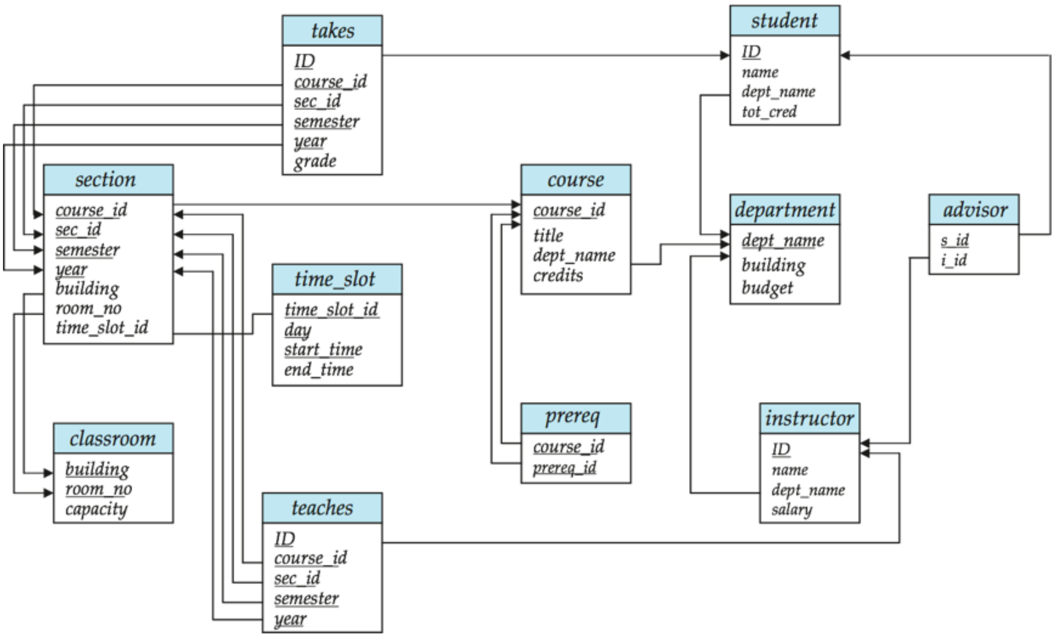
Aggregate functions and nested queries

Introduction to Database Design 2012, Lecture 3

Overview

- Null values
- Aggregate functions
- Nested queries
- Set operations: union, intersect and except

University db example



takes
ID
course_id
sec_id
semester
year
grade

student
ID
name
dept_name
tot_cred

section
course_id
sec_id
semester
year
building
room_no
time_slot_id

course
course_id
title
dept_name
credits

department
dept_name
building
budget

advisor
s_id
i_id

time_slot
time_slot_id
day
start_time
end_time

classroom
building
room_no
capacity

prereq
course_id
prereq_id

instructor
ID
name
dept_name
salary

teaches
ID
course_id
sec_id
semester
year

Use of null values

```
mysql> select * from takes;
+-------+-----------+--------+----------+------+-------+
| ID    | course_id | sec_id | semester | year | grade |
+-------+-----------+--------+----------+------+-------+
| 00128 | CS-101    | 1      | Fall     | 2009 | A     |
| 00128 | CS-347    | 1      | Fall     | 2009 | A-    |
| 12345 | CS-101    | 1      | Fall     | 2009 | C     |
| 12345 | CS-190    | 2      | Spring   | 2009 | A     |
| 12345 | CS-315    | 1      | Spring   | 2010 | A     |
| 12345 | CS-347    | 1      | Fall     | 2009 | A     |
| 19991 | HIS-351   | 1      | Spring   | 2010 | B     |
| 23121 | FIN-201   | 1      | Spring   | 2010 | C+    |
| 44553 | PHY-101   | 1      | Fall     | 2009 | B-    |
| 45678 | CS-101    | 1      | Fall     | 2009 | F     |
| 45678 | CS-101    | 1      | Spring   | 2010 | B+    |
| 45678 | CS-319    | 1      | Spring   | 2010 | B     |
| 54321 | CS-101    | 1      | Fall     | 2009 | A-    |
| 54321 | CS-190    | 2      | Spring   | 2009 | B+    |
| 55739 | MU-199    | 1      | Spring   | 2010 | A-    |
| 76543 | CS-101    | 1      | Fall     | 2009 | A     |
| 76543 | CS-319    | 2      | Spring   | 2010 | A     |
| 76653 | EE-181    | 1      | Spring   | 2009 | C     |
| 98765 | CS-101    | 1      | Fall     | 2009 | C-    |
| 98765 | CS-315    | 1      | Spring   | 2010 | B     |
| 98988 | BIO-101   | 1      | Summer   | 2009 | A     |
| 98988 | BIO-301   | 1      | Summer   | 2010 | NULL  |
+-------+-----------+--------+----------+------+-------+
22 rows in set (0,00 sec)
```

Null denotes grade not known yet

Null values

- The special value **null** denotes *unknown*

```
Table: classroom
+----------+-------------+----------+
| building | room_number | capacity |
+----------+-------------+----------+
| Packard  | 101         |      500 |
| Painter  | 514         |     NULL |
| Watson   | 120         |       50 |
+----------+-------------+----------+
```

- Creating tuples with **null** values

```
insert into classroom values (Painter, 514, NULL);
```

- or

```
insert into classroom(building, room_number)
    values (Painter, 514);
```

Not null constraints

- Disallowing null values

```
create table classroom
   (building    varchar(15),
    room_number varchar(7),
    capacity    numeric(4,0) not null,
    primary key (building, room_number)
   );
```

- Primary keys can not contain null values
- Foreign keys can

Null in boolean expressions

- Suppose attribute `capacity` can be null
- What does the below query mean?

```
select * from classroom where capacity > 100;
```

- Three truth values: true, false, unknown
- `null > 100` evaluates to unknown
- Select only those that evaluate to true
- But, then what does the following mean

```
select * from classroom
  where capacity > 100 or building = ‘Painter’;
```

Truthtables

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

Truthtables

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

not	true	false	unknown
	false	true	unknown

Aggregate functions

Aggregate functions

- Aggregate functions compute results based on contents of multiple tuples
- Aggregate functions in SQL
  - **avg**
  - **min**
  - **max**
  - **sum**
  - **count**

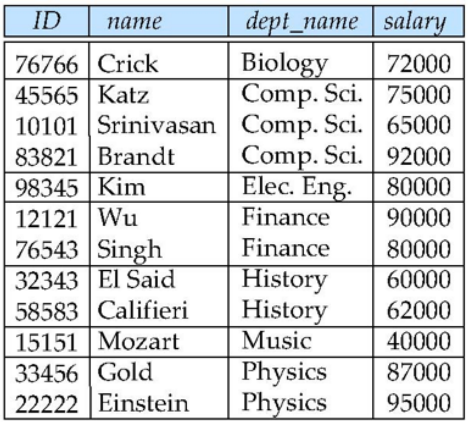
Examples

```
mysql> select count(*) from instructor;
+----------+
| count(*) |
+----------+
|       12 |
+----------+
1 row in set (0.00 sec)

mysql> select avg(salary) from instructor;
+--------------+
| avg(salary)  |
+--------------+
| 74833.333333 |
+--------------+
1 row in set (0.01 sec)

mysql> select min(salary) from instructor;
+-------------+
| min(salary) |
+-------------+
|    40000.00 |
+-------------+
1 row in set (0.00 sec)
```

Grouping and aggregation

```
mysql> select avg(salary), dept_name from instructor
    -> group by dept_name;
+--------------+------------+
| avg(salary)  | dept_name  |
+--------------+------------+
| 72000.000000 | Biology    |
| 77333.333333 | Comp. Sci. |
| 80000.000000 | Elec. Eng. |
| 85000.000000 | Finance    |
| 61000.000000 | History    |
| 40000.000000 | Music      |
| 91000.000000 | Physics    |
+--------------+------------+
7 rows in set (0.00 sec)
```

Semantics of aggregation



ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

- First group then compute average

```
mysql> select avg(salary), dept_name from instructor group by dept_name;
```

Restrictions on queries with grouping

- Only aggregate functions or attributes occurring in `group by` clause can be selected
- This should not be allowed

```
mysql> select avg(salary), dept_name, ID from instructor group by
dept_name;
+--------------+------------+-------+
| avg(salary)  | dept_name  | ID    |
+--------------+------------+-------+
| 72000.000000 | Biology    | 76766 |
| 77333.333333 | Comp. Sci. | 10101 |
| 80000.000000 | Elec. Eng. | 98345 |
| 85000.000000 | Finance    | 12121 |
| 61000.000000 | History    | 32343 |
| 40000.000000 | Music      | 15151 |
| 91000.000000 | Physics    | 22222 |
+--------------+------------+-------+
7 rows in set (0.00 sec)
```

- Result does not really make sense

Restrictions on queries with grouping

- This makes more sense

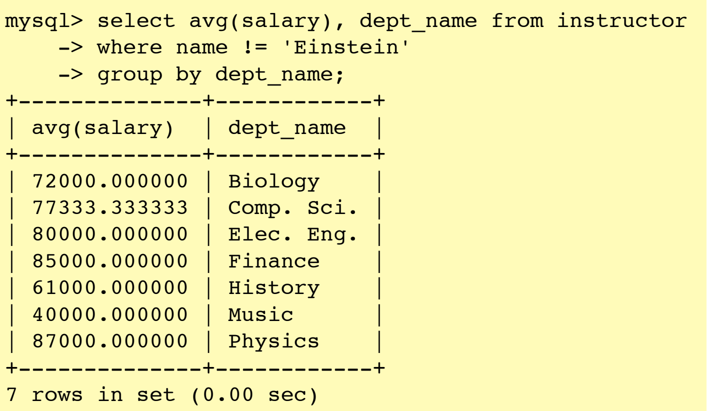
```
mysql> select avg(salary), dept_name, budget
    -> from instructor natural join department
    -> group by dept_name;
+--------------+------------+-----------+
| avg(salary)  | dept_name  | budget    |
+--------------+------------+-----------+
| 72000.000000 | Biology    |  90000.00 |
| 77333.333333 | Comp. Sci. | 100000.00 |
| 80000.000000 | Elec. Eng. |  85000.00 |
| 85000.000000 | Finance    | 120000.00 |
| 61000.000000 | History    |  50000.00 |
| 40000.000000 | Music      |  80000.00 |
| 91000.000000 | Physics    |  70000.00 |
+--------------+------------+-----------+
7 rows in set (0.01 sec)
```

Selections in queries with grouping

- Two variants of selection
  - Either
    - First select tuples, then group
    - Use **where**
  - Or
    - First group and then select on group level
    - Use **having**

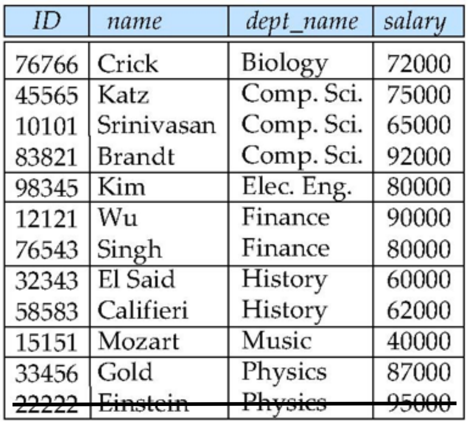
Example

- Compute average of salaries, not counting Einstein



```
mysql> select avg(salary), dept_name from instructor
    -> where name != 'Einstein'
    -> group by dept_name;
+--------------+------------+
| avg(salary)  | dept_name  |
+--------------+------------+
| 72000.000000 | Biology    |
| 77333.333333 | Comp. Sci. |
| 80000.000000 | Elec. Eng. |
| 85000.000000 | Finance    |
| 61000.000000 | History    |
| 40000.000000 | Music      |
| 87000.000000 | Physics    |
+--------------+------------+
7 rows in set (0.00 sec)
```

Semantics



ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
~~22222~~	~~Einstein~~	~~Physics~~	~~95000~~

- First delete tuples, then group and compute average

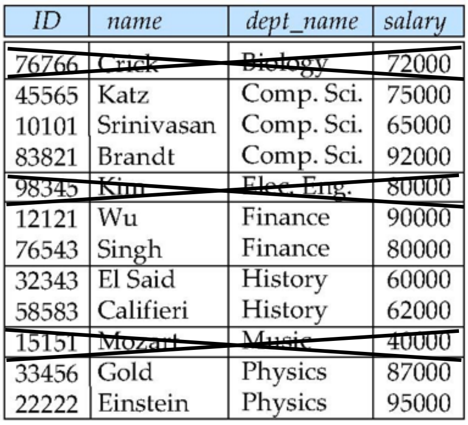
```
mysql> select avg(salary), dept_name from instructor
    -> where name != 'Einstein'
    -> group by dept_name;
```

Example

- Compute average of salaries of all departments with at least 2 instructors

```
mysql> select avg(salary), dept_name from instructor
    -> group by dept_name
    -> having count(ID)>1;
+--------------+------------+
| avg(salary)  | dept_name  |
+--------------+------------+
| 77333.333333 | Comp. Sci. |
| 85000.000000 | Finance    |
| 61000.000000 | History    |
| 91000.000000 | Physics    |
+--------------+------------+
4 rows in set (0.00 sec)
```

Semantics



ID	*name*	*dept_name*	*salary*
~~76766~~	~~Crick~~	~~Biology~~	~~72000~~
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
~~98345~~	~~Kim~~	~~Elec. Eng.~~	~~80000~~
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
~~15151~~	~~Mozart~~	~~Music~~	~~40000~~
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

- First group, then select groups

```
mysql> select avg(salary), dept_name from instructor
    -> group by dept_name
    -> having count(ID)>1;
```

Restrictions on having clauses

- Conditions in having clause are evaluated at group level
- So this is not allowed

```
mysql> select avg(salary), dept_name from instructor
    -> group by dept_name having name != 'Einstein';
ERROR 1054 (42S22): Unknown column 'name' in 'having clause'
```

Combining **where** and **having**

```
mysql> select avg(salary), dept_name from instructor
    -> where not name = 'Einstein'
    -> group by dept_name
    -> having count(ID)>1;
+--------------+------------+
| avg(salary)  | dept_name  |
+--------------+------------+
| 77333.333333 | Comp. Sci. |
| 85000.000000 | Finance    |
| 61000.000000 | History    |
+--------------+------------+
3 rows in set (0.00 sec)
```

- Semantics:
  - First select on tuple level
  - Then group
  - Then select on group level

Example

- Compute how many courses each student took in 2009

```
mysql> select * from takes natural join student;
+-------+-----------+--------+----------+------+-------+----------+------------+----------+
| ID    | course_id | sec_id | semester | year | grade | name     | dept_name  | tot_cred |
+-------+-----------+--------+----------+------+-------+----------+------------+----------+
| 00128 | CS-101    | 1      | Fall     | 2009 | A     | Zhang    | Comp. Sci. |      102 |
| 00128 | CS-347    | 1      | Fall     | 2009 | A-    | Zhang    | Comp. Sci. |      102 |
| 12345 | CS-101    | 1      | Fall     | 2009 | C     | Shankar  | Comp. Sci. |       32 |
| 12345 | CS-190    | 2      | Spring   | 2009 | A     | Shankar  | Comp. Sci. |       32 |
| 12345 | CS-315    | 1      | Spring   | 2010 | A     | Shankar  | Comp. Sci. |       32 |
| 12345 | CS-347    | 1      | Fall     | 2009 | A     | Shankar  | Comp. Sci. |       32 |
| 19991 | HIS-351   | 1      | Spring   | 2010 | B     | Brandt   | History    |       80 |
| 23121 | FIN-201   | 1      | Spring   | 2010 | C+    | Chavez   | Finance    |      110 |
| 44553 | PHY-101   | 1      | Fall     | 2009 | B-    | Peltier  | Physics    |       56 |
| 45678 | CS-101    | 1      | Fall     | 2009 | F     | Levy     | Physics    |       46 |
| 45678 | CS-101    | 1      | Spring   | 2010 | B+    | Levy     | Physics    |       46 |
| 45678 | CS-319    | 1      | Spring   | 2010 | B     | Levy     | Physics    |       46 |
| 54321 | CS-101    | 1      | Fall     | 2009 | A-    | Williams | Comp. Sci. |       54 |
| 54321 | CS-190    | 2      | Spring   | 2009 | B+    | Williams | Comp. Sci. |       54 |
| 55739 | MU-199    | 1      | Spring   | 2010 | A-    | Sanchez  | Music      |       38 |
| 76543 | CS-101    | 1      | Fall     | 2009 | A     | Brown    | Comp. Sci. |       58 |
| 76543 | CS-319    | 2      | Spring   | 2010 | A     | Brown    | Comp. Sci. |       58 |
| 76653 | EE-181    | 1      | Spring   | 2009 | C     | Aoi      | Elec. Eng. |       60 |
| 98765 | CS-101    | 1      | Fall     | 2009 | C-    | Bourikas | Elec. Eng. |       98 |
| 98765 | CS-315    | 1      | Spring   | 2010 | B     | Bourikas | Elec. Eng. |       98 |
| 98988 | BIO-101   | 1      | Summer   | 2009 | A     | Tanaka   | Biology    |      120 |
| 98988 | BIO-301   | 1      | Summer   | 2010 | NULL  | Tanaka   | Biology    |      120 |
+-------+-----------+--------+----------+------+-------+----------+------------+----------+
22 rows in set (0,00 sec)
```

Example (continued)

```
mysql> mysql> select * from takes natural join student where year = 2009;
+-------+-----------+--------+----------+------+-------+----------+------------+----------+
| ID    | course_id | sec_id | semester | year | grade | name     | dept_name  | tot_cred |
+-------+-----------+--------+----------+------+-------+----------+------------+----------+
| 00128 | CS-101    | 1      | Fall     | 2009 | A     | Zhang    | Comp. Sci. |      102 |
| 00128 | CS-347    | 1      | Fall     | 2009 | A-    | Zhang    | Comp. Sci. |      102 |
| 12345 | CS-101    | 1      | Fall     | 2009 | C     | Shankar  | Comp. Sci. |       32 |
| 12345 | CS-190    | 2      | Spring   | 2009 | A     | Shankar  | Comp. Sci. |       32 |
| 12345 | CS-347    | 1      | Fall     | 2009 | A     | Shankar  | Comp. Sci. |       32 |
| 44553 | PHY-101   | 1      | Fall     | 2009 | B-    | Peltier  | Physics    |       56 |
| 45678 | CS-101    | 1      | Fall     | 2009 | F     | Levy     | Physics    |       46 |
| 54321 | CS-101    | 1      | Fall     | 2009 | A-    | Williams | Comp. Sci. |       54 |
| 54321 | CS-190    | 2      | Spring   | 2009 | B+    | Williams | Comp. Sci. |       54 |
| 76543 | CS-101    | 1      | Fall     | 2009 | A     | Brown    | Comp. Sci. |       58 |
| 76653 | EE-181    | 1      | Spring   | 2009 | C     | Aoi      | Elec. Eng. |       60 |
| 98765 | CS-101    | 1      | Fall     | 2009 | C-    | Bourikas | Elec. Eng. |       98 |
| 98988 | BIO-101   | 1      | Summer   | 2009 | A     | Tanaka   | Biology    |      120 |
+-------+-----------+--------+----------+------+-------+----------+------------+----------+
13 rows in set (0,00 sec)
```

Example (continued)

```
mysql> select ID, name, count(*)
    -> from takes natural join student
    -> where year = 2009
    -> group by ID, name;
+-------+----------+----------+
| ID    | name     | count(*) |
+-------+----------+----------+
| 00128 | Zhang    |        2 |
| 12345 | Shankar  |        3 |
| 44553 | Peltier  |        1 |
| 45678 | Levy     |        1 |
| 54321 | Williams |        2 |
| 76543 | Brown    |        1 |
| 76653 | Aoi      |        1 |
| 98765 | Bourikas |        1 |
| 98988 | Tanaka   |        1 |
+-------+----------+----------+
9 rows in set (0,00 sec)
```

```
mysql> select name from takes natural join student
    -> where year = 2009
    -> group by ID
    -> having count(*)>1;
+----------+
| name     |
+----------+
| Zhang    |
| Shankar  |
| Williams |
+----------+
3 rows in set (0,00 sec)
```

Nested queries

Nested queries

- Nested queries allow us to use the results of queries in other queries
- Queries can be nested two places
  - In the **where** clause
  - In the **from** clause
- Queries can also be nested in insertions, updates or deletes

Queries nested in the **where** clause

- Suppose we want to find the instructors with the highest salaries
- First compute the highest salary, then find the instructors

```
mysql> select name from instructor
    -> where salary >= (select max(salary) from instructor);
+----------+
| name     |
+----------+
| Einstein |
+----------+
1 row in set (0.00 sec)
```

Scalar subqueries

- The query works because the subquery returns a table with exactly one row
- Such queries are called scalar subqueries
- This does not work:

```
mysql> select name from instructor
    -> where salary >= (select salary from instructor);
ERROR 1242 (21000): Subquery returns more than 1 row
```

Comparing values to tables

```
mysql> select name from instructor
    -> where salary >= all (select salary from instructor);
+----------+
| name     |
+----------+
| Einstein |
+----------+
1 row in set (0.00 sec)
```

```
mysql> select name from instructor
    -> where salary >= some
    -> (select salary from instructor where dept_name = 'Biology');
+----------+
| name     |
+----------+
| Wu       |
| Einstein |
| Gold     |
| Katz     |
| Singh    |
| Crick    |
| Brandt   |
| Kim      |
+----------+
8 rows in set (0.00 sec)
```

Nested queries, example

- Nested queries are useful because they allow us to break problems into smaller problems
- Find names of instructors working in departments with more than one instructor
- Can use query constructed earlier

```
mysql> select dept_name from instructor
    -> group by dept_name
    -> having count(ID) > 1;
```

Nested queries, example

- Find names of instructors working in departments with more than one instructor

```
mysql> select name from instructor
    -> where dept_name in
    -> (select dept_name from instructor
    ->  group by dept_name
    ->  having count(ID) > 1);
+------------+
| name       |
+------------+
| Srinivasan |
| Wu         |
| Einstein   |
| El Said    |
| Gold       |
| Katz       |
| Califieri  |
| Singh      |
| Brandt     |
+------------+
9 rows in set (0.00 sec)
```

Alternative

```
mysql> select name from instructor as S
    -> where 1 <
    -> (select count(ID) from instructor
    ->  where dept_name = S.dept_name);
+------------+
| name       |
+------------+
| Srinivasan |
| Wu         |
| Einstein   |
| El Said    |
| Gold       |
| Katz       |
| Califieri  |
| Singh      |
| Brandt     |
+------------+
9 rows in set (0.00 sec)
```

How many subqueries are run in this query compared to the one on the previous slide?

Warning: The answer to the above question may give the impression that one query is faster than the other. This may not be the case

- Subquery is called a *correlated subquery*

Alternative, no nesting

```
mysql> select S.name
    -> from instructor as S join instructor as R on (S.dept_name = R. dept_name)
    -> group by S.ID having count(R.ID) > 1;
+------------+
| name       |
+------------+
| Srinivasan |
| Wu         |
| Einstein   |
| El Said    |
| Gold       |
| Katz       |
| Califieri  |
| Singh      |
| Brandt     |
+------------+
9 rows in set (0.00 sec)
```

- Point is: using nested queries is often simpler

Nesting in the from clause

- Yet another way of finding the instructor with the highest salary

```
mysql> select name from instructor,
    -> (select max(salary) as max_salary from instructor) as S
    -> where instructor.salary = S.max_salary;
+----------+
| name     |
+----------+
| Einstein |
+----------+
1 row in set (0.00 sec)
```

- When nesting in the `from` clause renaming is mandatory
- A `natural join` can be thought of as a nested query in the from clause

Nested queries in insertions and deletes

```
mysql> insert into instructor
    -> values (12, 'Rasmus', 'Comp. Sci.',
    -> (select avg(salary) from instructor as S));
Query OK, 1 row affected, 1 warning (0,00 sec)
```

```
mysql> update instructor set salary = salary*2
    -> where ID = 12;
Query OK, 1 row affected (0,00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> update instructor
    -> set salary =
    -> 2*(select max(salary) from instructor as S)
    -> where ID = 12;
ERROR 1093 (HY000): You can't specify target table
'instructor' for update in FROM clause
```

```
mysql> delete from instructor where ID = 12;
Query OK, 1 row affected (0,01 sec)
```

More constructions

- I did not cover:
  - **Lateral** keyword,
  - nested queries using **with** (not supported by MySQL),
  - aggregation and null values,
  - **exists** keyword
- Read about these in book

Set operations

Sets and multisets in relational databases

- Recall that tables are *sets* of tuples
  - No duplicates
  - Order not important
  - Set theory example {0,1,2,4} = {0,2,4,1}
- Results of queries are multisets
  - Duplicates allowed
  - Tuples can be delivered in any order, unless order specified
  - Multiset example {0,1,2,2} = {0,2,1,2} ≠ {0,1,2}

Set operations

- On sets

$$\{0,1,4,5\} \cup \{2,4,8\} = \{0,1,2,4,5,8\}$$
$$\{0,1,4,5\} \cap \{2,4,8\} = \{4\}$$
$$\{0,1,4,5\} \setminus \{2,4,8\} = \{0,1,5\}$$

- On multisets

$$\{0,4,5\} \cup \{2,4,8\} = \{0,2,4,4,5,8\}$$
$$\{0,0,4,5\} \cap \{0,4,8\} = \{0,4\}$$
$$\{0,0,4,5\} \setminus \{0,4,4,8\} = \{0,5\}$$

Set operations on queries

- Set theoretic:
  - **union**, **intersect**, **except**
- Semantics: First eliminate duplicates, then apply set theoretic operations
- Multiset operations
  - **union all**, **intersect all**, **except all**
- Semantics: apply multiset operations

Examples

```
mysql> select * from instructor;
+-------+------------+------------+----------+
| ID    | name       | dept_name  | salary   |
+-------+------------+------------+----------+
| 10101 | Srinivasan | Comp. Sci. | 65000.00 |
| 12121 | Wu         | Finance    | 90000.00 |
| 15151 | Mozart     | Music      | 40000.00 |
| 22222 | Einstein   | Physics    | 95000.00 |
| 32343 | El Said    | History    | 60000.00 |
| 33456 | Gold       | Physics    | 87000.00 |
| 45565 | Katz       | Comp. Sci. | 75000.00 |
| 58583 | Califieri  | History    | 62000.00 |
| 76543 | Singh      | Finance    | 80000.00 |
| 76766 | Crick      | Biology    | 72000.00 |
| 83821 | Brandt     | Comp. Sci. | 92000.00 |
| 98345 | Kim        | Elec. Eng. | 80000.00 |
+-------+------------+------------+----------+
12 rows in set (0.01 sec)
```

Examples

```
mysql> select dept_name from instructor where name like 'E%'
    -> union all
    -> select dept_name from instructor where name like 'G%';
+-----------+
| dept_name |
+-----------+
| Physics   |
| History   |
| Physics   |
+-----------+
3 rows in set (0.00 sec)
```

```
mysql> select dept_name from instructor where name like 'E%'
    -> union
    -> select dept_name from instructor where name like 'G%';
+-----------+
| dept_name |
+-----------+
| Physics   |
| History   |
+-----------+
2 rows in set (0.00 sec)
```

Summary

- Null denotes unknown. Leads to three value logic
- Aggregate functions compute single values from sets of tuples
- Queries can be nested in **where** or **from** clause
- We can use nested queries to break problems into smaller problems
- Two versions of set operations: for sets and multisets