SQL Examples

SELECT	[DISTINCT] *target-list*
FROM	*relation-list*
WHERE	*qualification*

- *qualification*
  - Comparisons (***Attr op const*** or ***Attr1 op Attr2***, where op is one of $<, >, =, \leq, \geq, \neq$) combined using *AND*, *OR* and *NOT*.
- *DISTINCT*
  - an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are ***not*** eliminated.

- Example: Find the names of all branches in the loan relation.

```
SELECT branch-name
FROM Loan
```

Loan

branch_name	loan_number	amount
ScotiaBank	222	2
RoyalBank	333	1
ScotiaBank	777	2

Result

branch_name
ScotiaBank
RoyalBank
ScotiaBank

- To remove duplications

```
SELECT DISTINCT branch-name
FROM Loan
```

Loan

branch_name	loan_number	amount
ScotiaBank	222	2
RoyalBank	333	1
ScotiaBank	777	2

Result

branch_name
ScotiaBank
RoyalBank

- Example (Q1, p.137): Find the names of sailors who have reserved boat number 103.

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Instance R3 of Reserves

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Instance S4 of Sailors

Find names of the sailors who have reserved boot number 103

Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND
R.bid=103
```

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Row remains after selection.

Result

S4 X R3

More Examples

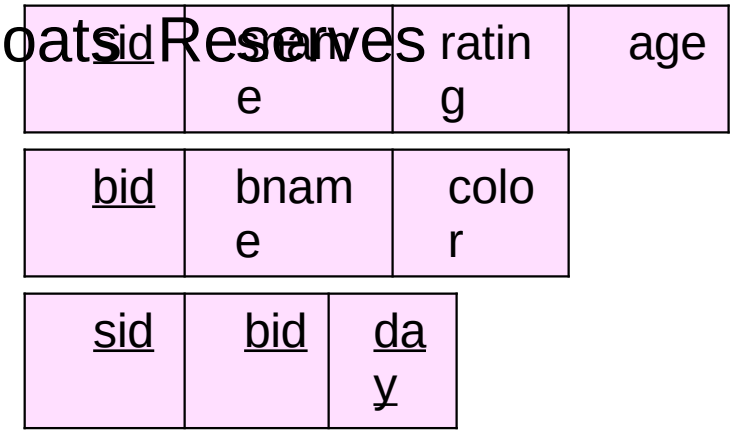
- Given the following schema:

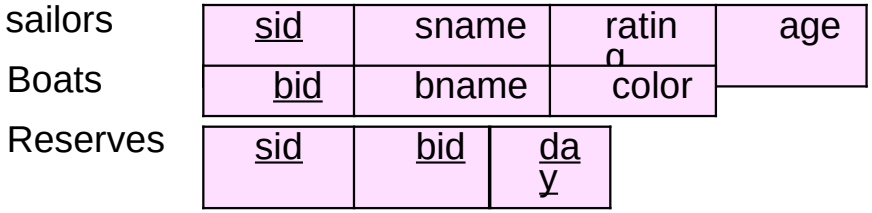
Sailors(*sid*; integer, *sname*: string, *rating*:integer, *age*: real)

Boats(*bid*: integer, *bname*; string, *color*: string)

Reserves(*sid*: integer, *bid*: integer, *day*: date)

sailors Boats Reserves



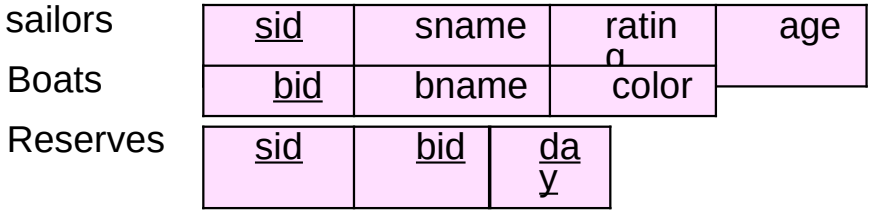


Example: Find the sids of sailors who have reserved a red boat.

```
SELECT R.sid
FROM  Boat B, Reserves R
WHERE  B.bid = R.bid AND  B.color = ‘red’
```

Example: Find the names of sailors who have reserved a red boat.

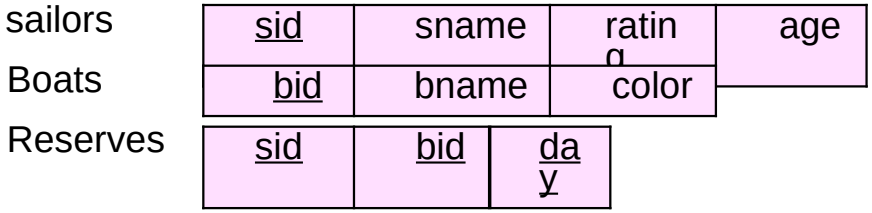
```
SELECT S.sname
FROM  Sailors S, Reserves R, Boat B
WHERE S.sid = R.sid AND  R.bid = B.bid
 AND   B.color = ‘red’
```



Example: Find the colors of boats reserved by Lubber.

```
SELECT B.color
FROM Sailors S, Reserves R, Boat B
WHERE S.sid = R.sid AND R.bid = B.bid
  AND  S.name = 'Lubber'.
```

(In general, there may be more than one sailor called Lubber. In this case, it will return the colors of boats reserved by some Lubber).



Example: Find the names of sailors who have reserved at least one boat.

```
SELECT S.name
FROM Sailors S,
Reserves R
WHERE S.sid = R.sid
```

(If a sailor has not made a reservation, the second step in the conceptual evaluation strategy would eliminate all rows in the cross-product that involve this sailor).

Expressions and Strings

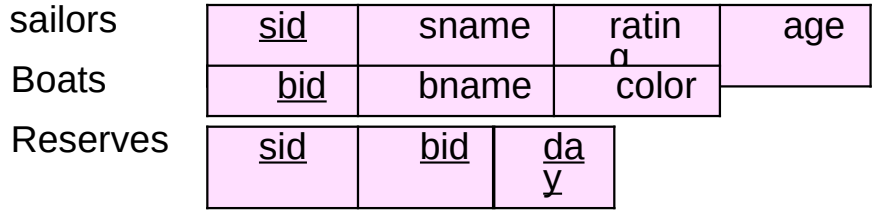
```
SELECT S.age, age1=S.age-5, 2*S.age
AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching: *Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.*
- AS and = are two ways to name fields in result.
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.

Union, Intersect, and Except

- SQL provides three set-manipulation constructs that extend the basic query form presented earlier.
  - Union (∪)
  - Intersection (∩)
  - Except (-)

(many systems recognize the keyword MINUS for EXCEPT)

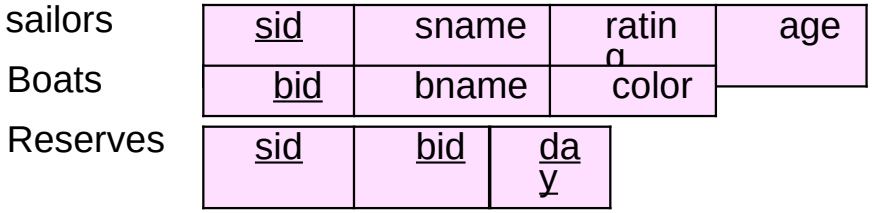


Example: Find sid's of sailors who've reserved a red or a green boat

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- If we replace OR by AND in the first version, what do we get?
- Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?)

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

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
      AND B.color='red'
UNION SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
      AND B.color='green'
```



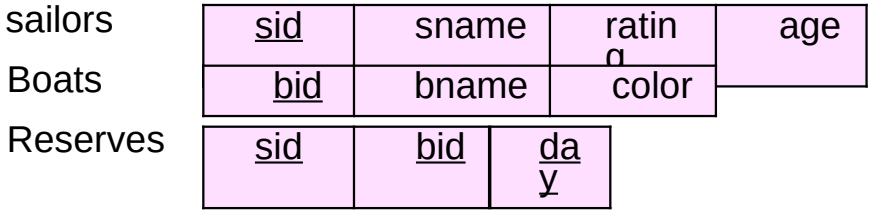
Example: Find sid's of sailors who've reserved a red and a green boat

- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.

- Included in the SQL/92 standard, but some systems don't support it.

```
SELECT S.sid
FROM Sailors S, Boats B1, Reserves
        R1,  Boats B2, Reserves R2
WHERE S.sid=R1.sid AND
 R1.bid=B1.bid  AND S.sid=R2.sid
 AND R2.bid=B2.bid  AND
 (B1.color='red' AND
```

```
SELECT S.sid
FROM Sailors S, Boats B,
Reserves  R
WHERE S.sid=R.sid AND
R.bid=B.bid
          AND B.color='red'
INTERSECT  SELECT S.sid
FROM Sailors S, Boats B,
Reserves  R
WHERE S.sid=R.sid AND
```

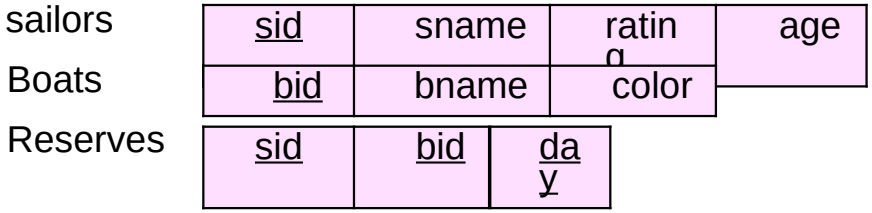


Example: Find sid's of all sailors who've reserved red boat but not green boat.

Indeed, since the Reserves relation contains sid information, there is no need to look at the Sailors relation.

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND
B.color='red'
EXCEPT
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND
B.color='green'
```

```
SELECT S.sid
FROM Sailors S, Boats B,
Reserves R
WHERE S.sid=R.sid AND
R.bid=B.bid
        AND B.color='red'
EXCEPT  SELECT S.sid
FROM Sailors S, Boats B,
Reserves R
WHERE S.sid=R.sid AND
R.bid=B.bid
        AND B.color='green'
```



Example: Find sid's of all sailors who have a rating of 10 or reserved boat 104

```
SELECT S.sid
FROM Sailor S
WHERE S.rating = 10
UNION  SELECT R.sid
FROM Reserves R
WHERE R.bid=104
```

Nested Queries

- A nested query is a query that has another query embedded within it.
- The embedded query is called a subquery.
- The embedded query can be a nested query itself.
  - Queries may have very deeply nested structures.

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

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE S.sid=R.sid AND
R.bid=103
```

alternative:

```
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.)

Consider: Find names of sailors who've not reserved b

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE S.sid=R.sid AND
R.bid≠103
```

correct?

correct answer:

```
SELECT S.sname  FROM Sailors S
WHERE S.sid NOT IN (SELECT R.sid
                    FROM Reserves R
                    WHERE R.bid=103)
```

- To understand semantics of nested queries, think of a *nested loops* evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*

Correlated Nested Queries

- In the previous example, the inner subquery has been completely independent of the outer query.
- In general, the inner subquery could depend on the row currently being examined in the outer query.

Example: Find names of sailors who've reserved

```
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, which allows us to test whether a set is nonempty.

Sailors

	sid	sname	rating	age
S →	22	dustin	7	45.0
→	31	lubber	8	55.5
S →	58	rusty	10	35.0
S				

Reserves

	sid	bid	day
R →	22	103	10/10/96
→	31	101	11/12/96
R →			
R	22	103	12/12/03
	58	105	8/21/05

Example: Find names of sailors with at most one reservation for boat #103

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

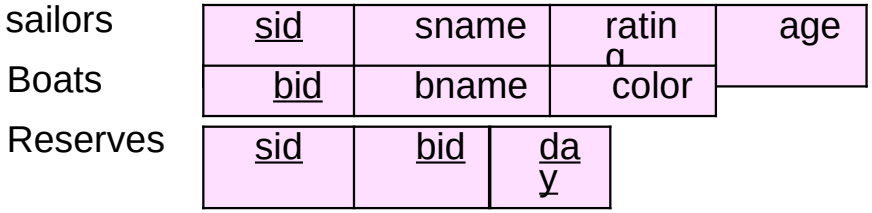
EXIST UNIQUE evaluates to true if the subquery returns a relation that contains no duplicated tuples (empty is a special case)

- Why do we have to replace * by *R.bid*?

Set-comparison Operators

- We've already seen IN, EXISTS and UNIQUE.
  We can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL
  - Where op is one of the arithmetic comparison operator
    $>, <, =, \geq, \leq,$
  - SOME is also available, but it is just a synonym for ANY.
- Example: 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')
```



Example: Find sailors whose rating is better than every sailor called Horatio.

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

Example: Find the sailors with the highest rating.

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

- Rewriting INTERSECT queries using IN

Example: 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.
- To find *names* (not *sid*'s) of Sailors who've reserved both red and green boats, just replace *S.sid* by *S.sname* in SELECT clause.

Division in SQL

Example: Find names of 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))
```

All boats

All boats reserved by S

sailors

sid	sname	rating	age

Boats

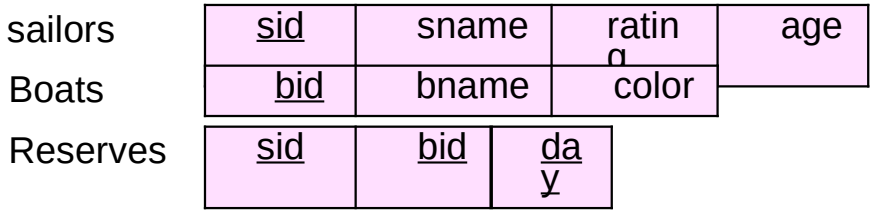
bid	bname	color

Reserves

sid	bid	day

Note that this query is correlated – for each sailor S, we select it if

there does not exist a boat that has not been reserves by him/her.



An Alternative way to write the previous query without using EXCEPT

```
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))
```

Again, for each sailor we check that there is no boat that has not been reserved by him/her:

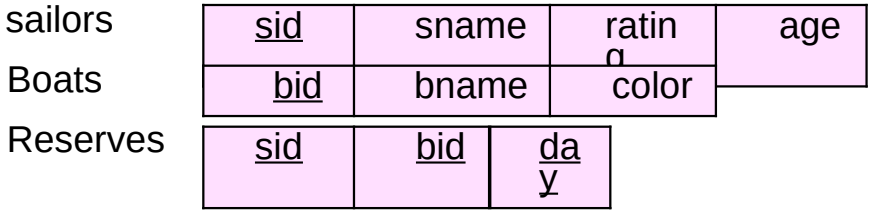
The 3rd level query returns empty set if B.bid is not reserved by S.sid

The 2nd level query returns empty set if no such B.bid exists

Aggregate Operators

- SQL allows the use of arithmetic expressions.
- SQL supports five aggregate operations, which can be applied on any column of a relation.

COUNT([DISTINCT] A)	The number of (unique) value in the A column.
SUM ([DISTINCT] A)	The sum of all (unique) values in the A column.
AVG ([DISTINCT A)	The average of all (unique) values in the A column.
MAX (A)	The maximum value in the A column.
MIN (A)	The minimum value in the A column.

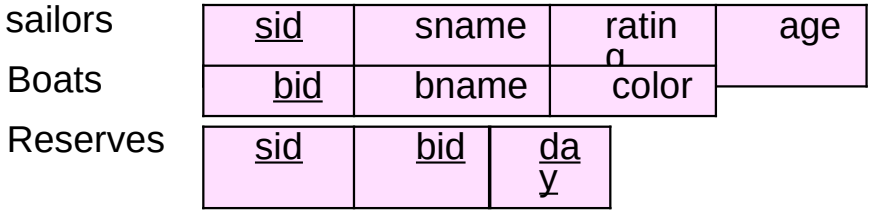


Example: *Count the number of Sailor*

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

Example: *Count the number of different sailor names*

```
SELECT  COUNT (DISTINCT S.name)
FROM    Sailors S
```



Example: Find the average age of all sailors

```
SELECT  AVG (S.age)
FROM     Sailors S
```

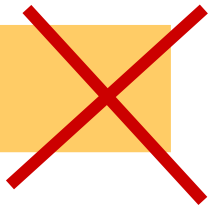
Example: Find the average age of sailors with rating of 10

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

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

Example: Find the name and age of the oldest sailor

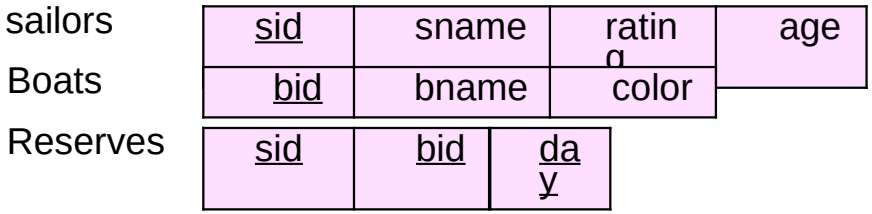
```
SELECT  S.sname, MAX
(S.age)
FROM    Sailors S
```



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

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

Equivalent to the second query, and is allowed in the SQL/92 standard, but is not supported in some systems.



Aggregate operations offer an alternative to the ANY and ALL constructs.

Example: *Find the names of sailors who are older than the oldest sailor with a rating of 10.*

```
SELECT  S.name
FROM    Sailors S
WHERE   S.age > ANY (SELET S2.age
            FROM Sailors S2
            WHERE S2.rating = 10)
```

Alternative

```
SELECT
S.name   FROM
Sailors     S       SELECT MAX
WHERE       S.age   (S2.age)
> (                 Sailors S2  S2.rating
            FROM    =10)
            WHER
```

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 $i = 1, 2, \ldots, 10$:

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

- To write such queries, we need a major extension to the basic SQL query form, namely the Group BY clause.
- The extension also includes an optional HAVING clause that can be used to specify qualifications over groups.

The query can be expressed as follows

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

The general format of GROUP BY and Having

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

- The *target-list* contains (i) attribute list (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

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

- 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*. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

sailors

sid	sname	rating	age

Boats

bid	bname	color

Reserves

sid	bid	day

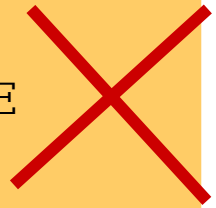
For each red boat, find its bid, and the number of reservations

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

- **Only B.bid is mentioned in the SELECT clause; other attributes are `*unnecessary*'.**
- **COUNT(*) is renamed**

alternative?

```
SELECT B.bid, COUNT (*) AS reservationcount
FROM Boats B, Reserves R WHERE R.bid=B.bid GROUP BY B.bid
HAVING B.color = 'red'
```



Only columns that appear in the GROUP BY clause can appear in the HAVING clause, unless they appear as arguments to an aggregate operator in the HAVING clause.

Example: Find the average age of sailors for each rating level that has at least two sailors.

```
SELECT S.rating, AVG(S.age) AS
avgage  FROM Sailor S
GROUP BY S.rating  HAVING
COUNT (S.sid) > 1
```

OR

```
SELECT S.rating, AVG(S.age) AS
avgage  FROM Sailor S
GROUP BY S.rating
HAVING 1  < (SELECT COUNT
(S2.sid)
      FROM Sailors S2
      WHERE S.rating = S2.rating)
```

Rating	avgage
3	44.5
7	40.0
8	40.5
10	25.5

Answer

We can use S.rating inside the nested subquery in the HAVING because it has a single value for the current group of sailors

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

Instance of Sailor

Example: Find the average age of sailors who are at least 18 years old for each rating level that has at least two sailors.

```
SELECT S.rating, AVG(S.age) AS
avgage  FROM Sailor 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
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	Bo0b	3	63.5
96	Frodo	3	25.5

Note that the answer is very similar to the previous one, with the only difference being that for the group 10, we now ignore the sailor with age 16 while computing the average.

Rating	avgage
3	44.5
7	40.0
8	40.5
10	35.5

Answer

of instance S3

Example: Find the average age of sailors who are at least 18 years old for each rating level that has at least two *such* sailors.

```
SELECT S.rating, AVG(S.age) AS
avgage  FROM Sailor S
WHERE S.age >=18
GROUP BY S.rating
HAVING COUNT(S.sid) > 1
```

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	Bo0b	3	63.5
96	Frodo	3	25.5

of instance S3

It differs from the answer of the previous question in that there is no tuple for rating 10, since there is only one tuple with rating and age >= 18.

Rating	avgage
3	44.5
7	40.0
8	40.5

Answer

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

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

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.

rating	age
1	33.
29	brutus	1	33.
7	45.
58	rusty	10	35.
7	35.

rating	
7	35. 35.

Answer relation

Some other ways to write: Find the average age of the sailors with age >= 18, for each rating with at least 2 such sailors

```
SELECT S.rating, AVG(S.age) AS avgage
FROM Sailor S  WHERE S.age >=18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT (*)
                FROM Sailor S1
                WHERE S1.age >= 18 AND S1.rating =
                S.rating)
```

alternative

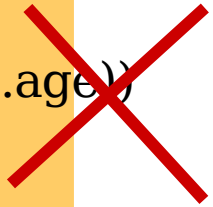
```
SELECT Temp.rating, Temp.avgage
FROM   (SELECT S.rating, AVG(S.age) AS
avgage,
                    COUNT (*) As ratingcount
          FROM Sailor S
                    WHERE S.age >=18
                     GROUP BY S.rating) AS
           Temp
          WHERE Temp.ratingcount > 1
```

The nested subquery returns the average age of the sailors with age >= 18 and a count of such sailors for each rating level

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

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

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



Aggregate operations cannot be nested!

This query will not work even if the expression MIN(AVG(S2.age)), which is illegal, is allowed. In the nested query, Sailors is partitioned into groups by rating, and the average age is computed for each rating value. For each group, applying MIN to this average age value for the group will return the same value.

Correct solution:

```
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)
```

1st subquery returns a table contains the average age for each rating value

2nd subquery returns a table contains the rating(s) for which this average age is the minimum.