SQL: Structured Query Language

Part III

More on Set-Comparison Operators

- We’ve already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL   >,<,=,≥,≤,≠
- *Find sailors whose rating is greater than some sailor called Jim:*

sid	sname	rating	age
1	Fred	7	20
2	Jim	2	39
9	Mike	7	20
4	Jim	1	17
22	Fred	7	50
3	Nancy	1	21

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

If the subquery returns an **empty set**, comparison returns **FALSE**

More on Set-Comparison Operators

- *Find sailors whose rating is greater than every sailor called Jim.*

sid	sname	rating	age
1	Fred	7	20
2	Jim	2	39
9	Mike	7	20
4	Jim	1	17
22	Fred	7	50
3	Nancy	1	21

```
SELECT *
FROM   Sailors S
WHERE  S.rating > ALL
                      (SELECT  S2.rating
                       FROM  Sailors S2
                       WHERE S2.sname=‘Jim')
```

If the subquery returns an empty set, comparison returns TRUE

More on Set-Comparison Operators

- *Find sailors with highest rating.*

sid	sname	rating	age
1	Fred	7	20
2	Jim	2	39
9	Mike	7	20
4	Jim	1	17
22	Fred	7	50
3	Nancy	1	21

```
SELECT  *
FROM  Sailors S
WHERE  S.rating >= ALL  (SELECT  S2.rating
                          FROM  Sailors S2)
```

Note: ***IN*** *equivalent to* ***= ANY***

NOT IN *equivalent to* ***<> ALL***

Division in SQL

Find sailors who’ve reserved all boats.

$$\rho\ (Tempsids, (\pi_{sid,bid}\mathrm{Re}serves)\ /\ (\pi_{bid}\ Boats))$$

$$\pi_{sname}\ (Tempsids \bowtie Sailors\ )$$

Let’s think in a complementary way.

Sailors without any unreserved boat.

Division in SQL

Find sailors who’ve reserved all boats.

```
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

(for which there is no reservation tuple)

Subquery in the green box:

The boats that are **not reserved** by the given sailor!

sid	name
1	fred
2	wilma

bid	color
101	red
102	green

Answer:

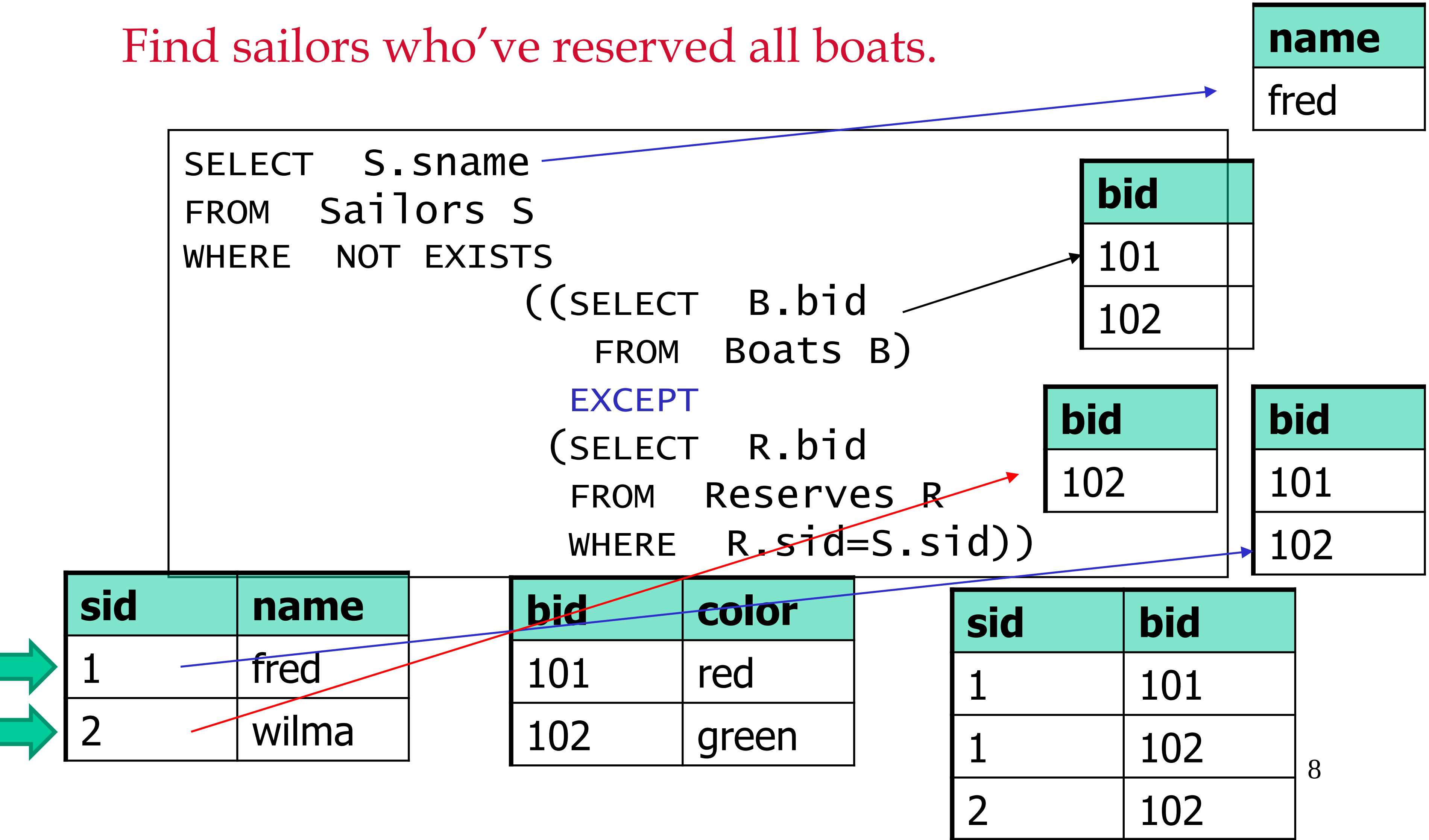
sid
Fred

sid	bid
1	101
1	102
2	102

```
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 ))
```

Division using EXCEPT

Find sailors who've reserved all boats.



```
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))
```

name
fred

bid
101
102

bid
102

bid
101
102

sid	name
1	fred
2	wilma

bid	color
101	red
102	green

sid	bid
1	101
1	102
2	102

Aggregate Operators

- Significant extension of relational algebra.

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

single column

Aggregate Operators

sid	sname	rating	age
1	Fred	7	20
2	Jim	2	39
9	Mike	7	20
4	Mary	1	17
22	Fred	7	50
3	Nancy	2	21

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

```
SELECT  COUNT (DISTINCT S.rating)
FROM  Sailors S
WHERE S.sname=‘Fred’
```

```
SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=7
```

```
SELECT  AVG ( DISTINCT S.age)
FROM  Sailors S
WHERE  S.rating=7
```

Aggregate Operators

Find the names of the sailors with the highest rating.

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

sid	sname	rating	age
1	Fred	7	20
2	Jim	2	39
9	Mike	7	20
4	Mary	1	17
22	Fred	7	50
3	Nancy	2	21

Find name and age of the oldest sailor(s)

- The first query is illegal: If the SELECT clause uses an aggregate operation, then it must use *only* aggregate operations unless the query contains GROUP BY clause)

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

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

GROUP BY and HAVING

- 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
```

Find the age of the youngest sailor for each rating level

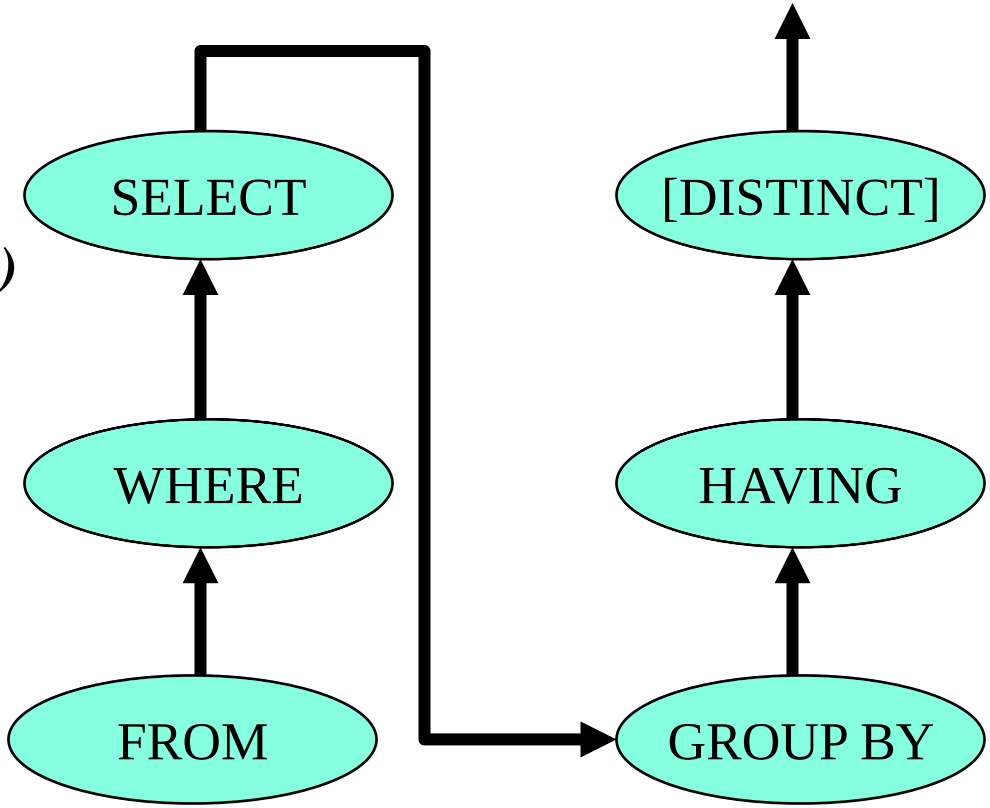
```
SELECT  S.rating, MIN (S.age)
FROM  Sailors S
GROUP BY  S.rating;
```

Find the age of the youngest sailor for each rating level with at least 2 sailors

```
SELECT  S.rating, MIN (S.age)
FROM  Sailors S
GROUP BY  S.rating
HAVING COUNT(*) >1;
```

Conceptual Evaluation

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



Project away columns (just keep those used in SELECT, GBY, HAVING)

Apply selections (eliminate rows)

Relation cross-product

Eliminate duplicates

Eliminate groups

Form groups & aggregate

Find the age of the youngest sailor for each rating level with at least 2 sailors

SELECT S.rating, MIN (S.age)
FROM Sailors S
GROUP BY S.rating
~~HAVING S.age > 30~~

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

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

Expressions in *group-qualification* must have a *single value per group*!

One answer tuple is generated per qualifying group.

Find the age of the youngest sailor for each rating level with at least 2 sailors

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

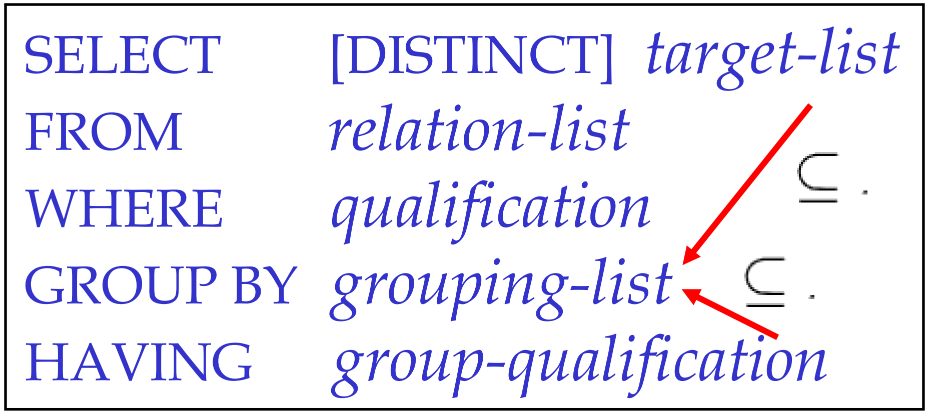
Equivalently…

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

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

- Shows HAVING clause can also contain a subquery.

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 (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*.
- The *group-qualification* is then applied to eliminate some groups. Expressions in *group-qualification* must have a *single value per group*!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*.
- One answer tuple is generated per qualifying group.

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

```
SELECT  B.bid,  COUNT (*) AS scount
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid AND B.color=‘red’
GROUP BY  B.bid
```

- Grouping over a join of two relations.

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

Reserves

sid	bid	date
1	101	1/1/2017
1	102	15/1/2017
2	108	3/3/2016
1	101	5/6/2016
1	108	4/4/2017

Boats

bid	color
101	red
102	green
108	red

sid	R.bid	date	B.bid	color
1	101	…	101	red
1	102		102	green
2	108		108	red
1	101		101	red
1	108		108	red

sid	R.bid		B.bid	color
1	101	…	101	red
1	101	…	101	red
2	108	…	108	red
1	108	…	108	red

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

```
SELECT  B.bid,  COUNT (*) AS scount
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid AND B.color=‘red’
GROUP BY  B.bid
```

- Grouping over a join of two relations.
- What do we get if we remove *B.color=‘red’* from the WHERE clause and add a HAVING clause with this condition?

Find the 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 )
```

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

For rating=7
Count(*)
2

For rating=10
Count(*)
2

rating	
7	35.0
10	35.0

- Shows HAVING clause can also contain a subquery.