Aggregate Functions (III)

- Each aggregate function (except **count**) eliminates null values first from the collection and operates only on the remaining non-null values

- The term **count**(*) indicates a special use of **count**; it counts all the rows of a table, regardless of whether null values or duplicate tuples occur

- The keyword **distinct**
  - placed in front of the column name in the aggregate function is used to eliminate duplicates before the aggregate is computed
  - has only an effect on the result of the functions **count**, **avg**, and **sum**

- Aggregate functions can *only* be used in the **select** clause and the **having** clause (see the later discussion of grouping)

Aggregate Functions (IV)

- Query 1: Calculate the number of students.

  select count(*) **from** students;

- Query 2: Calculate the number of students, and store the result in a new attribute named "total".

  select count(*) **as** total **from** students;

- Query 3: Calculate the number of students with different names.

  select count(**distinct** name) **from** students;

- Query 4: Calculate the number of different semesters the students are in, and store the result in a new attribute named "diff_sem".

  select count(**distinct** sem) **as** diff_sem **from** students;

8

Query 1

total
8

Query 2

8

Query 3

diff_sem
7

Query 4

Aggregate Functions (V)

- Query 5: Calculate the number of students in the second or eight semester.

  select count(*) **from** students **where** sem = 2 **or** sem = 8;

- Query 6: Calculate the average semester of all students, and store the result in a new attribute named “avg_sem”.

  select avg(sem) **as** avg_sem **from** students;

- Query 7: Compute the average grade for all performed tests so far.

  select avg(grade) **from** tests;

- Query 8: Calculate the total of credits for all lectures, and store the result in a new attribute named “total_cred”.

  select sum(credits) **as** total_cred **from** lectures;

3

Query 5

avg_sem
7.625

Query 6

1.66

Query 7

total_cred
30

Query 8

Aggregate Functions (VI)

- Query 9: Determine the lowest room number of assistants.

```
select min(room) from assistants;
```

- Query 10: Determine the highest semester number of a student, and store the result in a new attribute named “max_sem”.

```
select max(sem) as max_sem from students;
```

- Query 11: Compute the largest difference of student semester numbers.

```
select max(sem) – min(sem) from students;
```

- Query 12: Determine the lowest grade (new attribute “low_grade”) and the highest grade (new attribute “high_grade”) of tests.

```
select max(grade) as low_grade, min(grade) as high_grade from tests;
```

101

Query 9

max_sem
18

Query 10

16

Query 11

low_grade	high_grade
2	1

Query 12

Grouping (I)

- Aggregation so far relates to the set of values of a whole column
- Sometimes it is helpful to first partition the values of a column with respect to equality into groups and then apply an aggregation function to each group
- Query 13: We assume that the *lectures* schema is extended by the attribute *hpw* (hours per week). Determine the number of hours per week in which each professor has given lectures.

lectures				
id	title	credits	held_by	hpw
5001	foundations	4	2137	3
5041	ethics	4	2125	4
5043	epistemology	3	2126	4
5049	maieutics	2	2125	2
4052	logic	4	2125	4
5052	philosophy of science	3	2126	3
5216	bioethics	2	2126	2
5259	The Vienna Circle	2	2133	3
5022	faith and knowledge	2	2134	2
4630	The 3 Cutups	4	2137	4

Original table *lectures*

lectures				
id	title	credits	held_by	hpw
5041	ethics	4	2125	4
5049	maieutics	2	2125	2
4052	logic	4	2125	4
5043	epistemology	3	2126	4
5052	philosophy of science	3	2126	3
5216	bioethics	2	2126	2
5259	The Vienna Circle	2	2133	3
5022	faith and knowledge	2	2134	2
5001	foundations	4	2137	3
4630	The 3 Cutups	4	2137	4

Grouped table *lectures*

Grouping (II)

- Query 13 (*continued*)

```
select held_by, sum(hpw) as number
from lectures
group by held_by;
```

lectures				
id	title	credits	held_by	hpw
5041	ethics	4	2125	4
5049	maieutics	2	2125	2
4052	logic	4	2125	4
5043	epistemology	3	2126	4
5052	philosophy of science	3	2126	3
5216	bioethics	2	2126	2
5259	The Vienna Circle	2	2133	3
5022	faith and knowledge	2	2134	2
5001	foundations	4	2137	3
4630	The 3 Cutups	4	2137	4

Grouped table *lectures*

held_by	number
2125	10
2126	9
2133	3
2134	2
2137	7

Grouping (III)

- **group by** clause
  - A “group-by-expression” is an expression that only contains attributes (called grouping attributes) that are not used for computing the aggregate (that is, as aggregation attributes)
  - Tuples with equal grouping attribute values are summarized in groups
  - For each group the query produces a new tuple in the result relation
  - Only attributes with one value per group are permitted after the **select** clause

- Query 14: Determine the number of hours per week of those lectures held by professors who predominantly give long lectures (> 3 hours per week on average).

```
select held_by, sum(hpw) as number
from lectures
group by held_by
having avg(hpw) > 3;
```

Grouping (IV)

lectures				
id	title	credits	held_by	hpw
5041	ethics	4	2125	4
5049	maieutics	2	2125	2
4052	logic	4	2125	4
5043	epistemology	3	2126	4
5052	philosophy of science	3	2126	3
5216	bioethics	2	2126	2
5259	The Vienna Circle	2	2133	3
5022	faith and knowledge	2	2134	2
5001	foundations	4	2137	3
4630	The 3 Cutups	4	2137	4

Grouped table *lectures*

held_by	avg(hpw) > 3	number
2125	10 / 3 > 3 yes	10
2126	9 / 3 > 3 no	
2133	3 / 1 > 3 no	
2134	2 / 1 > 3 no	
2137	7 / 2 > 3 yes	7

Result tuples in dark gray

- **having** clause
  - Filtering *groups* (instead of tuples) with respect to a condition that may only contain arguments with one value per group

Grouping (V)

- **having** clause (*continued*)
  - Attributes used in this clause must also be grouping attributes or be contained in an aggregate function in the **select** clause
  - Includes at least one aggregate function
- Query 15: Determine the number of hours per week of those lectures held by C4 professors who predominantly give long lectures (> 3 hours per week on average). Also output the names of the respective professors.

```
select held_by, name, sum(hpw) as number
from lectures, professors
where held_by = pers-id and rank = ‘C4’
group by held_by, name
having avg(hpw) > 3;
```

- All attributes in the **select** clause that are not aggregation attributes must be listed as grouping attributes in the **group by** clause

General Form of an SQL Query

- General form

```
select [distinct] {* | <column expression> [as <new column name>] [, …]}
from <table name> [[as] <variable>] [, …]
[where <condition>]
[group by <column list>
 [having <condition>]]
[order by <column list>];
```

- Sequence of processing

from	specifies the table or tables to be used
where	filters the tuples subject to some condition
group by	forms groups of tuples with the same grouping attribute value
having	filters the groups subject to some condition
select	specifies which attributes are to appear in the output
order by	specifies the order of the output

Nested Queries

- A nested query or subquery is a select-from-where expression that is nested within another SQL query
- The result of an inner query is used to compute the result in the outer query
- Use cases
  - Test for set membership in the **where** clause
  - Set comparisons in the **where** clause
  - Set comparisons in the **having** clause
  - Test for empty tables in the **where** clause
  - Test for the absence of duplicate tuples in the **where** clause
  - Subqueries in the **from** clause
  - **with** clause for defining temporary tables
  - Scalar subqueries in the **where**, **select**, or **having** clause

Test for Set Membership in the WHERE Clause (I)

- The **in** connective tests for set membership in a collection of values produced by a subquery
- The **not in** connective tests for the absence of set membership
- Query 1: Output the names of students who have taken a test.

```
select name
from students
where reg-id in (select reg-id from tests);
```

- Query 2: Find the names of all professors that are not involved in teaching.

```
select name
from professors
where pers-id not in (select held_by from lectures);
```

Test for Set Membership in the WHERE Clause (II)

- Use of **in** and **not in** on enumerated sets

  Query 3: Select the names of assistants whose names are neither ‘Platon’ nor ‘Newton’.

```
select name
from assistants
where name not in (‘Platon’, ‘Newton’);
```

- Test for set membership in a multi-attribute table

  Query 4: Find the names of all professors who gave a test and also taught the lecture that was tested.

```
select name from professors
where pers-id in (select held_by from lectures
                    where (id, held_by) in (select id, pers-id from tests));
```

Set Comparisons in the WHERE Clause (I)

- Comparison of a single value with the elements of a set of values
- Query 5: Find the names of all students whose semester number is greater than at least one of the semester numbers of the students ‘Fichte’ (10th semester) or ‘Carnap’ (3rd semester).

```
select name from students
where sem > some (select sem from students
                  where name = ‘Fichte’ or name = ‘Carnap’);
```

- A condition with **some** is true if it is satisfied for at least one element in a set

> some	“greater than at least one”	
>= some	“greater than or equal to at least one”	
< some	“less than at least one”	
<= some	“less than or equal to at least one”	
= some	“equal to at least one”	[Note: the same as **in**]
<> some	“unequal to at least one”	[Note: *not* the same as **not in**]

Set Comparisons in the WHERE Clause (II)

- Query 6: Find the students with the largest number of semesters.

```
select name from students
where sem >= all (select sem from students);
```

- A condition with **all** is true if it is satisfied for *all* elements in the set

> all	"greater than all"	
>= all	"greater than or equal to all"	
< all	"less than all"	
<= all	"less than or equal to all"	
= all	"equal to all"	[Note: *not* the same as **in**]
<> all	"unequal to all"	[Note: the same as **not in**]

Set Comparisons in the HAVING Clause

- Set comparisons can also be performed in the **having** clause
- Query 7: Find the identifiers of professors who teach the most.

```
select held_by
from lectures
group by held_by
having sum(hpw) >= all (select sum(hpw)
                        from lectures
                        group by held_by);
```

Test for Empty Tables in the WHERE Clause (I)

- The **exists** construct returns the value *true* if the table as the result of the argument subquery is nonempty
- The **exists** constructs implements the existential quantifier
- Query 8: Determine the identifiers of all students who have taken a test.

```
select reg-id
from students as s
where exists (select *
              from tests as t
              where t.reg-id = s.reg-id);
```

- Feature of correlated queries in SQL: A correlation name from an outer query (‘s’ in the above query) can be used in a subquery in the **where** clause (compare to tuple variables in the tuple-relational calculus)

Test for Empty Tables in the WHERE Clause (II)

- The **not exists** construct returns the value *true* if the table as the result of the argument subquery is empty

- Query 9: Output the names of professors who do not hold lectures.

```
select name
from professors
where not exists (select *
                  from lectures
                  where pers-id = held_by);
```

- Attributes from the outer query (“pers-id” in the above query) can be used in a subquery in the **where** clause

Test for Empty Tables in the WHERE Clause (III)

- The **not exists** construct can help to simulate the *universal quantifier* in SQL for which no own construct exists

- “Table *A* contains table *B*” is the same as “**not exists(***B* **except** *A*)”

- Query 10: Output the names of students who attend all lectures offered by professor Curie.

```
select s.name
from students as s
where not exists ((select id from lectures, professors
                   where pers-id = held_by and name = ‘Curie’)
                  except
                  (select l.id from attends as a, lectures as l
                   where l.id = a.id and a.reg-id = s.reg-id)
                 );
```

Test for the Absence of Duplicate Tuples in the WHERE Clause

- The **unique** construct returns the value *true* if the table as the result of the argument subquery contains no duplicate tuples

- Query 11: Determine the names of professors who have at most one assistant working for them.

  select name **from** professors **as** p
  where unique (**select** a.boss **from** assistants **where** p.pers-id = a.boss);

- The **not unique** construct returns the value *true* if the table as the result of the argument subquery contains duplicate tuples

- Query 12: Determine the names of professors who have at least two assistants working for them.

  select name **from** professors **as** p
  where not unique (**select** a.boss **from** assistants **where** p.pers-id = a.boss);

Subqueries in the FROM Clause

- Any select-from-where expression returns a table as a result
- Therefore, it can be inserted into another select-from-where expression anywhere that a table can appear, e.g., in the **from** clause
- Query 13: Output the identifiers of those lectures that are attended by more than 20 students.

```
select id
from (select id, count(*) as number from attends group by id)
where number > 20;
```

- The attributes of a subquery result can be used in the outer query ("number" in the query above)
- We could but do not have to use the **having** clause instead

The WITH Clause for Defining Temporary Tables

- A way of defining a temporary table whose definition is available only to the query in which the **with** clause is used

- Query 14: Output the titles of those lectures that are attended by more than 20 students.

```
with attendance_rate(id, number) as
        (select id, count(*)
          from attends
          group by id)
select l.title
from lectures as l, attendance_rate as a
where l.id = a.id and a.number > 20;
```

Scalar Subqueries in the WHERE, SELECT, or HAVING Clause (I)

- A scalar subquery returns a single tuple that consists of a single column and a single row and that is interpreted and used as a *single value*
- A scalar subquery can occur in the **select**, **where**, or **having** clause
- Query 15: Determine the name and semester number of those students with a semester number less than the average.

```
select name, sem
from students
where sem < (select avg(sem) from students);
```

- It must be guaranteed that the subquery only returns a single value; otherwise, a runtime error occurs
- A scalar subquery can be used immediately following a relational comparison operator (=, <, >, <=, >=, <>)

Scalar Subqueries in the WHERE, SELECT, or HAVING Clause (II)

- Query 16: List the name and semester number of all students whose semester number is greater than the average semester number, and compute by how much their semester number is greater than the average.

```
select name, sem, sem – (select avg(sem) from students) as semDiff
from students
where sem > (select avg(sem) from students);
```

- The subquery computes the average semester number and is replaced by that number both in the **where** clause and the **select** clause

- One is not allowed to write "**where** sem > **avg**(sem)" since aggregation functions cannot be used in the **where** clause