$\frac{\text{Rewriting } \textit{Minus } \text{Queries Using }}{\textit{Not In}}$

SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid = R.sid and R.bid = B.bid
and B.color = red
MINUS
SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid = R.sid and R.bid = B.bid
and B.color = green;

Rewriting <u>Minus</u> Queries Using <u>Not In</u>

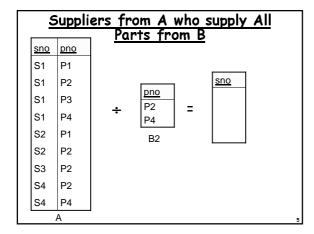
Division

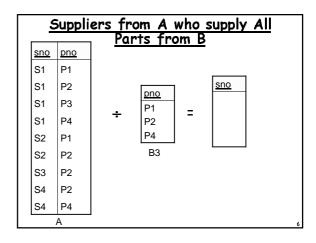
· Consider: A(X,Y) and B(Y).

Then A+B =

$$\{\langle x \rangle | (\forall \langle y \rangle \in B) | \langle x, y \rangle \in A\}$$

• In general, we require that the set of fields in B be contained in those of A.





Sailors who Reserved all Boats

· To find the Sailors who reserved all boats:

```
(\pi_{\text{sid,bid}} \text{ Reserves}) \div (\pi_{\text{bid}} \text{ Boats})
```

 Division can be expressed using other relational algebra operators. How?

Similar to Expression Containment

- Sailor S whose "set of boats reserved" contains the "set of all boats"
- Sailor S for which there does not exist a boat B in Boats that he did not reserve
- Sailor S for which there does not exist a boat B in Boats for which there is no reservation in Reserves

Division in SQL (1)

```
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 in SQL (2)

```
SELECT S.sname

FROM Sailors S

WHERE NOT EXISTS((SELECT B.bid

FROM Boats B)

MINUS

(SELECT R.bid

FROM Reserves R

WHERE R.sid = S.sid));
```

Aggregation

Aggregate Operators

- The aggregate operators available in SQL
 - COUNT(*)
 - COUNT([DISTINCT] A)
 - SUM([DISTINCT] A)
 - AVG([DISTINCT] A)
 - -MAX(A)
 - -MIN(A)

2

Some Examples

SELECT COUNT(*)
FROM Sailors S

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

SELECT COUNT(distinct color)
FROM Boats

Find Average Age for each Rating

- So far, aggregation has been applied to all tuples that passed the WHERE clause test.
- How can we apply aggregation to groups of tuples?

Basic SQL Query

SELECT [Distinct] target-list

FROM relation-list

WHERE condition

GROUP BY grouping-list

HAVING group-condition;

- <u>target-list:</u> Fields appearing in grouping-list and aggregation operators
- <u>group-condition:</u> Can only constrain attributes appearing in grouping-list

Evaluation

- 1. Compute cross product of relations in FROM
- 2. Tuples failing WHERE are thrown away
- 3. Tuples are partitioned into groups by values of *grouping-list* attributes
- 4. The *group-condition* is applied to eliminate groups
- 5. One answer in generated for each group

Find Average Age for each Rating

<u>Sailors</u>				
<u>sid</u>	sname	rating	age	
22	Dustin	7	45.0	
31	Lubber	8	55.5	
58	Rusty	10	35.0	
63	Fluffy	7	44.0	
78	Morley	7	31.0	
84	Popeye	10	33.0	

<u>Sailors</u>				
<u>sid</u>	sname	rating	age	
22	Dustin	7	45.0	
63	Fluffy	7	44.0	
78	Morley	7	31.0	
31	Lubber	8	55.5	
58	Rusty	10	35.0	
84	Popeye	10	33.0	
_	10 55	5.5	34	

3

Find name and age of oldest Sailor

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

Wrong!!

SELECT S.sname, S.age
FROM Sailors S
WHERE S.age =

(SELECT MAX(S2.age)
FROM Sailors S2)

Right!!
How else can this be done?
Hint: >= ALL

What does this return?

SELECT B.bid, COUNT(*)

FROM Boats B, Reserves R

WHERE R.bid=B.bid and B.color= red

GROUP BY B.bid

?

What would happen if we put the condition about the color in the HAVING clause?

Names of Boats that were not Reserved on more than 5 days

Can we move the condition in the HAVING to the WHERE?

The Color for which there are the most boats

SELECT color
FROM Boats B
GROUP BY color
HAVING max(count(bid))

What is wrong with this?
How would you fix it?

Aggregation Instead of Exists

- · Aggregation can take the place of exists.
- Example:

SELECT color
FROM Boats B1
WHERE not exists(
SELECT *
FROM Boats B2
WHERE B1.bno<> B2.bno
and B1.color=B2.color)

Aggregation Instead of Exists

SELECT color FROM Boats B1 GROUP BY color HAVING count(bid) = 1

24

Sub-queries and Views

A Complex Query

- · We would like to create a table containing 3 columns:
 - Sailor id
 - Sailor age
 - Age of the oldest Sailor

How can this be done?

Attempt 1

SELECT S.sid, S.age, MAX(S.age) FROM Sailors S;

Why is this wrong?

Attempt 2

SELECT S.sid, S.age, MAX(S.age) FROM Sailors S GROUP BY S.id, S.age;

Why is this wrong?

Solution 1: Subquery in FROM

SELECT S.sid, S.age, M.mxage FROM Sailors S,(SELECT MAX(S2.age) as mxage FROM Sailors S2) M;

- · We can put a query in the FROM clause instead of a
- · The query in the FROM clause must be renamed with a range variable (M in this case).

Solution 2: Subquery in SELECT

SELECT S.sid, S.age, (SELECT MAX(S2.age) FROM Sailors S2)

FROM Sailors S;

• A query in the SELECT clause must return at most one value for each row returned by the outer query.

Another Example of a Sub-query in SELECT

SELECT S.sid, S.age, (SELECT MAX(S2.age)
FROM Sailors S2
WHERE S2.age<S.age)

FROM Sailors S;

- · What does this query return?
- \bullet Note the use of S (defined in the outer query) within the inner query.

Another Example of a Sub-query in FROM??

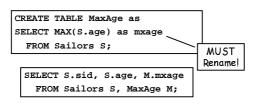
?

Why is this wrong?

ا ...

Solution 3: Create a Table

 A View is a query that looks like a table and can be used as a table.



Views

- A view is a "virtual table" defined using a query
- You can use a view as if it were a table, even though it doesn't contain data
- The view is computed every time that it is referenced

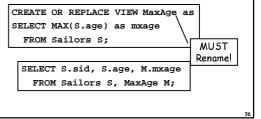
. .

Advantages and Disadvantages

- · Advantages:
 - no memory used for table
 - update of table does not require updating views
 - gives query processor more choices for optimizing
- Disadvantages:
 - must be recomputed every time used
 - if tables that view uses are dropped, view data is lost

Solution 4: Views

 A View is a query that looks like a table and can be used as a table.



Views For Restricting Access

• Suppose that we have a table:

Grades(Login, Exercise, Grade)

 We would like a user to only be able to see his own grades. We create the following view and grant privileges to query the view (not the underlying table)

```
CREATE OR REPLACE VIEW UserGrades as
SELECT *
FROM Grades
WHERE Login = User;
Pseudo-column
which is equal to
the user name.
```