SQL: Structured Query Language

Part IV

Find the age of the youngest sailor with age \geq 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

| sid | sname | rating | age |
|-----|---------|--------|------|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 71 | zorba | 10 | 16.0 |
| 64 | horatio | 7 | 35.0 |
| 29 | brutus | 1 | 33.0 |
| 58 | rusty | 10 | 35.0 |

| Rating | age |
|--------|------|
| 1 | 33.0 |
| 7 | 45.5 |
| 7 | 35.0 |
| 8 | 55.5 |
| 10 | 35.0 |

WHERE S.rating=S2.rating)

For rating=7

Count(*)

2

For rating=10

Count(*)

2

10

7 35.0 10 35.0

Shows HAVING clause can also contain a subquery.

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

• Only S.rating and S.age are mentioned in the SELECT, GROUP BY or HAVING clauses; other attributes `unnecessary'.

| Rating | age |
|--------|------|
| 7 | 45.0 |
| 8 | 55.5 |
| 7 | 35.0 |
| 1 | 33.0 |
| 10 | 35.0 |

| sid | sname | rating | age |
|-----|---------|--------|------|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 71 | zorba | 10 | 16.0 |
| 64 | horatio | 7 | 35.0 |
| 29 | brutus | 1 | 33.0 |
| 58 | rusty | 10 | 35.0 |

| Rating | age |
|--------|------|
| 1 | 33.0 |
| 7 | 45.5 |
| 7 | 35.0 |
| 8 | 55.5 |
| 10 | 35.0 |

| rating | |
|--------|------|
| 7 | 35.0 |

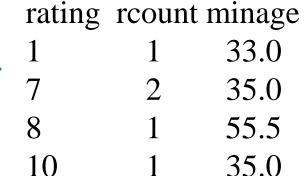
Answer relation

| sid | sname | rating | age |
|-----|---------|--------|------|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 71 | zorba | 10 | 16.0 |
| 64 | horatio | 7 | 35.0 |
| 29 | brutus | 1 | 33.0 |
| 58 | rusty | 10 | 35.0 |

SELECT Temp.rating, Temp.minage FROM

WHERE Temp.rcount > 1 Assume a TEMP storing number & min age of sailors >= 18 per rating

22 45.0 dustin 31 8 55.5 lubber 71 10 16.0 zorba 64 horatio 35.0 29 33.0 brutus 58 35.0 10 rusty



SELECT Temp.rating, Temp.minage FROM

(SELECT S.rating, COUNT(*) AS roount, MIN(S.age) AS minage

FROM Sailors S

WHERE S.age \geq 18

GROUP BY S.rating) AS Temp

WHERE Temp.rcount > 1 Assume a TEMP storing number

| sid | sname | rating | age |
|-----|---------|--------|------|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 71 | zorba | 10 | 16.0 |
| 64 | horatio | 7 | 35.0 |
| 29 | brutus | 1 | 33.0 |
| 58 | rusty | 10 | 35.0 |

& min age of sailors >= 18 per rating

35.()

rating rount minage



| 1 | 1 | 33.0 |
|-----|---|-------|
| 7 | 2 | 35.0 |
| 8 | 1 | 55.5 |
| 4.0 | _ | 2 - 0 |

6

RECALL: Find name and age of the oldest sailor(s)

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

- Aggregate operations cannot be nested!
- The following is 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

WHERE Temp.avgage = (SELECT MIN (Temp.avgage)

FROM Temp)

OK to define once on pen&paper

| sid | sname | rating | age |
|-----|---------|--------|------|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 71 | zorba | 10 | 16.0 |
| 64 | horatio | 7 | 35.0 |
| 29 | brutus | 1 | 33.0 |
| 58 | rusty | 10 | 35.0 |

| rating | avgage |
|--------|--------|
| 1 | 33.0 |
| 7 | 40.0 |
| 8 | 55.5 |
| 10 | 25.5 |

> 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 (Temp.avgage)

| FROM | Temp) |
|------|-------|
| | / |

OK to define once

on pen&paper

| <u>sid</u> | sname | rating | age |
|------------|---------|--------|------|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 71 | zorba | 10 | 16.0 |
| 64 | horatio | 7 | 35.0 |
| 29 | brutus | 1 | 33.0 |
| 58 | rusty | 10 | 35.0 |

| rating | avgage |
|--------|--------|
| 1 | 33.0 |
| 7 | 40.0 |
| 8 | 55.5 |
| 10 | 25.5 |

10

How about...?
 SELECT Temp.rating, MIN(Temp.avgage)
 FROM (SELECT S.rating, AVG (S.age) AS avgage
 FROM Sailors S

If the SELECT clause uses an aggregate operation, then it must use *only* aggregate operations <u>unless</u> the query contains <u>GROUP BY</u> clause

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

GROUP BY S.rating) AS Temp

GROUP BY Temp.rating

55.5

25.5

```
SELECT Temp.rating, Temp.avgage

FROM (SELECT S.rating, AVG (S.age) AS avgage What if you
FROM Sailors S can't use MIN?

GROUP BY S.rating) AS Temp

WHERE Temp.avgage = (SELECT MIN (Temp.avgage)

FROM Temp)

What if you
can't use TEMP?
```

| rating | avgage |
|--------|--------|
| 1 | 33.0 |
| 7 | 40.0 |
| 8 | 55.5 |
| 1.0 | 255 |

```
SELECT Temp.rating, Temp.avgage
 FROM (SELECT S.rating, AVG (S.age) AS avgage
                                               What if you
        FROM Sailors S
                                               can't use MIN?
        GROUP BY S.rating) AS Temp
 WHERE Temp.avgage = (SELECT MIN (Temp.avgage)
                        FROM Temp)
                                              What if you
 SELECT
            S.rating, AVG (S.age) AS avgage
                                             can't use TEMP?
             Sailors S
 FROM
GROUP BY S.rating
                                               33.0
            AVG (S.age) <= ALL
 HAVING
                                               40.0
                         (SELECT AVG(S2.age)
rating avgage
                                               55.5
                         FROM Sailors S2
       33.0
                                               25.5
                         GROUP BY S2.rating)
       40.0
       55.5
                                                       13
```

25 5

Null Values

- Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name).
 - SQL provides a special value <u>null</u> for such situations.
- The presence of *null* complicates many issues.
 - Special operators "IS NULL" and "IS NOT NULL".

Null Values: 3 valued logic

- Is *rating*>8 true or false when *rating* is equal to *null*? What about AND, OR and NOT connectives?
- We need a <u>3-valued logic</u> (true, false and *unknown*).

| sid | sname | rating | age |
|-----|-------|--------|-----|
| 21 | Dan | NULL | 38 |

- rating>8 can be true, false or unknown
- rating > 8 OR age < 40???
- rating > 8 OR age >40 ???
- rating > 8 AND age >40 ???

Null Values: 3 valued logic

- Is *rating*>8 true or false when *rating* is equal to *null*? What about AND, OR and NOT connectives?
- We need a <u>3-valued logic</u> (true, false and *unknown*).

| sid | sname | rating | age |
|-----|-------|--------|-----|
| 21 | Dan | NULL | 38 |

- rating>8 can be true, false or unknown
- rating > 8 OR age < 40 ???
- rating > 8 OR age >40 ???
- rating > 8 AND age >40 ???

| OR TABLE | | |
|----------|---------|---------|
| Т | T | T |
| T | F | T |
| F | T | T |
| T | Unknown | T |
| Unknown | T | T |
| F | F | F |
| F | Unknown | Unknown |
| Unknown | F | Unknown |
| Unknown | Unknown | Unknown |

Null Values: Aggregate Operators

- COUNT(*) \rightarrow counts also the null values
- All others (as below): discards NULL values

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

Outer Joins

- Left outer join for S and R:
 - Each S row without a matching R rows appears (once) in the result, with R columns including NULLs
 - SELECT S.sid, R.bid
 FROM Sailors S NATURAL LEFT OUTER JOIN Reserves R

Sailors

| <u>sid</u> | sname | rating | age |
|------------|--------|--------|------|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 71 | zorba | 10 | 16.0 |

Reserves

| sid | bid | date |
|-----|-----|------|
| 22 | 101 | ••• |
| 31 | 102 | |

| sid | bid |
|-----|------|
| 22 | 101 |
| 31 | 102 |
| 71 | null |

Modifying Tables – Insert

- Inserting a single row into a table
 - Attribute list can be omitted if it is the same as in CREATE TABLE
 - NULL and DEFAULT values can be specified

INSERT INTO Transcript(StudId, CrsCode, Semester, Grade) VALUES (12345, 'CSE305', 'S2000', NULL)

Bulk Insertion

Insert the rows output by a SELECT

```
CREATE TABLE DeansList (

StudId INTEGER,

Credits INTEGER,

CumGpa REAL,

PRIMARY KEY (StudId))
```

```
INSERT INTO DeansList (StudId, Credits, CumGpa)

SELECT T.StudId, 3 * COUNT (*), AVG(T.Grade)

FROM Transcript T

GROUP BY T.StudId

HAVING AVG (T.Grade) > 3.5 AND

COUNT(*) > 30
```

20

Modifying Tables – Delete

- Similar to SELECT except:
 - No project list in DELETE clause
 - No Cartesian product in FROM clause (only 1 table name)
 - Rows satisfying WHERE clause (general form, including subqueries, allowed) are deleted instead of output

DELETE FROM Transcript T
WHERE T. Grade IS NULL AND T. Semester <> 'F2020'

Modifying Data - Update

```
UPDATE Employee E

SET E.Salary = E.Salary * 1.05

WHERE E.Department = 'R&D'
```

- Updates rows in a single table
- All rows satisfying WHERE clause (general form, including subqueries, allowed) are updated

Test first!

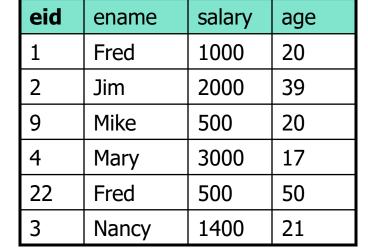
UPDATE Employee E

SET E.Salary * 5

WHERE E.Salary <= (SELECT AVG(E2.salary)

FROM Employee E2)





Avg: 8400/6 = 1400