# SUBQUERIES IN SQL

Week 4 January 30, 2013

CS 386/586 Winter 2013

Lois Delcambre

# What is a subquery?

- A subquery is a query – surrounded by parentheses.

- In relational algebra, we do this all the time:
  Here's a query:

$$\sigma_{s.rating>4} sailors\ s$$

Put parentheses around it and apply another operator:

$$(\sigma_{s.rating>4} sailors\ s) \bowtie_{s.sid=r.sid} reserves\ r$$

$$\pi_{s.sid,\ s.name,\ r.day}((\sigma_{s.rating>4} sailors\ s) \bowtie_{s.sid=r.sid} reserves\ r\ )$$

# What about SQL?

- Sometimes … we can put parentheses around a complete SQL query and apply another operator:
  - union
  - except
  - intersect
- What else can we do?
  - Let's consider what kind of answers queries return.
  - Queries return a table of rows
    - How many rows?
      - sometimes zero (empty answer), sometimes one, sometimes many
    - How many columns?
      - it is specified by the query; it could be one or more

# SCALAR QUERIES (AND SUBQUERIES)

# Scalar queries (and thus, subqueries)

- Which queries are guaranteed to return exactly one row?

  Answer: aggregate queries

  ```
  select   max(salary), min(salary
  from     agent
  ```

- This query returns exactly one row.
  (If the table is empty, Null values are returned for max and min.)
- This query returns exactly one row and exactly one column. Such queries are called *scalar* queries

  ```
  select avg(salary)
  from   agent
  ```

# Back to SQL …
# where can we put scalar subqueries?

- Since scalar subqueries return just one value, we can put a scalar subquery in places where we would normally put an atomic value.

- Let's try it in the WHERE clause –
  we compare attribute values to constants all the time.

```
select *
from sailors s
where rating = 7
```

This is a constant. Exercise: replace it with a scalar subquery.

# Using a scalar subquery in place of an atomic value - examples

```
select  *
from    agent a
where a.salary = (select max(a2.salary) from agent a2)
```

Here we have a subquery.
It is enclosed in parentheses.
This is a scalar subquery – it returns one value – the maximum salary.

This query will return agents who make the maximum salary.
Is there at least one such agent?
Is there more than one?

# Use a scalar subquery in the where clause

- Exercise: write a query that lists all agents from Poland who have a security clearance equal to the maximum security clearance of agents from the city of Boston.

# Answer

```
select * from agent
where country = 'Poland' and clearance_id =
(select max(clearance_id) from agent where city = 'Boston')
```

| agent_id | first | middle | last | address | city | country | salary | clearance_id |
|---|---|---|---|---|---|---|---|---|
| 85 | Nick | *NULL* | Coeckx | 105 48th Avenue | Warsaw | Poland | 57933 | 5 |
| 99 | Charles | *NULL* | Mou | *NULL* | Warsaw | Poland | 71207 | 5 |
| 152 | Jason | *NULL* | Noel | 6 97th Avenue | Warsaw | Poland | 72403 | 5 |
| 237 | Serguie | *NULL* | Bikkenning | 6 55th Avenue | Warsaw | Poland | 67893 | 5 |
| 248 | George | *NULL* | Kuzas | 34 64th Avenue | Warsaw | Poland | 56593 | 5 |
| 280 | Roberto | *NULL* | Johnson | 3 86th Avenue | Warsaw | Poland | 89667 | 5 |

# Exercise: write these queries; then issue extra queries to check your answers

- Write an SQL query that finds teams where the meeting frequency is the maximum meeting frequency for all teams.
- Write an SQL query that finds agents whose salary is greater than two times the average salary of all agents
- Write an SQL query that finds missions where the access id is equal to the minimum access id of all missions and the mission status is not equal to the minimum mission status of all missions.

```
select * from team t
where t.meeting_frequency =
        (select max(meeting_frequency) from team)
```

| team_id | name | meeting_frequency |
|---|---|---|
| 1 | Renegade | weekly |
| 2 | Haberdash | weekly |
| 7 | FlyOnTheWall | weekly |
| 9 | BumbleBee | weekly |
| 16 | Vikings | weekly |
| 18 | SqueakyClean | weekly |
| 22 | Leadphut | weekly |
| 27 | Swing Voters | weekly |
| 28 | Cha Cha Cha | weekly |
| 29 | Ghost Hunters | weekly |
| 37 | Jester | weekly |
| 38 | Scorpion | weekly |

```
select *
from agent a1
where a1.salary > 2 * (select avg(salary) from agent a2)
order by a1.salary
```

| agent_id | first | middle | last | address | city | country | salary | clearance_id |
|---|---|---|---|---|---|---|---|---|
| 749 | Tim | B | Thune | *NULL* | Pittsburgh | USA | 173190 | 3 |
| 780 | James | M | DeMint | *NULL* | Pittsburgh | USA | 173190 | 3 |
| 702 | Richard | D | Salazar | *NULL* | Pittsburgh | USA | 173190 | 3 |
| 779 | Pat | F | Obama | *NULL* | Pittsburgh | USA | 173190 | 3 |
| 744 | Elizabeth | P | Coburn | *NULL* | Pittsburgh | USA | 173190 | 3 |
| 778 | Joseph | J | Isakson | *NULL* | Pittsburgh | USA | 173190 | 3 |
| 757 | Tom | B | Vitter | *NULL* | Pittsburgh | USA | 173190 | 3 |
| 748 | Mel | J | Martinez | *NULL* | Pittsburgh | USA | 173190 | 3 |
| 769 | Craig | R | Burr | *NULL* | Pittsburgh | USA | 173190 | 3 |
| 790 | Jack | M | Dayton | 346 RUSSELL | Miami | USA | 173692 | 4 |
| 719 | Mark | J | Lincoln | 355 DIRKSEN | Pittsburgh | USA | 178210 | 1 |
| 720 | Michael | H | McConnell | 361-A RUSSELL | Miami | USA | 181222 | 1 |
| 758 | John | L | Ensign | 364 RUSSELL | Miami | USA | 182728 | 4 |

```
select * from mission
where access_id = (select min(access_id) from mission) and
    mission_status != (select min(mission_status) from mission)
order by mission_id
```

| mission_id | name | access_id | team_id | mission_status |
|---|---|---|---|---|
| 11 | Guarded City | 1 | 20 | ongoing |
| 12 | Methedras | 1 | 15 | success |
| 38 | Maura | 1 | 14 | success |
| 43 | Bofur | 1 | 4 | success |
| 50 | Iron Comb | 1 | 30 | success |
| 53 | KARÁN | 1 | 7 | success |
| 54 | Fell Winter | 1 | 20 | success |
| 56 | Nob | 1 | 17 | ongoing |
| 70 | Narchost | 1 | 3 | success |
| 77 | Twofoot | 1 | 2 | ongoing |
| 102 | Tuckborough | 1 | 20 | ongoing |
| 111 | Singollo | 1 | 37 | success |
| 112 | Ambaróna | 1 | 2 | success |

# Atomic values in join clause? (If yes, can we use a scalar subquery?)

- try this (it has what we think of as a relational algebra select clause in the ON clause of a join):

```
select *
from sailors s join reserves r
        on s.sid = r.sid and s.rating = 9
```

  Here … 9 is a value in the query.
  Replace it with a scalar subquery (try min).

- Here's a join with boats and reserves and a scalar subquery:

```
select * from reserves r join boats b
   on r.bid = b.bid and color = (select max(color) from boats)
```

# Let's try replacing a scalar subquery with a subquery that is NOT scalar

```
select * from reserves r join boats b
        on r.bid = b.bid and color = (select color from boats)
```

Here we have a subquery that returns lots of rows.

**SQL error:**

ERROR: more than one row returned by a subquery used as an expression
**In statement:**

select * from reserves r join boats b on r.bid = b.bid and color = (select color from boats)

# Atomic values in the select clause

- Try this:

select a.first, a.last, 36 as age
from agent a

Here we see that we can introduce a constant into a query result. We just list it; we often give it a name.

What happens if you don't give this attribute a name? (Try it)

| first | last | age |
|---|---|---|
| Nick | Black | 36 |
| Bill | Bundt | 36 |
| Mathew | Cohen | 36 |
| Jim | Cowan | 36 |
| George | Fairley | 36 |
| Bill | Heeman | 36 |
| Andrew | James | 36 |
| Kristin | Delcambre | 36 |
| John | Johnston | 36 |
| George | Jones | 36 |
| Jim | Kieburtz | 36 |
| George | Launchbury | 36 |
| Chris | Leen | 36 |

# constants can appear in select; how about scalar subqueries?

Write a query that lists the first, last, and salary for all agents along with the minimum salary (of all agents) and the maximum salary (of all agents).

# Answer

select a.first, a.last, a.salary, (select min(salary) from agent) as min, (select max(salary) as max from agent)

from agent a

| first | last | salary | min | ?column? |
|---|---|---|---|---|
| Nick | Black | 50553 | 50008 | 366962 |
| Bill | Bundt | 50955 | 50008 | 366962 |
| Mathew | Cohen | 55920 | 50008 | 366962 |
| Jim | Cowan | 66554 | 50008 | 366962 |
| George | Fairley | 76396 | 50008 | 366962 |
| Bill | Heeman | 51564 | 50008 | 36696 |

Why does the final column not have a name?

# Try using a scalar subquery in HAVING

Write an SQL query that lists the boat id and the minimum rating of sailors that have reserved that boat where the minimum rating of the sailors is the minimum rating of all sailors.

# Try using a scalar subquery in HAVING

Write an SQL query that lists the boat id and the minimum rating of sailors that have reserved that boat.

```
select  r.bid, min(s1.rating)
from    sailors s1 join reserves r on s1.sid = r. sid
group by r.bid
having min(s1.rating) = (select min(s2.rating) from sailors s2)
```

Write queries to determine whether or not this is the correct query answer.

| bid | min |
|---|---|
| 102 | 1 |
| 103 | 1 |
| 101 | 1 |
| 104 | 1 |

# Scope inside subqueries

```
select  r.bid, min(s1.rating)
from    sailors s1 join reserves r on s1.sid = r. sid
group by r.bid
having min(s1.rating) = (select min(s2.rating) from sailors s2)
```

subquery

Notice the correlation names: s1, r and s2

We are using two different copies of sailors.

But we don't need to use different correlation names because the inner query has its own scope. Try it.

# Correlation names in previous query

```
select  r.bid, min(s1.rating)
from    sailors s1 join reserves r on s1.sid = r. sid
group by r.bid
having min(s1.rating) = (select min(s2.rating) from sailors s2)
```

compared to:

```
select  r.bid, min(s.rating)
from sailors s join reserves r on s.sid = r. sid
group by r.bid
having min(s.rating) = (select min(rating) from sailors)
```

This query works as well; the subquery works with its own copy of sailors.

# Where can we use scalar subqueries in SQL?

- A scalar subquery can appear anywhere a constant can appear:
  - In the WHERE clause
    this a.agent_salary = 5300
    versus a.agent_salary = (select max(salary) from agent)
  - In the HAVING clause
    compare HAVING count(*) > 5
    versus HAVING COUNT(*) > (select … from …)
  - In the SELECT clause (to introduce a constant)
    compare SELECT sname, 'good sailor' as rate FROM …
    versus SELECT sname, (select … from …) as rate FROM …
  - In the JOIN clause of the FROM clause (in the join condition)
    compare FROM tbl1 JOIN tbl2 on rating = 8
    versus FROM tbl1 JOIN tbl2 on rating = (select … from …)

# NONSCALAR SUBQUERIES IN THE FROM CLAUSE

# Can we use nonscalar subqueries?

- Nonscalar subqueries return tables (with > 1 row)
- Where do tables appear in SQL?
- In the from clause!
- We can put subqueries in the from clause.

# Example: subquery in the from

```
select p.first, p.last, p.city
from (select *
        from agent a
        where city = 'Paris') p
```

| first | last | city |
|---|---|---|
| Bill | Bundt | Paris |
| Andrew | James | Paris |
| George | Jones | Paris |
| Jonathan | Hammerstrom | Paris |
| George | van Santen | Paris |
| George | Day | Paris |
| Pete | Heinlein | Paris |
| John | Freitag | Paris |
| Tom | Lymar | Paris |
| Bill | Spadaro | Paris |
| Michail | Cushing | Paris |

Notice: we MUST use a correlation name for the subquery in the FROM – even if we don't intend to use it.

Question: why do I have p.first in the query – instead of a.first?

# Using nonscalar subqueries in the from clause

- Explain these two queries in English

  compare:
  SELECT sid
  FROM sailors

  with:
  SELECT sid
  FROM (select sid from reserves) as x

- Notice the correlation name (x) which is not used anywhere in the query. But it is required.

# Try using a scalar subquery in having clause & a (non-scalar) subquery in from

- Find the team id for teams that have the maximum number of members on their teams.

# Try using a scalar subquery in having clause & a (non-scalar) subquery in from

- Find the team id for teams that have the maximum number of members on their teams.

```
select tr.team_id
from teamrel tr
group by tr.team_id
having count(*) = (select max(mcnt) from
                        (select     count(*) as mcnt
                          from      teamrel tr2
                          group by  tr2.team_id) x)
order by tr.team_id
```

# Query answer (from preceding page):

| team_id |
|---|
| 1 |
| 2 |
| 3 |
| 7 |
| 8 |
| 10 |
| 11 |
| 13 |
| 14 |
| 15 |
| 16 |
| 17 |
| 23 |
| 24 |
| 27 |
| 28 |
| 31 |
| 32 |
| 40 |

Issue queries to check to see whether or not team 1 (and team 2) have the maximum number of members on their team.

# NEW PREDICATES IN WHERE

to use with subqueries – including nonscalar subqueries

# Predicates that work with tables in where clause

- Predicates that work on tables:
  - EXISTS <table> true if table is non-empty
  - NOT EXISTS <table> true if table is empty
- Additional comparators that work with tables:
  - IN <table> a.salary in (select salary from agent)
  - NOT IN <table> a.rating not in (select …)
- Additional predicates that work with standard comparators:
  - ALL <table> a.salary > all (select salary from agent)
  - NOT ALL <table> a.salary not > all (select …)
  - ANY <table> s.rating = any (select ….)
    SOME is a synonym for ANY
  - NOT ANY <table> s.age = NOT ANY (select ….)

# Meaning of SOME and ALL

```
SELECT  S.Number, S.Name
FROM    Salesperson S
WHERE   S.Name = SOME (SELECT C.Name
                       FROM    Customer;);
```

- For SOME, the expression must be true for at least one row in the subquery answer
  - ANY is an older form of SOME
- For ALL, the expression must be true for all rows in the subquery answer.

# SOME before a subquery:
# works with any scalar or non-scalar subquery

Syntax:

<attribute-name> <comparator> SOME | ANY | ALL <subquery>

can appear in the WHERE clause

```
SELECT  S.Number, S.Name
FROM    Salesperson S
WHERE   S.Name = SOME (SELECT C.Name FROM  Customer)
```

How many rows will the subquery return?

SOME evaluates to TRUE if S.Name matches at least one of the names returned from the subquery.

# Exercise using some or any

- List agents where their salary is greater than at least one agent from Boston

# Exercise

```
select a.agent_id, a.first, a.last
from agent a where a.salary > some
        (select salary from agent where city = 'Boston')
```

| agent_id | first | last |
|---|---|---|
| 3 | Mathew | Cohen |
| 4 | Jim | Cowan |
| 5 | George | Fairley |
| 14 | John | Johnston |
| 21 | Jim | Kieburtz |
| 22 | George | Launchbury |
| 24 | Chris | Leen |
| 27 | George | McNamee |

Write queries to see if
this query answer is correct.

# Exercise

Write a query with ALL in the where clause.

Explain what your query means in English.

# IN or NOT IN before a subquery

```
SELECT    C1.Number, C1.Name
FROM      Customer C1
WHERE     C1.Name IN
            (SELECT Name
             FROM  Salesman)
```

Any SQL query:

IN and NOT IN can appear in these forms:

<attribute-name> IN (subquery)

(<attrib-name$_1$>, …, <attrib-name$_n$>) IN (subquery)

or

<attribute-name> NOT IN (subquery)

(<attrib-name$_1$>, …, <attrib-name$_n$>) NOT IN (subquery)

# How many times do subqueries run?

- Look back at all of the subqueries that we've written in class so far today.

- How often do they need to be executed?
  - just once?
  - or, for a subquery in the WHERE clause, does it need to be executed every time the WHERE clause is evaluated?

- All of the queries we've seen so far – only need to be executed once. The answer to the subquery does not change – during the time that the query (that it is part of) is running.

# ALL – with non-correlated subquery

```
SELECT S.Number, S.Name
FROM   Salesperson S
WHERE   S.Number = ALL  (SELECT C.Salesperson
                         FROM     Customer C
                         WHERE  C.CRating = 3)
```

Notice that the inner query doesn't mention any attributes from the outer query. That is, S is not mentioned in the inner query.

In this case, you only need to evaluate the inner query once – because nothing changes when each tuple from the outer query is evaluated.

# This is a Correlated Subquery

```
SELECT  S.Number, S.Name
FROM    Salesperson S
WHERE   S.Number IN      (SELECT C.Salesperson
                          FROM   Customer C
                          WHERE C.Name = S.Name)
```

Because the subquery mentions an attribute from a table in the outer query

The subquery must be re-evaluated every time the WHERE clause from the outer query is evaluated.

You should look at the use of correlation names to figure out whether it is a correlated subquery.

# Subquery with "IN" – can be equivalent to a join

```
SELECT    S.Number, S.Name
FROM      Salesperson S
WHERE     S.Number IN    (SELECT  C.Salesperson
                          FROM    Customer C
                          WHERE   C.Name = S.Name)
```

```
SELECT    DISTINCT S.Number, S.Name
FROM      Salesperson S, Customer C
WHERE     S.Number = C.Salesperson AND C.Name = S.Name
```

Are these two queries equivalent?

Do we need to use the DISTINCT clause in the second query in order for these two queries to be equivalent?

# "IN" and "= SOME"

You can substitute = SOME for IN, and vice versa, to make an equivalent query, e.g.,

```
SELECT    C.Number, C.Name
FROM      Customer C
WHERE     C.address IN
          (SELECT S.address
           FROM    Salesman S)
```

```
SELECT   C.Number, C.Name
FROM     Customer C
WHERE    C.address = SOME
                     (SELECT S.address
                      FROM    Salesman S)
```

# Is this a correlated subquery? (query repeated from slide 6)

```
SELECT      C1.Number, C1.Name
FROM        Customer C1
WHERE       C1.CRating IN
                          (SELECT MAX (C2.CRating)
                           FROM    Customer C2)
```

What is the advantage of a subquery that is NOT correlated?

# EXISTS before a correlated subquery in a WHERE clause

```
SELECT    C.Name
FROM      Customer C
WHERE     EXISTS (SELECT    *
                  FROM      Salesperson S
                  WHERE     S.Number = C.Salesperson
                            AND S.Name = C.Name)
```

If the answer to the subquery is not empty - then the EXISTS predicate returns TRUE

# When can a query be correlated?

- In the select clause? Try this:

```
select a.agent_id,
 (select first from agent a2 where a.agent_id = a2.agent_id)
     as new
from agent a
```

| agent_id | new |
|---|---|
| 1 | Nick |
| 2 | Bill |
| 3 | Mathew |
| 4 | Jim |
| 5 | George |
| 7 | Bill |
| 8 | Andrew |

See … this is correlated.

Notice, this is a scalar subquery.
It must return a single value.

# Try writing a correlated subquery in the select

- try writing a correlated subquery in the select that returns more than one column.

- Try writing a correlated subquery in the select that returns more than one row.

# Correlated subqueries in the from clause

- You can't use a correlated subquery in the from clause

```
select  a, b, c, …
from    table1 t1, table2 t2, …, (select .. from … where…)
```

this subquery delivers a table that is used at the beginning of the outer query.

```
select  a, b, c, …
from    table1 t1, table2 t2, …
where … EXISTS (select … from … where t1.whatever = ..)
```

this subquery is in the where clause; the where clause is evaluated for every candidate combination that is delivered from the from clause.

# Challenge

- Write a query that finds for each team, the agent who speaks the most languages on that team.

# Challenge:

Write an SQL query that finds the sailors who do not have a reservation. (Do not use outer join.)

If you succeed with that, extend your query to introduce three Null values into the query answer – on the right side of the existing columns – one for each reservation attribute

If you succeed with that, extend your SQL query to compute the left outer join of sailors with reserves – without using the left outer join operator in SQL