Prep 5 - Part 1 of 1

Null Values

When we studied relational algebra, we assumed every attribute in every tuple had a value. But in a real database, sometimes information is missing. Read about Null Values in SQL from the **SQL: Null Values** link on the **Lectures** page on Quercus.

Not high salary ✓

Suppose this is the content of the Employee table:

```
 eid |       name       | salary | dept
-----+------------------+--------+------
   2 | Marissa Mayer    |     82 |   55
   3 | Sheryl Sandberg  |     17 |   55
   4 | Larry Ellison    |     55 |   33
   5 | Tim Cook         |     48 |   55
   6 | Mark Zuckerberg  |     95 |   22
   7 | Jeff Weiner      |     28 |   33
   8 | Larry Page       |    145 |   33
  25 | Julia            |        |   55
   1 | Bill Gates       |    100 |   55
(9 rows)
```

What is the output of this query?

```
select * from employee
where not salary > 100;
```

☐
```
 eid |       name       | salary | dept
-----+------------------+--------+------
   2 | Marissa Mayer    |     82 |   55
   3 | Sheryl Sandberg  |     17 |   55
   4 | Larry Ellison    |     55 |   33
   5 | Tim Cook         |     48 |   55
   6 | Mark Zuckerberg  |     95 |   22
   7 | Jeff Weiner      |     28 |   33
```

☑
```
 eid |       name       | salary | dept
-----+------------------+--------+------
   2 | Marissa Mayer    |     82 |   55
   3 | Sheryl Sandberg  |     17 |   55
   4 | Larry Ellison    |     55 |   33
   5 | Tim Cook         |     48 |   55
   6 | Mark Zuckerberg  |     95 |   22
   7 | Jeff Weiner      |     28 |   33
   1 | Bill Gates       |    100 |   55
```

☐
```
 eid |       name       | salary | dept
-----+------------------+--------+------
   2 | Marissa Mayer    |     82 |   55
   3 | Sheryl Sandberg  |     17 |   55
   4 | Larry Ellison    |     55 |   33
   5 | Tim Cook         |     48 |   55
   6 | Mark Zuckerberg  |     95 |   22
   7 | Jeff Weiner      |     28 |   33
  25 | Julia            |        |   55
   1 | Bill Gates       |    100 |   55
```

☐
```
 eid |       name       | salary | dept
-----+------------------+--------+------
   2 | Marissa Mayer    |     82 |   55
   3 | Sheryl Sandberg  |     17 |   55
   4 | Larry Ellison    |     55 |   33
```

```
     5 | Tim Cook         |      48 |    55
     6 | Mark Zuckerberg |      95 |    22
     7 | Jeff Weiner      |      28 |    33
    25 | Julia            |         |    55
```

History Submit

✓ Your solution is complete.

Submitted after the deadline!

Comparing counts

✓

This SQL query runs without error:

```
SELECT count(stuff), count(*), count(distinct stuff)
FROM StuffAndNonsense;
```

(We learned about using distinct inside an aggregation last week.)

Which of the following statements are true?

- [✓] **count(stuff) <= count(*)**
- [ ] **count(*) <= count(stuff)**
- [ ] **count(*) <= count(distinct stuff)**
- [✓] **count(distinct stuff) <= count(*)**
- [ ] **count(stuff) <= count(distinct stuff)**
- [✓] **count(distinct stuff) <= count(stuff)**

History Submit

✓ Your solution is complete.

Submitted after the deadline!

Different Kinds of Joins

To prepare for the next questions, read the **SQL: Joins** page on the **Lectures** page on Quercus.

For the following queries, follow these rules:

- **Duplicates**: Do not remove any duplicates.
- **Technique**: Use the technique specified. You could these in other ways, for instance with GROUP BY and aggregation, but the purpose of this prep is to practise various kinds of joins. If you don't use the specified technique, your answer may be graded as incorrect due to having differences in what duplicates are or are not present.
- **Order**: The order in which your tuples are returned does not matter, unless explicitly specified.

We begin with inner joins, for which there is no padding with NULL values.

Double Manager

✓

sales(eid, day, amount)
employee(eid, name, salary, dept)

manages(manager, junior)
department(did, name, division)
employee[dept] ⊆ department[did]
manages[manager] ⊆ employee[eid]
manages[junior] ⊆ employee[eid]
sales[eid] ⊆ employee[eid]

Report the eid of every employee who manages at least two different people. Use the usual technique of table renaming and self-join, except use the keywords `CROSS JOIN` to take the Cartesian product of the two tables:

```
SELECT * FROM Table1 CROSS JOIN Table2
```

`Table1 CROSS JOIN Table2` is completely equivalent to `Table1, Table2` - the latter is simply a convenient shorthand for us.
Your output table should have a single column named 'manager'.

```
select m1.manager as manager
from manages m1, manages m2
where m1.manager = m2.manager and m1.junior <> m2.junior
```

History Submit

✓ Your submission is correct!

Submitted after the deadline!

✓ Test Case Passed

Expected Result is Hidden

Actual

manager
1
1
3
3

Rich Sales ✓

sales(eid, day, amount)
employee(eid, name, salary, dept)
manages(manager, junior)
department(did, name, division)
employee[dept] ⊆ department[did]
manages[manager] ⊆ employee[eid]
manages[junior] ⊆ employee[eid]
sales[eid] ⊆ employee[eid]

Report the name and sales of everyone with a salary over 50, using the keywords `NATURAL JOIN` (plus a `WHERE` clause, of course):

```
SELECT * FROM Table1 NATURAL JOIN Table2
```

Your output table should have three columns: 'name', 'day', and 'amount', in that order.

```sql
select name, day, amount
from employee NATURAL JOIN sales
where salary > 50
```

History Submit

✔ Your submission is correct!

Submitted after the deadline!

✔ Test Case Passed

Expected Result is Hidden

Actual

name	day	amount
Larry Ellison	2011-11-01	155
Larry Ellison	2011-11-02	19
Larry Ellison	2011-11-03	10
Larry Ellison	2012-11-04	100
Larry Ellison	2012-12-01	25
Larry Ellison	2012-12-04	29
Larry Ellison	2009-12-05	20
Larry Ellison	2009-12-07	120
Mark Zuckerberg	2011-11-06	60
Mark Zuckerberg	2012-11-07	18
Mark Zuckerberg	2009-12-04	2
Mark Zuckerberg	2009-12-05	95
Larry Page	2012-11-01	98
Larry Page	2012-11-02	92
Larry Page	2009-12-31	91
Larry Page	2009-12-21	98

Department Salaries I ✓

sales(eid, day, amount)
employee(eid, name, salary, dept)
manages(manager, junior)
department(did, name, division)
employee[dept] ⊆ department[did]
manages[manager] ⊆ employee[eid]
manages[junior] ⊆ employee[eid]
sales[eid] ⊆ employee[eid]

Report the salaries of everyone in every department, using the keywords `INNER JOIN ... ON` (equivalent to theta join):

```
SELECT * FROM Table1 INNER JOIN Table2 ON Condition
```

Your output table should have three columns: 'name' (name of employee), 'department' (name of employee's department) and 'salary' (salary of an employee in that department).

```
select employee.name, department.name as department, employee.salary
from employee INNER JOIN department ON employee.dept = department.did
```

History Submit

✓ Your submission is correct!

Submitted after the deadline!

✓ Test Case Passed

Expected Result is Hidden

Actual

name	department	salary
Bill Gates	Widgets	59
Marissa Mayer	Widgets	82
Sheryl Sandberg	Widgets	17
Larry Ellison	Electronics	55
Tim Cook	Widgets	48
Mark Zuckerberg	Housewares	95
Jeff Weiner	Electronics	28
Larry Page	Electronics	145

Now let's think about outer joins. First, we'll do some tracing questions to confirm that you understand what they do.

Nulls and joins 1 ✓

Suppose table R contains this:

```
 a | b
---+---
 1 | 2
 8 | 7
 5 |
   | 6
```

and table S contains this:

```
 a | b
---+---
 3 | 4
 8 | 7
 5 |
   | 6
```

What is the output of this query?

```
SELECT *
FROM R Natural JOIN S;
```

- [ ]
```
 a | b
---+---
 1 | 2
 5 |
 8 | 7
   | 6
```
- [✓]
```
 a | b
---+---
 8 | 7
```
- [ ]
```
 a | b
---+---
 3 | 4
 5 |
 8 | 7
   | 6
```
- [ ]
```
 a | b
---+---
 1 | 2
 5 |
 3 | 4
 5 |
 8 | 7
   | 6
   | 6
```
- [ ] **None of the above**

History Submit

✓ Your solution is complete.

Submitted after the deadline!

Nulls and joins 2 ✓

Suppose table R contains this:

```
 a | b
---+---
 1 | 2
 8 | 7
 5 |
   | 6
```

and table S contains this:

```
 a | b
---+---
 3 | 4
 8 | 7
 5 |
   | 6
```

What is the output of this query?

```
SELECT *
FROM R NATURAL LEFT JOIN S;
```

- [✓]
```
 a | b
```

```
---+---
 1 | 2
 5 |
 8 | 7
   | 6
```

- [ ]
```
a | b
---+---
 8 | 7
```

- [ ]
```
 a | b
---+---
 3 | 4
 5 |
 8 | 7
   | 6
```

- [ ]
```
 a | b
---+---
 1 | 2
 5 |
 3 | 4
 5 |
 8 | 7
   | 6
   | 6
```

- [ ] **None of the above**

History Submit

✓ Your solution is complete.

Submitted after the deadline!

Nulls and joins 3

✓

Suppose table R contains this:

```
 a | b
---+---
 1 | 2
 8 | 7
 5 |
   | 6
```

and table S contains this:

```
 a | b
---+---
 3 | 4
 8 | 7
 5 |
   | 6
```

What is the output of this query?

```
SELECT *
FROM R NATURAL  RIGHTJOIN S;
```

- [ ]
```
 a | b
---+---
 1 | 2
 5 |
 8 | 7
   | 6
```

- [ ] **a | b**

```
---+---
 8 | 7
```

[✓]
```
 a | b
---+---
 3 | 4
 5 |
 8 | 7
   | 6
```

☐
```
 a | b
---+---
 1 | 2
 5 |
 3 | 4
 5 |
 8 | 7
   | 6
   | 6
```

☐ None of the above

History Submit

✓ Your solution is complete.

Submitted after the deadline!

Nulls and joins 4 ✓

Suppose table R contains this:

```
 a | b
---+---
 1 | 2
 8 | 7
 5 |
   | 6
```

and table S contains this:

```
 a | b
---+---
 3 | 4
 8 | 7
 5 |
   | 6
```

What is the output of this query?

```
SELECT *
FROM R NATURAL FULL JOIN S;
```

☐
```
 a | b
---+---
 1 | 2
 5 |
 8 | 7
   | 6
```

☐
```
a | b
---+---
 8 | 7
```

☐
```
 a | b
---+---
 3 | 4
 5 |
  8 | 7
    | 6
```

[✓]
```
  a | b
 ---+---
  1 | 2
  5 |
  3 | 4
  5 |
  8 | 7
    | 6
    | 6
```

☐ None of the above

History Submit

✓ Your solution is complete.

Submitted after the deadline!

Department Salaries II ✓

sales(eid, day, amount)
employee(eid, name, salary, dept)
manages(manager, junior)
department(did, name, division)
employee[dept] ⊆ department[did]
manages[manager] ⊆ employee[eid]
manages[junior] ⊆ employee[eid]
sales[eid] ⊆ employee[eid]

And now let's write some queries that require outer joins.

As in a previous question, report the salaries of everyone in every department
except this time, each employee with no department should appear in a single tuple in the output, with corresponding department being NULL (this is displayed in PCRS as simply "`null`")

As before, your output table should have three columns: 'name' (name of employee), 'department' (name of employee's department) and 'salary' (salary of an employee in that department).

```sql
select employee.name, department.name as department, employee.salary
from employee LEFT JOIN department ON employee.dept = department.did
```

History Submit

✓ Your submission is correct!

Submitted after the deadline!

✓ Test Case Passed

Expected Result is Hidden

Actual

name	department	salary
Bill Gates	Widgets	59

Marissa Mayer	Widgets	82
Sheryl Sandberg	Widgets	17
Larry Ellison	Electronics	55
Susan Wojcicki	null	300
Tim Cook	Widgets	48
Mark Zuckerberg	Housewares	95
Jeff Weiner	Electronics	28
Larry Page	Electronics	145
Dick Costolo	null	108

Department Salaries III ✓

sales(eid, day, amount)
employee(eid, name, salary, dept)
manages(manager, junior)
department(did, name, division)
employee[dept] ⊆ department[did]
manages[manager] ⊆ employee[eid]
manages[junior] ⊆ employee[eid]
sales[eid] ⊆ employee[eid]

Report the salaries of every person by department, as in the earlier questions, except that:

- Any employee belonging to no department is reported in the output with a NULL department.
- Any department with no employees is reported in the output with a NULL salary and employee name.

As before, your output table should have three columns, 'name', 'department', and 'salary', where any may be NULL in an output tuple.

```
select employee.name, department.name as department, employee.salary
from employee FULL JOIN department on employee.dept = department.did
```

History Submit

✓ Your submission is correct!

Submitted after the deadline!

✓ Test Case Passed

Expected Result is Hidden

Actual

name	department	salary
Bill Gates	Widgets	59
Marissa Mayer	Widgets	82
Sheryl Sandberg	Widgets	17
Larry Ellison	Electronics	55
Susan Wojcicki	null	300

Tim Cook	Widgets	48
Mark Zuckerberg	Housewares	95
Jeff Weiner	Electronics	28
Larry Page	Electronics	145
Dick Costolo	null	108
null	Videos	null
null	Minions	null

Using PostgreSQL on the CS Teaching Lab Machines

Here's something completely different. For this question, you are going to practice two skills: using PostgreSQL on a CS Teaching Lab machine, and submitting a file on MarkUs. Follow these instructions to complete this question:

1. Complete Parts 1 through 4 of the **Using PostgreSQL on the Teaching Labs** instructions on the **Lectures** page of Quercus.
2. Cut and paste your interaction for Part 4, from starting up psql to quitting it, into a new file called prep5.txt. (Your terminal should allow you to simply select and copy text from the terminal into your favourite text editor.)
3. Login to MarkUs, and submit prep5.txt for the assignment called "Prep5". Note that for this "assignment", you must submit individually, i.e., no groups can be formed.

Coordinating between the CS Teaching Labs and your own machine

Students who are working on their own machine sometimes get confused about how to coordinate the between their and the CS Teaching Labs. Suppose you are working on your own machine in a terminal window logged in to dbsrv1 (where you are running psql). At the same time, start up the editor of your choice on your own machine -- sublime, textmate, even an IDE like eclipse or wing. You can cut the output of your query from dbsrv1, and paste it into the editor running on your own machine. When you save it, the file will be saved to your machine. You can then start up a browser on your machine, log in to MarkUs, and upload the file. You don't have to be on the Teaching Labs to create, save, and upload the file.

It does need to be a text file

Make sure that what you save is not only named with the .txt extension that we require, but actually is a plain text file. For instance, you could create a file in Word, rename it from .docx to .txt, and it would *not* be a text file.