Basic sql-part 4

Aggreagate and null value to scalar sub-query

Null Values and Aggregates

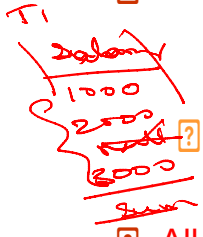
- Total all salaries

  select sum (*salary*)
  from *instructor*

- Above statement ignores null amounts

- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes

- What if collection has only null values?
  - count returns 0
  - all other aggregates return null



SELECT 2 + NULL + 2 + 2 + 2 + 5 + 5 + 2 + 5

This **SELECT** statement will return NULL, since NULL plus anything always returns NULL.

If we use the **SUM()** function on the table, however, we can rest assured that NULLs are eliminated.

Result=25

Table1:

```
Field1     Field2     Field3
-------------------------
  1          1          1
NULL       NULL       NULL
  2          2          NULL
  1          3          1
```

Then

```
SELECT COUNT(*), COUNT(Field1), COUNT(Field2), COUNT(DISTINCT Field3)
FROM Table1
```

Output Is:

```
COUNT(*) = 4; -- count all rows even null/duplicates

-- count only rows without null values on that field
COUNT(Field1) = COUNT(Field2) = 3

COUNT(Field3) = 2
COUNT(DISTINCT Field3) = 1 -- Ignore duplicates
```

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for
  1. set membership,
  2. set comparisons,
  3. set cardinality.

- Set membership
- SQL allows testing tuples for membership in a relation. The **in** connective tests for set membership, where the set is a collection of values produced by a **select** clause.

- The **not in** connective tests for the absence of set membership.

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

- Question hint " by checking for membership….."

Set Membership (Operates: in, not in)

- Find courses offered in Fall 2009 and in Spring 2010

```
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);
```

- Find courses offered in Fall 2009 but not in Spring 2010

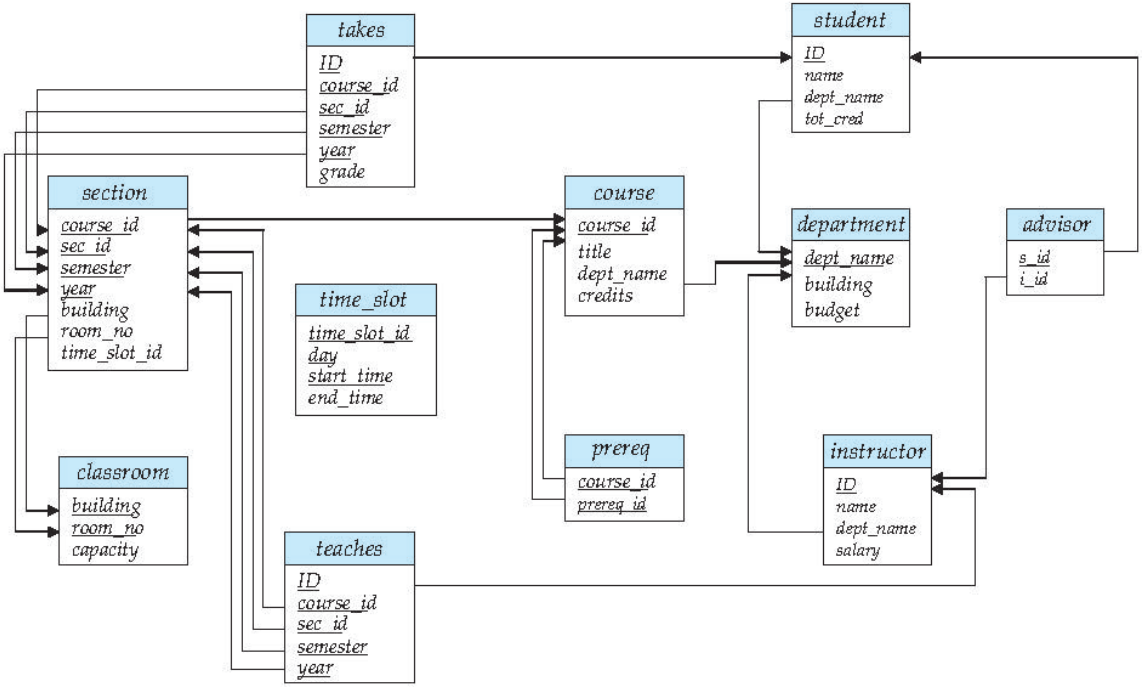
```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
      course_id not in (select course_id
                        from section
                        where semester = 'Spring' and year= 2010);
```

Example Query

- 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);
```

- The **in** and **not in** operators can also be used on **enumerated sets.** The following query selects the names of instructors whose names are neither “Mozart” nor “Einstein”.
- **select distinct** *name*
- **from** *instructor* **where** *name* **not in** (’Mozart’, ’Einstein’);


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
prereq
course_id
prereq_id
instructor
ID
name
dept_name
salary
classroom
building
room_no
capacity
teaches
ID
course_id
sec_id
semester
year

Set Comparison (operators: some, all)

- 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');
```

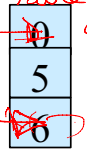
- The > **some** comparison in the **where** clause of the outer **select** is true if the *salary* value of the tuple is greater than at least one member of the set of all salary values for instructors in Biology.

- SQL also allows < **some**, <= **some**, >= **some**, = **some**, and <> **some** comparison
- = **some** is identical to **in**,
- whereas <> **some** is *not* the same as **not in**

Definition of Some Clause

- F <comp> **some** $r \Leftrightarrow \exists t \in r$ such that (F <comp> t)

  Where <comp> can be: $<$, $\leq$, $=$, $\neq$

(5 < **some** 

) = true

(read: 5 < some tuple in the relation)

(5 < **some** 

) = false

(5 = **some** 

) = true

(5 ≠ **some** 

) = true (since 0 ≠ 5)

(= **some**) ≡ **in**

However, (≠ **some**) ≢ **not in**

Example Query

- 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');
```

As it does for **some**, SQL also allows < **all**, <= **all**, >= **all**, = **all**, and <> **all** comparisons.

As an exercise, verify that <> **all** is identical to ~~**not in**~~, whereas = **all** is *not* the same as **in**.

Definition of all Clause

- F <comp> **all** $r \Leftrightarrow \forall t \in r$ (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/ set cardinality

- The **exists** construct returns the value **true** if the argument subquery is nonempty.

- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

- *Ex:* Get the list of courses which do not have pre-requisite courses.

Using SET operation

Select course_id from Course **except** select cours_id from Prereq

Using Nested Query : NOT exists

Select course_id from Cousre s where not exists (select pre_id from Prereq P where P.course_id=S.course_id)

Correlation Variables

- Yet another way of specifying the query "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**
- **Correlation name** or **correlation variable**
-

Important terminology

- The above query also illustrates a feature of SQL where a correlation name from an outer query (*s* in the above query), can be used in a subquery in the **where** clause.

- What is **correlated subquery?**
- A subquery that uses a correlation name from an outer query is
- called a **correlated subquery**

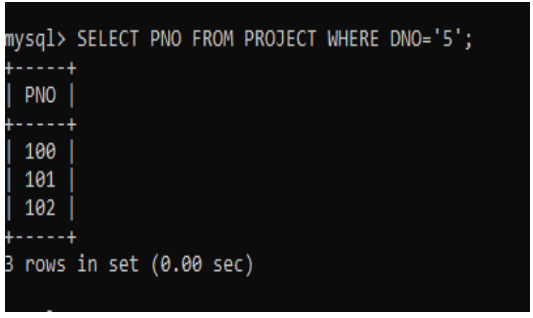
- In queries that contain subqueries, a scoping rule applies for correlation
- names. In a subquery, according to the rule, it is legal to use only correlation names defined in the subquery itself or in any query that contains the subquery.
- If a correlation name is defined both locally in a subquery and globally in a containing query, the local definition applies

```
mysql> SELECT * FROM EMPLOYEE;
+-----------+------+----------+----------+-------+-----------+------+--------+
| SSN       | DNO  | SUPERSSN | FNAME    | LNAME | ADDRESS   | SEX  | SALARY |
+-----------+------+----------+----------+-------+-----------+------+--------+
| RNSACC01  | 1    | RNSACC02 | AHANA    | K     | MANGALORE | F    | 350000 |
| RNSACC02  | 1    | NULL     | SANTHOSH | KUMAR | MANGALORE | M    | 300000 |
| RNSCSE01  | 5    | RNSCSE02 | JAMES    | SMITH | BANGALORE | M    | 500000 |
| RNSCSE02  | 5    | RNSCSE03 | HEARN    | BAKER | BANGALORE | M    | 700000 |
| RNSCSE03  | 5    | RNSCSE04 | EDWARD   | SCOTT | MYSORE    | M    | 500000 |
| RNSCSE04  | 5    | RNSCSE05 | PAVAN    | HEGDE | MANGALORE | M    | 650000 |
| RNSCSE05  | 5    | RNSCSE06 | GIRISH   | MALYA | MYSORE    | M    | 450000 |
| RNSCSE06  | 5    | NULL     | NEHA     | SN    | BANGALORE | F    | 800000 |
| RNSECE01  | 3    | NULL     | JOHN     | SCOTT | BANGALORE | M    | 450000 |
| RNSISE01  | 4    | NULL     | VEENA    | M     | MYSORE    | M    | 600000 |
| RNSIT01   | 2    | NULL     | NAGESH   | HR    | BANGALORE | M    | 500000 |
+-----------+------+----------+----------+-------+-----------+------+--------+
11 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM PROJECT;
+-----+--------------------+------+-----------+
| PNO | PNAME              | DNO  | PLOCATION |
+-----+--------------------+------+-----------+
| 100 | IOT                | 5    | BANGALORE |
| 101 | CLOUD              | 5    | BANGALORE |
| 102 | BIGDATA            | 5    | BANGALORE |
| 103 | SENSORS            | 3    | BANGALORE |
| 104 | BANK MANAGEMENT    | 1    | BANGALORE |
| 105 | SALARY MANAGEMENT  | 1    | BANGALORE |
| 106 | OPENSTACK          | 4    | BANGALORE |
| 107 | SMART CITY         | 2    | BANGALORE |
+-----+--------------------+------+-----------+
8 rows in set (0.00 sec)
```

```
mysql> SELECT PNO FROM PROJECT WHERE DNO='8';
Empty set (0.00 sec)
```

```
ysql> SELECT E.FNAME, E.LNAME FROM EMPLOYEE E WHERE EXISTS(SELECT PNO FROM PROJECT WHERE DNO='8');
mpty set (0.00 sec)
```


mysql> SELECT PNO FROM PROJECT WHERE DNO='5';
+-----+
| PNO |
+-----+
| 100 |
| 101 |
| 102 |
+-----+
3 rows in set (0.00 sec)

```
mysql> SELECT E.FNAME, E.LNAME FROM EMPLOYEE E WHERE EXISTS(SELECT PNO FROM PROJECT WHERE DNO='5');
+----------+-------+
| FNAME    | LNAME |
+----------+-------+
| AHANA    | K     |
| SANTHOSH | KUMAR |
| JAMES    | SMITH |
| HEARN    | BAKER |
| EDWARD   | SCOTT |
| PAVAN    | HEGDE |
| GIRISH   | MALYA |
| NEHA     | SN    |
| JOHN     | SCOTT |
| VEENA    | M     |
| NAGESH   | HR    |
+----------+-------+
11 rows in set (0.00 sec)
```

Not Exists

We can test for the nonexistence of tuples in a subquery by using the **not exists**

construct

- ?
- ?

```
mysql> SELECT E.FNAME, E.LNAME FROM EMPLOYEE E WHERE NOT EXISTS(SELECT PNO FROM PROJECT WHERE DNO='8');
+----------+-------+
| FNAME    | LNAME |
+----------+-------+
| AHANA    | K     |
| SANTHOSH | KUMAR |
| JAMES    | SMITH |
| HEARN    | BAKER |
| EDWARD   | SCOTT |
| PAVAN    | HEGDE |
| GIRISH   | MALYA |
| NEHA     | SN    |
| JOHN     | SCOTT |
| VEENA    | M     |
| NAGESH   | HR    |
+----------+-------+
11 rows in set (0.00 sec)
```

not exists = false

not empty

```
mysql> SELECT E.FNAME, E.LNAME FROM EMPLOYEE E WHERE NOT EXISTS(SELECT PNO FROM PROJECT WHERE DNO='5');
Empty set (0.00 sec)
```

? Find the 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 T.ID = S.ID));
```

(**select** *course_id* **from** *course*
where *dept_name* = 'Biology')

- BIO101
- BIO102

select *T.course_id* **from** *takes* **as** *T*
where T.*ID* = S.*ID*));

- CS101
- CS102
- BIO101

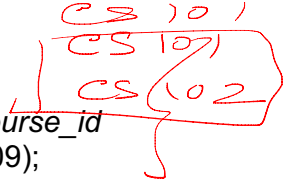
- EXCEPT: BIO102(there is a biology course not taken by student)

Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
  - (Evaluates to "true" on an empty set)

- Find all courses that were offered at most once in 2009

```
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 = 2009);
```



Q1: Get the students list who have taken at most one subject.

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)
where avg_salary > 42000;
```

- Note that we do not need to use the **having** clause

Subqueries in the From Clause (Cont.)

- To display instructor, his salary along with his department's average salary.

  And yet another way to write it: **lateral** clause

```
select name, salary, avg_salary
from instructor I1,
     lateral (select avg(salary) as avg_salary
              from instructor I2
              where I2.dept_name= I1.dept_name);
```

- Lateral clause permits later part of the **from** clause (after the lateral keyword) to access correlation variables from the earlier part.

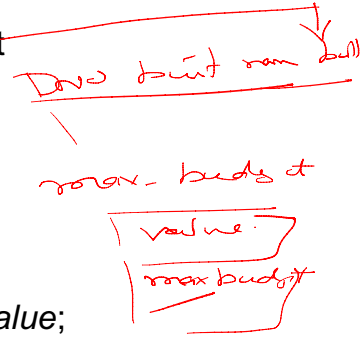
- Note: lateral is part of the SQL standard, but is not supported on many database systems; some databases such as SQL Server offer alternative syntax

With Clause

- The **with** clause provides a way of defining a temporary view/relation/table 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 budget
from department, max_budget
where department.budget = max_budget.value;
```



Complex Queries using With Clause

- With clause is very useful for writing complex queries
- Supported by most database systems, with minor syntax variations
- 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;
```

o/p: dept_total
Cs 50000
IT   60000
EC   70000

o/p: dept_total_avg
60000

o/p: EC department

Scalar Subquery

- Scalar subquery is one which is used where a single value is expected

- List out the number of instructors in each department.
- E.g.

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

- Runtime error if subquery returns more than one result tuple