P.Name
from Professors P
where not exists ( select *
                   from Lectures L
                   where L.Given_by = P.PersNr );
```



Set comparison

Uncorrelated subquery: usually more efficient, evaluated only once

```
select Name
from Professors
where PersNr not in ( select Given_by
                      from Lectures );
```

Uncorrelated versus correlated subqueries

- correlated

```
select s.*
from Students s
where exists
        (select p.*
        from Professors p
        where p.Birthdate > s.Birthdate);
```

Query Rewrite

Equivalent uncorrelated form

```
select s.*
from Students s
where s.Birthdate <
      (select max (p.Birthdate)
       from Professors p);
```

Advantage: result of subquery can be materialized

Subquery has to be evaluated only once

Un-nesting correlated subqueries

```
select a.*
from Assistants a
where exists
    ( select p.*
      from Professors p
      where a.Boss = p.PersNr and p.Birthdate > a.Birthdate);
```

- Un-nesting via join

```
select a.*
from Assistants a, Professors p
where a.Boss=p.PersNr and p.Birthdate > a.Birthdate;
```