CS122A: Introduction to Data Management

Lecture 9
SQL II: Nested Queries, Aggregation, Grouping

Instructor: Chen Li

Nested Queries

Find names of sailors who've reserved boat #103:

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid=103)
```

- A very powerful feature of SQL: a WHERE clause can itself contain an SQL query! (Actually, so can FROM and HAVING clauses!!)
- To find sailors who've *not* reserved #103, use NOT IN.
- To understand semantics (including cardinality) of nested queries, think *nested loops* evaluation: *For each Sailors tuple, check qualification by computing subquery.*

Nested Queries with Correlation

Find names of sailors who've reserved boat #103:

```
SELECT  S.sname
FROM  Sailors S
WHERE  EXISTS  (SELECT  *
                FROM  Reserves R
                WHERE  R.bid=103 AND S.sid=R.sid)
```

- EXISTS is another set comparison operator, like IN.
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple (conceptually).

NOTE: Recall that there was a join way to express this query, too. Relational query optimizers will try to unnest queries into joins when possible to avoid nested loop query evaluation plans.

More on Set-Comparison Operators

- We've already seen IN and EXISTS.. Can also use NOT IN and NOT EXISTS.
- Also available: *op* ANY, *op* ALL (for *ops:* >,<,=,≥,≤,≠)
- Find sailors whose rating is greater than that of some sailor called Horatio:

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ANY  (SELECT  S2.rating
                        FROM  Sailors S2
                        WHERE S2.sname= 'Horatio' )
```

Rewriting INTERSECT Queries Using IN

Find sid's of sailors who've reserved both a red and a green boat:

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color= 'red'
        AND S.sid IN  (SELECT  S2.sid
                       FROM  Sailors S2, Boats B2, Reserves R2
                       WHERE  S2.sid=R2.sid AND R2.bid=B2.bid
                              AND  B2.color= 'green' )
```

- Similarly, EXCEPT queries can be re-written using NOT IN.

Division, SQL Style

Find sailors who've reserved all boats.

(1)
```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS
        ((SELECT  B.bid
          FROM  Boats B)
        EXCEPT
          (SELECT  R.bid
           FROM  Reserves R
           WHERE  R.sid=S.sid))
```

(This Sailor's unreserved Boat ids..!.)

Sailors S such that ...

the set of all Boat ids ...

minus ...

this Sailor's reserved Boat ids...

*is **empty!***

Division in SQL (cont.)

(1)

```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS
          ((SELECT  B.bid
            FROM  Boats B)
           EXCEPT
            (SELECT  R.bid
             FROM  Reserves R
             WHERE  R.sid=S.sid))
```

Find sailors who've reserved all boats.

- Let's do it the hard way, i.e.., without EXCEPT:

(2)
```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS  (SELECT  B.bid
                    FROM  Boats B
                    WHERE  NOT EXISTS  (SELECT  R.bid
                                        FROM  Reserves R
                                        WHERE  R.bid=B.bid
                                        AND R.sid=S.sid))
```

Sailors S such that ...

***there is no** boat B **without** ...*

a Reserves tuple showing S reserved B

Aggregate Operators

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

single column

- Significant extension of relational algebra.

```
SELECT  COUNT (*)
FROM  Sailors S
```

```
SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=10
```

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.rating= (SELECT  MAX(S2.rating)
                  FROM  Sailors S2)
```

```
SELECT  COUNT (DISTINCT S.rating)
FROM  Sailors S
WHERE S.sname= 'Bob'
```

```
SELECT  AVG( DISTINCT S.age)
FROM  Sailors S
WHERE  S.rating=10
```

Find name and age of the oldest sailor(s)

- The first query is *illegal*! (We'll look into the reason a bit later, when we discuss GROUP BY.)
- The third query is equivalent to the second one, and allowed in the SQL/92 standard, but not supported in all systems.

~~SELECT S.sname, MAX (S.age)~~
~~FROM Sailors S~~

```
SELECT  S.sname, S.age
FROM  Sailors S
WHERE  S.age =
          (SELECT  MAX (age)
           FROM  Sailors)
```

```
SELECT  S.sname, S.age
FROM  Sailors S
WHERE  (SELECT  MAX (S2.age)
           FROM  Sailors S2)
           = S.age
```

Motivation for Grouping

- So far, we've applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several *groups* of tuples.
- Consider: *Find the age of the youngest sailor for each rating level.*
  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (☺):

For i = 1, 2, ... , 10:

```
SELECT  MIN (S.age)
FROM  Sailors S
WHERE  S.rating = i
```

Queries With GROUP BY and HAVING

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

- The *target-list* contains (i) attribute names and (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
  - The attribute list (i) must be a subset of *grouping-list*. Intuitively, each answer tuple corresponds to a *group*, and these attributes must have **a single value per group**. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

Conceptual Evaluation

- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, `*unnecessary*' fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- A *group-qualification* (HAVING) is then applied to eliminate some groups. Expressions in *group-qualification* must also have a *single value per group*!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op must appear in *grouping-list*. (Note: SQL does not consider primary key semantics here.)
- One answer tuple is generated per qualifying group.

Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors

```
SELECT  S.rating,  MIN (S.age)
                          AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) >= 2
```

Answer relation:

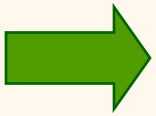
rating	minage
3	25.5
7	35.0
8	25.5

Sailors instance:

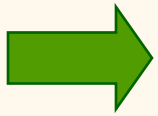
sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors.

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5



rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0



rating	minage
3	25.5
7	35.0
8	25.5

Find age of the youngest sailor with age ≥18, for each rating with at least 2 sailors between 18 and 60.

```
SELECT  S.rating,  MIN (S.age)
                          AS minage
FROM  Sailors S
WHERE  S.age >= 18 AND S.age <= 60
GROUP BY  S.rating
HAVING  COUNT (*) >= 2
```

Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

For each red boat, find the number of reservations for this boat

```
SELECT  B.bid,  COUNT (*) AS scount
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color= 'red'
GROUP BY  B.bid
```

- We're grouping over a join of three relations!
- What do we get if we remove *B.color= 'red'* from the WHERE clause and add a HAVING clause with this condition? (**Hint**: Trick question... ☺)
- What if we drop Sailors and the condition involving S.sid?

Find age of the youngest sailor with age > 18, for each rating with at least 2 sailors (of any age)

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age > 18
GROUP BY  S.rating
HAVING  1  <  (SELECT  COUNT (*)
                FROM  Sailors S2
                WHERE  S.rating=S2.rating)
```

- Shows HAVING clause can also contain a subquery.
- Compare this with the query where we considered only ratings with 2 sailors over 18!
- What if HAVING clause is replaced by:
  - HAVING COUNT(*) >1

Find those ratings for which the average age is the minimum age over all Sailors

- Aggregate operations cannot be nested! WRONG:

```
SELECT S.rating
FROM Sailors S
WHERE S.age = (SELECT MIN (AVG (S2.age)) FROM Sailors S2)
```

- Correct solution (in SQL/92):

```
SELECT Temp.rating, Temp.avgage
FROM (SELECT S.rating, AVG (S.age) AS avgage
      FROM Sailors S
      GROUP BY S.rating) AS Temp
WHERE Temp.avgage = (SELECT MIN (age) FROM Sailors)
```