CompSci 516
Data Intensive Computing Systems

Lecture 3
SQL - 2

Instructor: Sudeepa Roy

Announcements

- HW1 reminder:
  - Due on 09/21 (Thurs), 11:55 pm, no late days
- Your piazza, sakai, gradiance accounts should be active
  - Occasional Pop up quizzes will start from next week
  - Bring a laptop in class
  - Part of class participation

Recap: Lecture 2

- XML overview
  - differences with relational model and transformation
- SQL
  - Creating/modifying relations
  - Specifying integrity constraints
  - Key/candidate key, superkey, primary key, foreign key

Today's topic

- More SQL
  - semantic
  - joins
  - group bys and aggregates
  - nested queries

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

Basic SQL Query

```
SELECT    [DISTINCT] <target-list>
FROM      <relation-list>
WHERE     <qualification>
```

- relation-list A list of relation names
  - possibly with a "range variable" after each name
- target-list A list of attributes of relations in relation-list
- qualification Comparisons
  - (Attr op const) or (Attr1 op Attr2)
  - where op is one of = , <, >, <=, >= combined using AND, OR and NOT
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates
  - Default is that duplicates are not eliminated!

Conceptual Evaluation Strategy

```
SELECT      [DISTINCT] <target-list>
FROM        <relation-list>
WHERE       <qualification>
```

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of <*relation-list*>
  - Discard resulting tuples if they fail <*qualifications*>
  - Delete attributes that are not in <*target-list*>
  - If DISTINCT is specified, eliminate duplicate rows

- This strategy is probably the least efficient way to compute a query!
  - An optimizer will find more efficient strategies to compute the same answers

Example of Conceptual Evaluation

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

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Step 1: Form **cross product** of Sailor and Reserves

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
22	dustin	7	45	58	103	11/12/96
31	lubber	8	55	22	101	10/10/96
31	lubber	8	55	58	103	11/12/96
58	rusty	10	35	22	101	10/10/96
58	rusty	10	35	58	103	11/12/96

Reserves

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

Example of Conceptual Evaluation

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

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Step 2: Discard tuples that do not satisfy <qualification>

sid	sname	rating	age	sid	bid	day
~~22~~	~~dustin~~	~~7~~	~~45~~	~~22~~	~~101~~	~~10/10/96~~
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
58	rusty	10	35	58	103	11/12/96

Reserves

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

Example of Conceptual Evaluation

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

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

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

Step 3: Select the specified attribute(s)

sid	sname	rating	age	sid	bid	day
~~22~~	~~dustin~~	~~7~~	~~45~~	~~22~~	~~101~~	~~10/10/96~~
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
58	rusty	10	35	58	103	11/12/96

A Note on "Range Variables"

- Really needed only if the same relation appears twice in the FROM clause
  - sometimes used as a short-name
- The previous query can also be written as:

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

OR

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

It is good style, however, to use range variables always!

Find sailor ids who've reserved at least one boat

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

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

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

Find sailor ids who've reserved at least one boat

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

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

- Would adding DISTINCT to this query make a difference?

Reserves

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

Find sailors who've reserved at least one boat

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

- Would adding DISTINCT to this query make a difference?
  - Note that if there are multiple bids for the same sid, you get multiple output tuples for the same sid
  - Without distinct, you get them multiple times

- What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause?
  - Would adding DISTINCT to this variant of the query make a difference even if one sid reserves at most one bid?

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

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

Joins

- Condition/Theta-Join
- Equi-Join
- Natural-Join
- (Left/Right/Full) Outer-Join

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

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

Condition/Theta Join

```
SELECT *
FROM Sailors S, Reserves R
WHERE S.sid=R.sid and age >= 40
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Form cross product, discard rows that do not satisfy the condition

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~58~~	~~103~~	~~11/12/96~~

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

Equi Join

```
SELECT *
FROM Sailors S, Reserves R
WHERE S.sid=R.sid and age = 45
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

A special case of theta join
Join condition only has equality predicate =

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
31	lubber	8	55	22	101	10/10/96
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~58~~	~~103~~	~~11/12/96~~

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

Natural Join

```
SELECT *
FROM Sailors S NATURAL JOIN Reserves R
```

A special case of equi join
Equality condition on ALL common predicates (sid)
Duplicate columns are eliminated

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

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

sid	sname	rating	age	bid	day
22	dustin	7	45	101	10/10/96
~~22~~	~~dustin~~	~~7~~	~~45~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~101~~	~~10/10/96~~
58	rusty	10	35	103	11/12/96

Outer Join

```
SELECT  S.sid, R. bid
FROM  Sailors S LEFT OUTER JOIN Reserves R
ON  S.sid=R.sid
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Preserves all tuples from the left table whether or not there is a match
if no match, fill attributes from right with null
Similarly RIGHT/FULL outer join

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

sid	bid
22	101
31	null
58	103

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*
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters
  - You will need these often

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

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- Assume a Boats relation
- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples
  - can themselves be 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'
```

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

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

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

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- 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 B2.color='green')
```

Key field!

```
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 R.bid=B.bid
          AND B.color='green'
```

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)
```

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- A very powerful feature of SQL:
  - a WHERE/FROM/HAVING clause can itself contain an SQL query
- To find sailors who've not reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a nested loops evaluation
  - For each Sailors tuple, check the qualification by computing the 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

Nested Queries with Correlation

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

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

- If UNIQUE is used, and * is replaced by *R.bid*, finds sailors with at most one reservation for boat #103
  - UNIQUE checks for duplicate tuples

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, *op* IN
  - where op : >, <, =, <=, >=
- Find sailors whose rating is greater than that of some sailor called Horatio
  - similarly ALL

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

Aggregate Operators

Check yourself:
What do these queries compute?

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

single column

```
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
```

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 (need to replace i by num):

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

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

First go over the examples in the following slides
Then come back to this slide and study yourself

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
  - Here a group is a set of tuples that have the same value for all attributes in grouping-list

First go over the examples in the following slides
Then come back to this slide and study yourself

Conceptual Evaluation

- The cross-product of relation-list is computed
- Tuples that fail qualification are discarded
- `Unnecessary' fields are deleted
- 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
  - like "…GROUP BY bid, sid HAVING bid = 3"
- 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 (*) > 1
```

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

Answer relation:

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 such sailors.

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

Step 1: Form the cross product: FROM clause
(some attributes are omitted for simplicity)

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

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 (*) > 1
```

Step 2: Apply WHERE clause

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
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

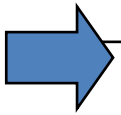
Find age of the youngest sailor with age >= 18, for each rating with at least 2 such sailors.

Step 3: Apply GROUP BY according to the listed attributes

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

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
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

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

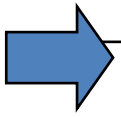
```
SELECT  S.rating,  MIN (S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

Step 4: Apply HAVING clause

The *group-qualification* is applied to eliminate some groups

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
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

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 (*) > 1

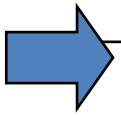
Step 5: Apply SELECT clause

Apply the aggregate operator

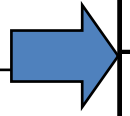
At the end, one tuple per group

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
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

Additional Examples for Practice

Check yourself

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 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

More in RA

Find sailors who've reserved all boats.

- Option 1:
- Option 2: Let's do it the hard way, without EXCEPT:

option 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))
```

option 2

```
SELECT  S.sname      Sailors S such that ...
FROM  Sailors S
WHERE  NOT EXISTS  (SELECT  B.bid      there is no boat B...
                    FROM  Boats B
                    WHERE  NOT EXISTS  (SELECT  R.bid      ...without ...
                                        FROM  Reserves R
                                        WHERE  R.bid=B.bid
                                               AND R.sid=S.sid))
                    ...a Reserves tuple showing S reserved B
```

Find name and age of the oldest sailor(s)

- The first query is illegal!
  - Recall the semantic of GROUP BY
- The third query is equivalent to the second query
  - and is allowed in the SQL/92 standard, but is not supported in some systems

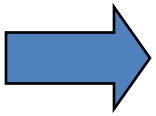
```
SELECT  S.sname, MAX (S.age)
FROM  Sailors S
```

```
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
```

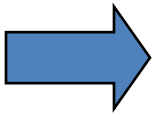
Find age of the youngest sailor with age >= 18, for each rating with at least 2 such sailors and with every sailor under 60.

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

```
SELECT  S.rating,  MIN (S.age)
AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1 AND
EVERY (S.age <=60)
```



rating	minage
7	35.0
8	25.5

What is the result of changing EVERY to ANY?

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 (*) > 1
```

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
```

- 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?
- 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 over all ratings

- 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 (Temp.avgage)
                      FROM  Temp)
```

Triggers

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- Three parts:
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run)
  - Action (what happens if the trigger runs)

```
CREATE TRIGGER youngSailorUpdate
      AFTER INSERT ON SAILORS
REFERENCING NEW TABLE NewSailors
FOR EACH STATEMENT
      INSERT
        INTO YoungSailors(sid, name, age, rating)
        SELECT sid, name, age, rating
        FROM NewSailors N
        WHERE N.age <= 18
```