Set Membership

Set Membership

- Find courses offered in Fall 2017

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017;
```

- Find courses offered in Fall 2017 and in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
      course_id in (select course_id
                    from section
                    where semester = 'Spring' and year= 2018);
```

- Find courses offered in Fall 2017 but not in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
      course_id not in (select course_id
                        from section
                        where semester = 'Spring' and year= 2018);
```


Set Membership (Cont.)

- Name all instructors whose name is neither "Mozart" nor Einstein"

```
select distinct name
from instructor
where  name not in ('Mozart', 'Einstein')
```

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
        (select course_id, sec_id, semester, year
         from teaches
         where teaches.ID= 10101);
```

- Note: Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features

Set Comparison

Set Comparison – "some" Clause

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';
```

- Same query using > some clause

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = 'Biology');
```


Definition of "some" Clause

- F <comp> some r t such that (F <comp> t)
  Where <comp> can be:

(5 < some | 0 5 6 |) = true (read: 5 < some tuple in the relation)

(5 < some | 0 5 |) = false

(5 = some | 0 5 |) = true

(5 ≠ some | 0 5 |) = true (since 0 ≠ 5)

(= some) ≡ in
However, (≠ some) ≡ not in

Set Comparison – "all" Clause

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```


Definition of "all" Clause

- F <comp> all r t r (F <comp> t)

(5 < all | 0 5 6 |) = false

(5 < all | 6 10 |) = true

(5 = all | 4 5 |) = false

(5 ≠ all | 4 6 |) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠all) ≡not in
However, (= all) ≡ in

Test for Empty Relations

- The exists construct returns the value true if the argument subquery is nonempty.
- exists r r ≠ Ø
- not exists r r = Ø

Use of "exists" Clause

- Yet another way of specifying the query "Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester"

```
select course_id
from section as S
where semester = 'Fall' and year = 2017 and
      exists (select *
              from section as T
              where semester = 'Spring' and year= 2018
                    and S.course_id = T.course_id);
```

- Correlation name – variable S in the outer query
- Correlated subquery – the inner query
- "Find all courses taught in the Fall 2017 semester but not in the Spring 2018 semester"
  - How to revise above query?
  - exist ¬ not exist

Use of "not exists" Clause

- Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                   except
                   (select T.course_id
                    from takes as T
                    where S.ID = T.ID));
```

  - First nested query lists all courses offered in Biology
  - Second nested query lists all courses a particular student took
- Note that X – Y = Ø ¬ X ® Y
- Note: Cannot write this query using = all and its variants

Test for Absence of Duplicate Tuples

- The unique construct tests whether a subquery has any duplicate tuples in its result.
- The unique construct evaluates to "true" if a given subquery contains no duplicates .
- Find all courses that were offered at most once in 2017

```
select T.course_id
from course as T
where unique ( select R.course_id
               from section as R
               where T.course_id= R.course_id
                     and R.year = 2017);
```

- Remark: MySQL not support unique query

Subqueries in the From Clause

Subqueries in the From Clause

- SQL allows a subquery expression to be used in the from clause
- Find the average instructors' salaries of those departments where the average salary is greater than $42,000."

```
select dept_name, avg_salary
from ( select dept_name, avg (salary) as avg_salary
       from instructor
       group by dept_name) as dept_avg
where avg_salary > 42000;
```

- Note that we do not need to use the having clause
- Another way to write above query

```
select dept_name, avg_salary
from ( select dept_name, avg (salary)
       from instructor
       group by dept_name)
       as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```


With Clause

- The with clause provides a way of defining a temporary relation whose definition is available only to the query in which the with clause occurs.
- Find all departments with the maximum budget

```
with max_budget (value) as
    (select max(budget)
     from department)
select department.dept_name
from department, max_budget
where department.budget = max_budget.value;
```


Complex Queries using With Clause

- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept_total (dept_name, value) as
    (select dept_name, sum(salary)
     from instructor
     group by dept_name),
dept_total_avg(value) as
    (select avg(value)
     from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```


Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- List the number of instructors in 'CIS' department

```
select count(*)
from instructor
where instructor.dept_name='Com.Sci';
```

- List all departments along with the number of instructors in each department

```
select dept_name,
    ( select count(*)
      from instructor
      where instructor.dept_name= department.dept_name )
    as num_instructors
from department;
```

- Runtime error if subquery returns more than one result tuple

Modification of the Database

- Deletion of tuples from a given relation.
- Insertion of new tuples into a given relation
- Updating of values in some tuples in a given relation

Deletion

- Delete all instructors

```
delete from instructor;
```

- Delete all instructors from the Finance department

```
delete from instructor
where dept_name= 'Finance';
```

- *Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building.*

```
delete from instructor
where dept_name in (select dept_name
                    from department
                    where building = 'Watson');
```


Deletion (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors

```
delete from instructor
where salary < (select avg (salary)
                from instructor);
```

  - Problem: as we delete tuples from *instructor*, the average salary changes
  - Solution used in SQL:
    1. First, compute avg (salary) and find all tuples to delete
    2. Next, delete all tuples found above (without recomputing avg or retesting the tuples)

Insertion

- Add a new tuple to *course*

```
insert into course
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4),
           ('CS-456', 'Data Mining', 'Comp. Sci.', 4);
```

- or equivalently

```
insert into course (course_id, title, dept_name, credits)
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4),
           ('CS-456', 'Data Mining', 'Comp. Sci.', 4);
```

- Add a new tuple to *student* with *tot_creds* set to null

```
insert into student
    values ('3003', 'Green', 'Finance', null);
```


Insertion (Cont.)

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.

```
insert into instructor
select ID, name, dept_name, 18000
    from student
    where dept_name = 'Music' and total_cred > 144;
```

- The select from where statement is evaluated fully before any of its results are inserted into the relation.

  Otherwise queries like

  insert into *table1* select * from *table1*

  would cause problem

Updates

- Give a 5% salary raise to all instructors

```
update instructor
    set salary = salary * 1.05
```

- Give a 5% salary raise to those instructors who earn less than 70000

```
update instructor
    set salary = salary * 1.05
    where salary < 70000;
```

- Give a 5% salary raise to instructors whose salary is less than average

```
update instructor
    set salary = salary * 1.05
    where salary < (select avg (salary)
                    from instructor);
```


Updates (Cont.)

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two update statements:

```
update instructor
    set salary = salary * 1.03
    where salary > 100000;
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

  - The order is important
  - Can be done better using the case statement (next slide)

Case Statement for Conditional Updates

- Same query as before but with case statement

```
update instructor
    set salary = case
                 when salary <= 100000 then salary * 1.05
                 else salary * 1.03
                 end
```


Updates with Scalar Subqueries

- Recompute and update tot_creds value for all students

```
update student S
set tot_cred = (select sum(credits)
                from takes, course
                where takes.course_id = course.course_id and
                      S.ID= takes.ID and
                      takes.grade <> 'F' and
                      takes.grade is not null);
```

- Sets *tot_creds* to null for students who have not taken any course
- Instead of sum(*credits*), use:

```
case
    when sum(credits) is not null then sum(credits)
    else 0
end
```


End of Chapter 3