Database Management Systems (CSN-351)
SQL (Contd. 2)

BTech 3rd Year (CS) + Minor + Audit

Instructor: **R**anita **B**iswas
Department of Computer Science and Engineering
Indian Institute of Technology Roorkee
Roorkee, Uttarakhand - 247 667, India

Online Classroom

url: `https://tinyurl.com/ya8yphht`
Secret code: <use the one announced during lecture>

Having Clause

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

- Find the average salary of all instructors in only those departments where the average salary of the instructors is more than 42,000.

Having Clause

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

- Find the average salary of all instructors in only those departments where the average salary of the instructors is more than 42,000.

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

Sequence of Operations

- As was the case for queries without aggregation, the **from** clause is first evaluated to get a relation.

Sequence of Operations

- As was the case for queries without aggregation, the **from** clause is first evaluated to get a relation.
- If a **where** clause is present, the predicate in the **where** clause is applied on the result relation of the **from** clause.

Sequence of Operations

- As was the case for queries without aggregation, the **from** clause is first evaluated to get a relation.
- If a **where** clause is present, the predicate in the **where** clause is applied on the result relation of the **from** clause.
- Tuples satisfying the **where** predicate are then placed into groups by the **group by** clause if it is present. If the **group by** clause is absent, the entire set of tuples satisfying the **where** predicate is treated as being in one group.

Sequence of Operations

- As was the case for queries without aggregation, the **from** clause is first evaluated to get a relation.
- If a **where** clause is present, the predicate in the **where** clause is applied on the result relation of the **from** clause.
- Tuples satisfying the **where** predicate are then placed into groups by the **group by** clause if it is present. If the **group by** clause is absent, the entire set of tuples satisfying the **where** predicate is treated as being in one group.
- The **having** clause, if it is present, is applied to each group; the groups that do not satisfy the **having** clause predicate are removed.

Sequence of Operations

- As was the case for queries without aggregation, the **from** clause is first evaluated to get a relation.
- If a **where** clause is present, the predicate in the **where** clause is applied on the result relation of the **from** clause.
- Tuples satisfying the **where** predicate are then placed into groups by the **group by** clause if it is present. If the **group by** clause is absent, the entire set of tuples satisfying the **where** predicate is treated as being in one group.
- The **having** clause, if it is present, is applied to each group; the groups that do not satisfy the **having** clause predicate are removed.
- The **select** clause uses the remaining groups to generate tuples of the result of the query, applying the aggregate functions to get a single result tuple for each group.

Having and Where Together

student(<u>*ID*</u>, *name*, *dept_name*, *tot_cred*)

takes(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *grade*)

- For each course section offered in 2009, find the average total credits (*tot_cred*) of all students enrolled in the section, if the section had at least 2 students.

Having and Where Together

student(<u>*ID*</u>, *name*, *dept_name*, *tot_cred*)

takes(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *grade*)

- For each course section offered in 2009, find the average total credits (*tot_cred*) of all students enrolled in the section, if the section had at least 2 students.

```
select course_id, semester, year, sec_id, avg (tot_cred)
from takes natural join student
where year = 2009
group by course_id, semester, year, sec_id
having count (ID) >= 2;
```

Nested Subqueries: Set Membership

section(<u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *building*, *room_number*, *time_slot_id*)

- Find all the courses taught in both the Fall 2009 and Spring 2010 semesters.

Nested Subqueries: Set Membership

section(<u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *building*, *room_number*, *time_slot_id*)

- Find all the courses taught in both the Fall 2009 and Spring 2010 semesters.

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id in (select course_id
                      from section
                      where semester = 'Spring' and year= 2010);
```

Set Membership (contd.)

$\textit{section}(\underline{\textit{course\_id}}, \underline{\textit{sec\_id}}, \underline{\textit{semester}}, \underline{\textit{year}}, \textit{building}, \textit{room\_number}, \textit{time\_slot\_id})$

- Find all the courses taught in the Fall 2009 semester but not in the Spring 2010 semester.

Set Membership (contd.)

section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)

- Find all the courses taught in the Fall 2009 semester but not in the Spring 2010 semester.

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course_id not in (select course_id
                  from section
                  where semester = 'Spring' and year= 2010);
```

Set Membership (contd.)

instructor(*ID*, *name*, *dept_name*, *salary*)

takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)

- Find the names of instructors whose names are neither “Mozart” nor “Einstein”.

Set Membership (contd.)

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

takes(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *grade*)

- Find the names of instructors whose names are neither "Mozart" nor "Einstein".

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

Set Membership (contd.)

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

takes(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *grade*)

- Find the names of instructors whose names are 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.

Set Membership (contd.)

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

takes(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *grade*)

- Find the names of instructors whose names are 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);
```

Set Comparison

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

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

Set Comparison

instructor(*ID*, *name*, *dept_name*, *salary*)

- Find the names of all instructors whose salary is greater than 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';
```

Set Comparison

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

- Find the names of all instructors whose salary is greater than 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';
```

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

Set Comparison (contd.)

instructor(ID, *name*, *dept_name*, *salary*)

- Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

Set Comparison (contd.)

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

- Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

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

Set Comparison (contd.)

instructor(ID, *name*, *dept_name*, *salary*)

- Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

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

- Find the departments that have the highest average salary.

Set Comparison (contd.)

instructor(*ID*, *name*, *dept_name*, *salary*)

- Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

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

- Find the departments that have the highest average salary.

```
select dept_name
from instructor
group by dept_name
having avg (salary) >= all (select avg (salary)
                            from instructor
                            group by dept_name);
```

Set Comparison (contd.)

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

- Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

Set Comparison (contd.)

instructor(*ID*, *name*, *dept_name*, *salary*)

- Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

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

Set Comparison (contd.)

instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)

- Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

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

- Find the departments that have the highest average salary.

Set Comparison (contd.)

instructor(*ID*, *name*, *dept_name*, *salary*)

- Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

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

- Find the departments that have the highest average salary.

```
select dept_name
from instructor
group by dept_name
having avg (salary) >= all (select avg (salary)
                            from instructor
                            group by dept_name);
```

Test for Empty Relations

$section(\underline{course\_id}, \underline{sec\_id}, \underline{semester}, \underline{year}, building, room\_number, time\_slot\_id)$

- Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester.

Test for Empty Relations

section(<u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *building*, *room_number*, *time_slot_id*)

- Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester.

```
select course_id
from section as S
where semester = 'Fall' and year= 2009 and
        exists (select *
                from section as T
                where semester = 'Spring' and year= 2010 and
                        S.course_id= T.course_id);
```

Test for Empty Relations

section(<u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *building*, *room_number*, *time_slot_id*)

- Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester.

```
select course_id
from section as S
where semester = 'Fall' and year= 2009 and
        exists (select *
                from section as T
                where semester = 'Spring' and year= 2010 and
                        S.course_id= T.course_id);
```

Correlated subquery

Test for Empty Relations (contd.)

$$\textit{student}(\underline{\textit{ID}}, \textit{name}, \textit{dept\_name}, \textit{tot\_cred})$$

$$\textit{course}(\underline{\textit{course\_id}}, \textit{title}, \textit{dept\_name}, \textit{credits})$$

$$\textit{takes}(\underline{\textit{ID}}, \underline{\textit{course\_id}}, \underline{\textit{sec\_id}}, \underline{\textit{semester}}, \underline{\textit{year}}, \textit{grade})$$

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

Test for Empty Relations (contd.)

student(<u>*ID*</u>, *name*, *dept_name*, *tot_cred*)

course(<u>*course_id*</u>, *title*, *dept_name*, *credits*)

takes(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *grade*)

- 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));
```