Chapter 7: SQL Data Retrieval

Content:

- How do we access data in a database using SQL

Next:

- Aggregation in SQL

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

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 do 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

Query Execution Example

```
select   PersNr, Name
from     Professors
where    Level= ´W3´;
```

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

Where Clause: Predicates

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

AND, OR, NOT

- Comparison operators can be:

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

Example for 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)

Aggregation (simple)

select avg(semester) **from** students;

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

Result
1
7.625

- Aggregats: **avg, max, min, count, sum**

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 evaluation

- 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

Three valued logic tables

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 Quiz Time !

select 1 + null;	→ null
select 1 or null;	→ error (‘1’ not boolean)
select true or null;	→ true
select false or null;	→ null / unknown
select true and null;	→ null / unknown
select false and null;	→ false

Null Quiz Time !

Aggreation on:

Test
a
1
null
2

select *count*(*) from test; → 3

select *count*(a) from test; → 2

select *sum*(a) from test; → 3

select *avg*(a) from test; → 1.5

select *sum*(a)/*count*(a) from test; → 1

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

Queries with several relations: Cartesian product

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

• Example:
Query: ”Who is giving which lecture?„

select *
from Lectures, Professors**;**

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;
```

Join Example

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

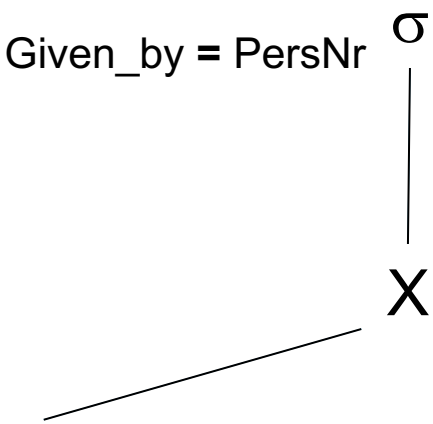
Profs	
PersNr	**Name**
1	Kant
2	Russel

Lectures	
Given_by	**Title**
1	Logic
2	Ethic

Join Example: Result

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

Result			
PersNr	**Name**	**Given_by**	**Title**
1	Kant	1	Logic
2	Russel	2	Ethic



Intermediate Result			
PersNr	**Name**	**Given_by**	**Title**
1	Kant	1	Logic
1	Kant	2	Ethic
2	Russel	1	Logic
2	Russel	2	Ethic

Profs	
PersNr	**Name**
1	Kant
2	Russel

Lectures	
Given_by	**Title**
1	Logic
2	Ethic

Queries with several Relations: Joins cont.

Which professor gives "Mäeutik"?

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 / Filtering

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;
```

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

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
- They 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

Who is a Professors **and** Assistants

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

Who is a 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;
```

Example

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

1. Sort by name in an ascending order
2. If two names are the same: sort them by semester in descending order.

Nested queries

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

```
select s.Name
from Students s
where s.semester = (select max(s2.semester)
                    from Students s2);
```

Nested queries

- 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

Nested queries

- The result of a subquery can be used like any other expression (if it is guaranteed a single value)
- In addition there are two new operators:
  - “in”: Checks if the value is equals to any of the rows in the subquery
  - “exists”: Checks if the subquery returns at least one row
  - “all”: Checks if the condition holds for all values in the subquery

Uncorrelated Subquery

Students in the highest semester:

```
select s.Name
from Students s
where s.semester = (select max(s2.semester)
                    from Students s2);
```

- Subquery is evaluated once
- For every tuple of the outer query it is checked whether the students semester is equal to the result of the subquery
- The subquery needs to produce **exactly** one row and one column (→ a single value: one integer in this case)

Uncorrelated Subquery with “in”

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 it is checked whether the studNr matches any of the studNrs produced by the subquery

Uncorrelated Subquery with “all”

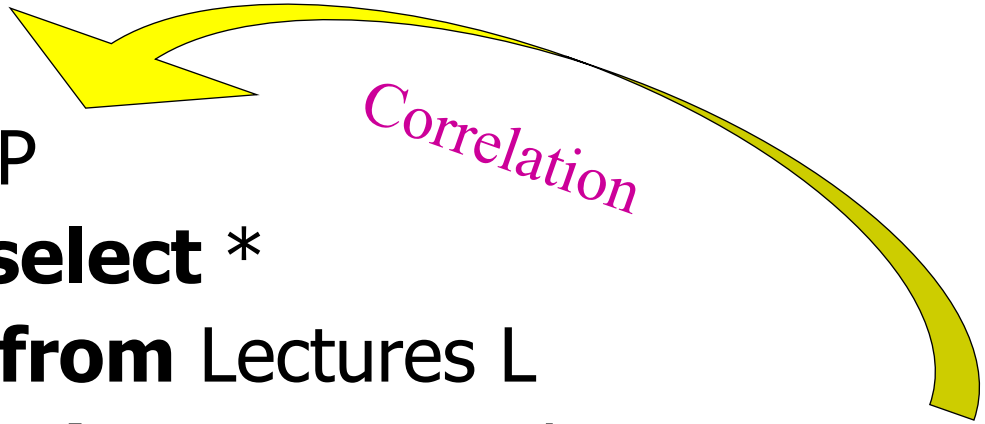
Name of all students, who attend LectureNr 5041

```
select S.Name
from Students S
where S.Semester >= all ( select Semester
                          from Students );
```

- Subquery is evaluated once
- For every tuple of the outer query it is checked whether the semester is larger than all of the semesters produced by the subquery

Correlated Subquery with ”exists”

```
select P.Name
from Professors P
where exists ( select *
               from Lectures L
               where L.Given_by = P.PersNr );
```



Profs	
PersNr	**Name**
1	Kant
2	Russel

Lectures	
Given_by	**Title**
1	Logic
1	Ethic

Result
Name
Kant

Set comparison

Uncorrelated subquery: usually more efficient, evaluated only once

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

→ We can use **not** to negate the result of an **in** or **exists** expression.

Correlated subquery

Find those professors that have at least two assistants who work on different topics

Correlated subquery

Find those professors that have at least two assistants who work on different topics

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

- 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

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;
```

Exercise: Uncorrelated versus correlated subqueries

- Find students who are older than at least one professor
  - 1. Correlated
  - 2. Uncorrelated

Exercise: Uncorrelated versus correlated subqueries

- 1. Correlated: "Take these students where there exists a professor that is younger."

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

Exercise: Uncorrelated versus correlated subqueries

- 2. Uncorrelated: "Take all students that are older than the youngest professor."

```
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