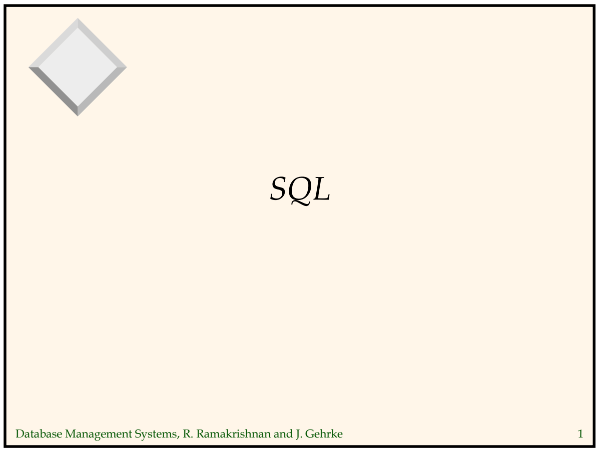
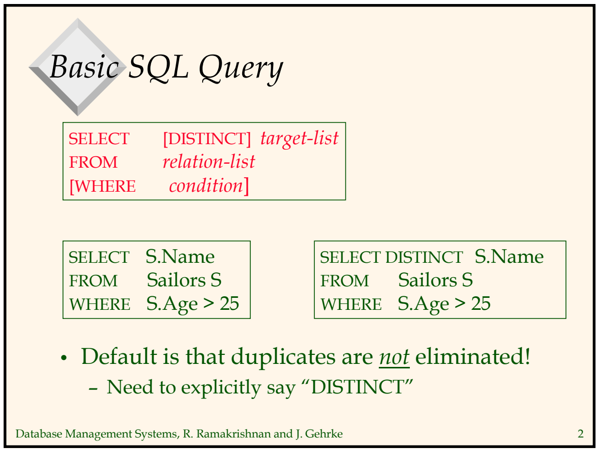


SQL



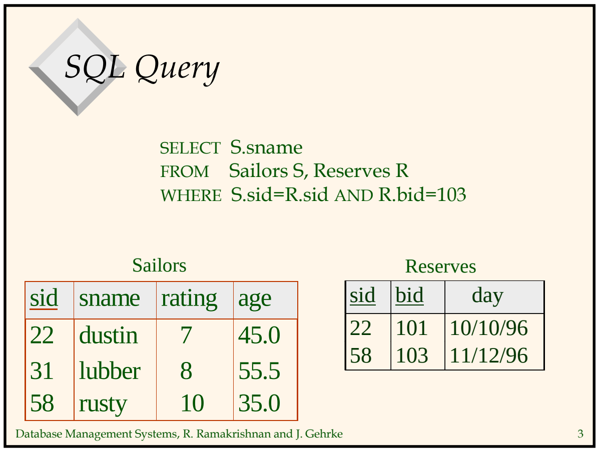
Basic SQL Query

```
SELECT      [DISTINCT]  target-list
FROM        relation-list
[WHERE      condition]
```

```
SELECT   S.Name
FROM     Sailors S
WHERE    S.Age > 25
```

```
SELECT DISTINCT  S.Name
FROM     Sailors S
WHERE    S.Age > 25
```

- Default is that duplicates are *not* eliminated!
  - Need to explicitly say "DISTINCT"



SQL Query

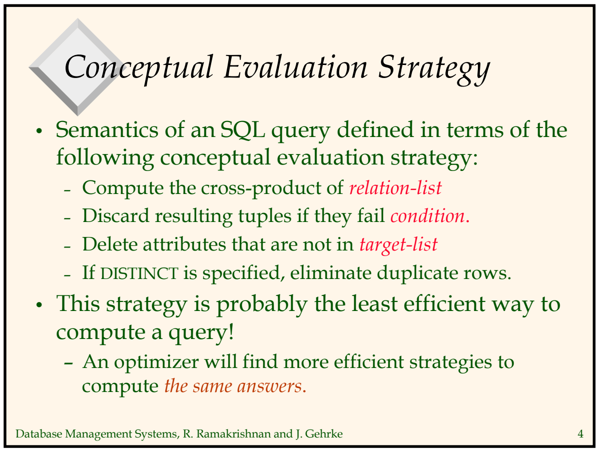
```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

Sailors

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

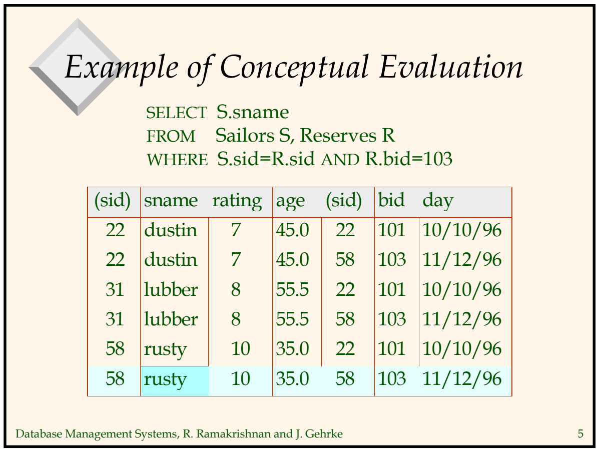
Reserves

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



Conceptual Evaluation Strategy

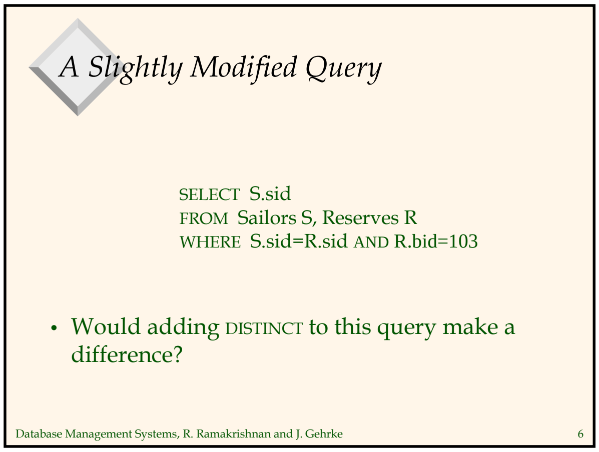
- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*
  - Discard resulting tuples if they fail *condition*.
  - Delete attributes that are not in *target-list*
  - If DISTINCT is specified, eliminate duplicate rows.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

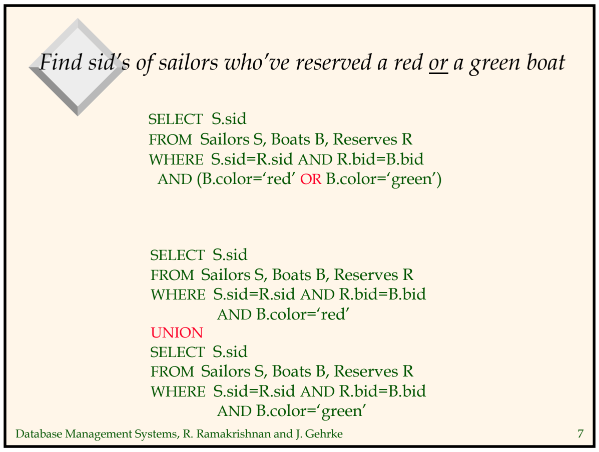
(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96



A Slightly Modified Query

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

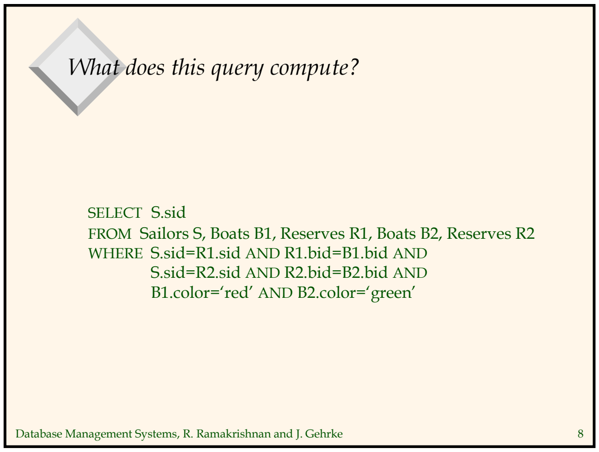
- Would adding DISTINCT to this query make a difference?



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

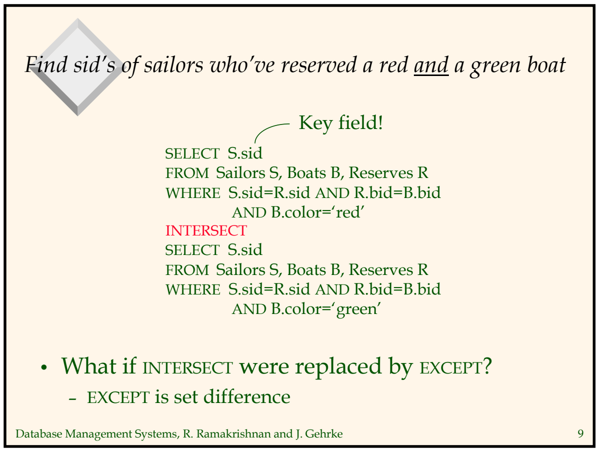
```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
  AND (B.color='red' OR B.color='green')
```

```
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'
```



What does this query compute?

```
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'
```

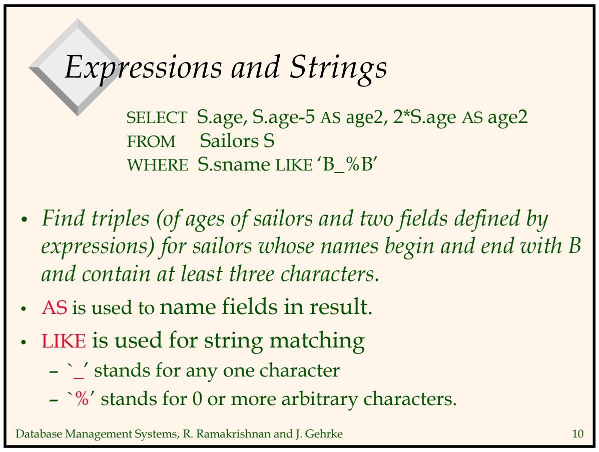


Find sid's of sailors who've reserved a red and a green boat

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'
```

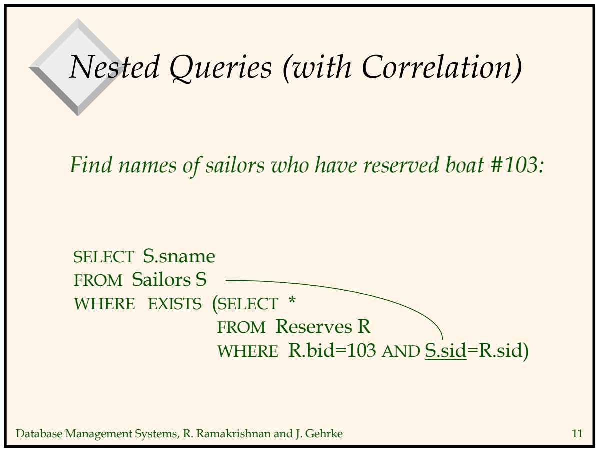
- What if INTERSECT were replaced by EXCEPT?
  - EXCEPT is set difference



Expressions and Strings

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

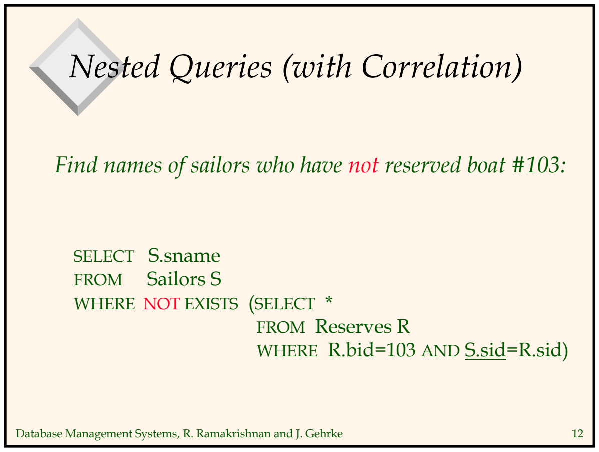
- *Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.*
- AS is used to name fields in result.
- LIKE is used for string matching
  - \`_’ stands for any one character
  - \`%’ stands for 0 or more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



Nested Queries (with Correlation)

Find names of sailors who have reserved boat #103:

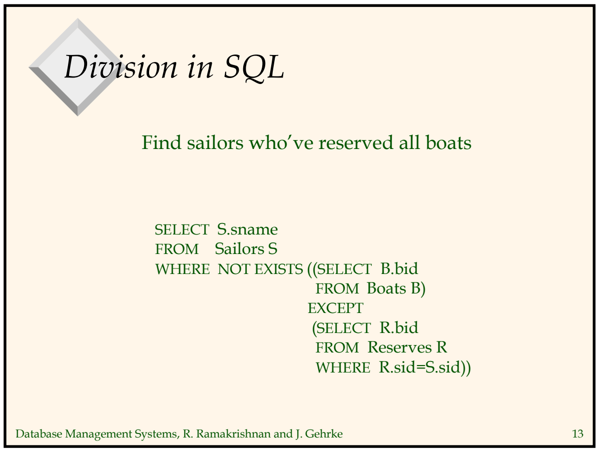
```
SELECT  S.sname
FROM  Sailors S
WHERE   EXISTS  (SELECT  *
                 FROM  Reserves R
                 WHERE  R.bid=103 AND S.sid=R.sid)
```



Nested Queries (with Correlation)

Find names of sailors who have not reserved boat #103:

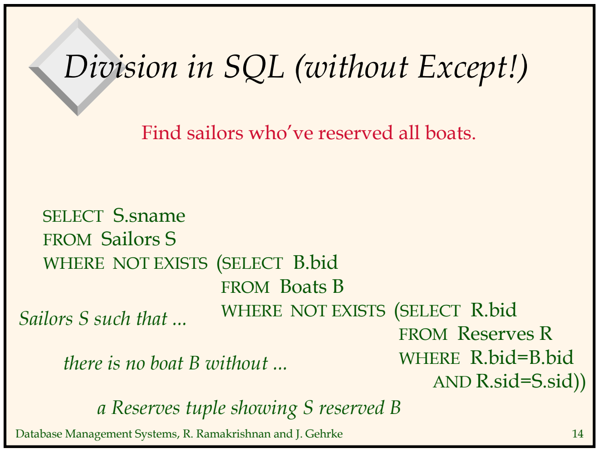
```
SELECT   S.sname
FROM     Sailors S
WHERE  NOT EXISTS  (SELECT  *
                    FROM  Reserves R
                    WHERE  R.bid=103 AND S.sid=R.sid)
```



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)
                   EXCEPT
                    (SELECT  R.bid
                     FROM  Reserves R
                     WHERE  R.sid=S.sid))
```



Division in SQL (without Except!)

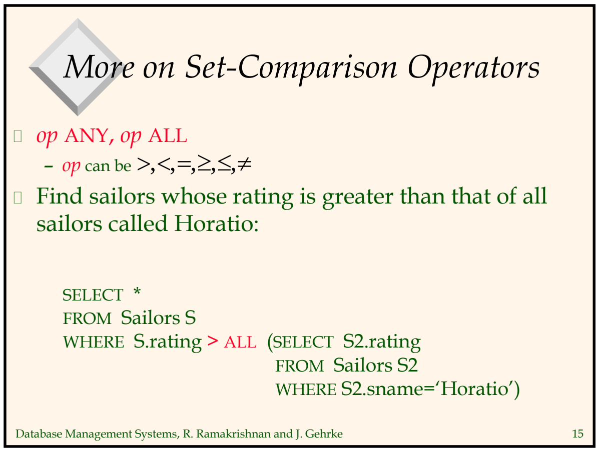
Find sailors who've reserved all boats.

```
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 ...

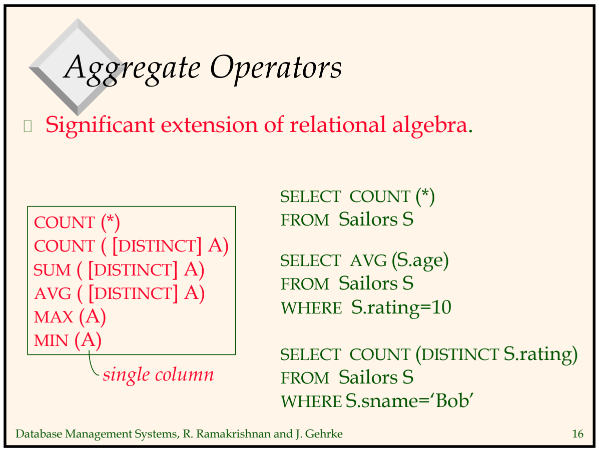
a Reserves tuple showing S reserved B



More on Set-Comparison Operators

- *op* ANY, *op* ALL
  - *op* can be >,<,=,≥,≤,≠
- Find sailors whose rating is greater than that of all sailors called Horatio:

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



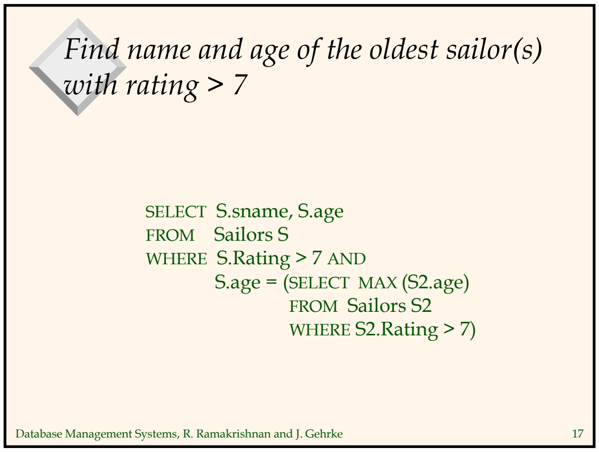
Aggregate Operators

- Significant extension of relational algebra.

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

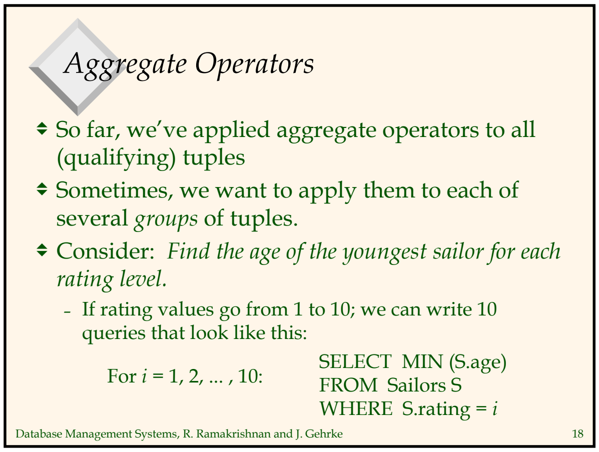
single column

```
SELECT  COUNT (*)
FROM  Sailors S

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

SELECT  COUNT (DISTINCT S.rating)
FROM  Sailors S
WHERE S.sname='Bob'
```



Find name and age of the oldest sailor(s) with rating > 7

```
SELECT  S.sname, S.age
FROM    Sailors S
WHERE  S.Rating > 7 AND
           S.age = (SELECT  MAX (S2.age)
                    FROM  Sailors S2
                    WHERE S2.Rating > 7)
```

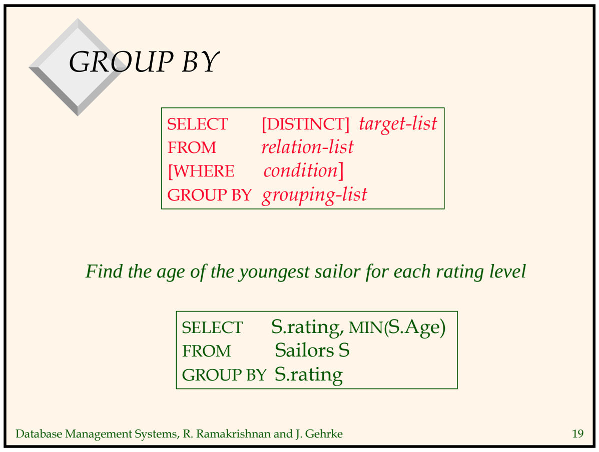


Aggregate Operators

- 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.*
  - If 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
```

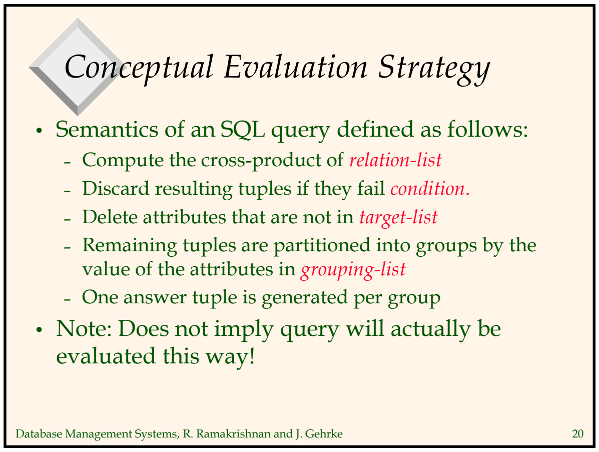


GROUP BY

```
SELECT      [DISTINCT]  target-list
FROM        relation-list
[WHERE      condition]
GROUP BY    grouping-list
```

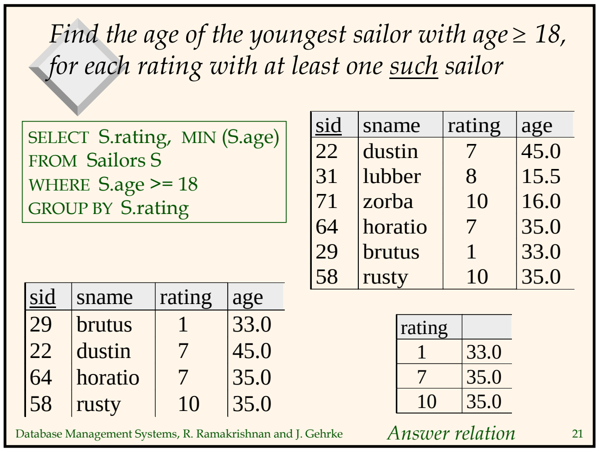
Find the age of the youngest sailor for each rating level

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



Conceptual Evaluation Strategy

- Semantics of an SQL query defined as follows:
  - Compute the cross-product of *relation-list*
  - Discard resulting tuples if they fail *condition.*
  - Delete attributes that are not in *target-list*
  - Remaining tuples are partitioned into groups by the value of the attributes in *grouping-list*
  - One answer tuple is generated per group
- Note: Does not imply query will actually be evaluated this way!



Find the age of the youngest sailor with age ≥ 18, for each rating with at least one such sailor

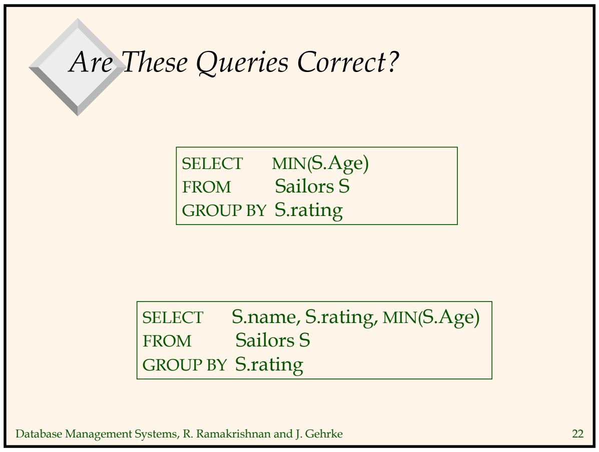
```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
```

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	15.5
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
58	rusty	10	35.0

rating	
1	33.0
7	35.0
10	35.0

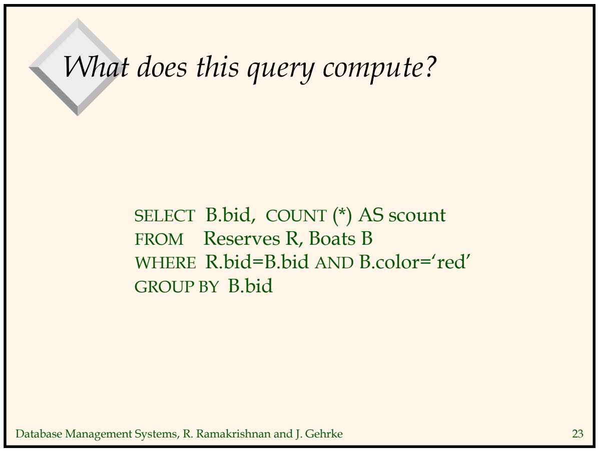
Answer relation



Are These Queries Correct?

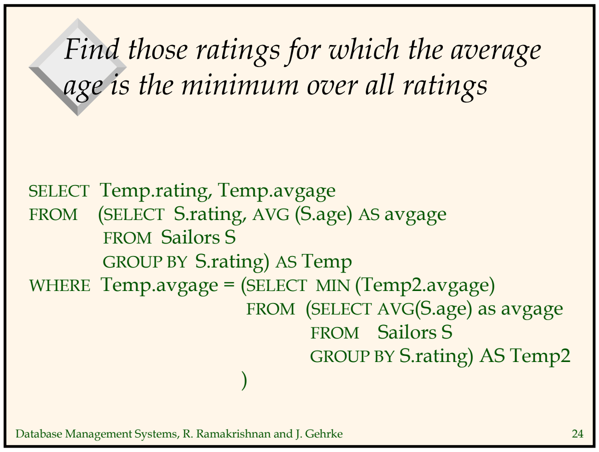
```
SELECT      MIN(S.Age)
FROM        Sailors S
GROUP BY S.rating
```

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



What does this query compute?

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



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

```
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 (Temp2.avgage)
                                FROM  (SELECT AVG(S.age) as avgage
                                            FROM    Sailors S
                                            GROUP BY S.rating) AS Temp2
                               )
```

What does this query compute?

```
SELECT  Temp.rating, Temp.minage
FROM    (SELECT  S.rating, MIN (S.age) AS minage, COUNT(*) AS cnt
         FROM  Sailors S
         WHERE S.age >= 18
         GROUP BY  S.rating) AS Temp
WHERE  Temp.cnt >= 2
```

Queries With GROUP BY and HAVING

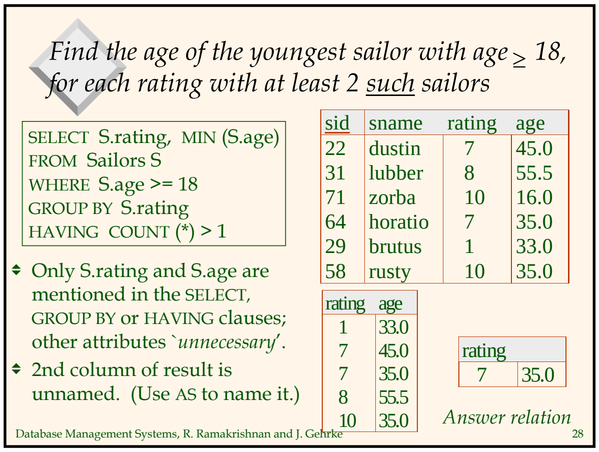
```
SELECT      [DISTINCT]  target-list
FROM        relation-list
[WHERE       qualification]
GROUP BY  grouping-list
HAVING      group-qualification
```

Find the age of the youngest sailor with age >= 18 for each rating level 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(*) >= 2
```

Conceptual Evaluation Strategy

- Semantics of an SQL query defined as follows:
  - Compute the cross-product of *relation-list*
  - Discard resulting tuples if they fail *condition.*
  - Delete attributes that are not in *target-list*
  - Remaining tuples are partitioned into groups by the value of the attributes in *grouping-list*
  - The *group-qualification* is applied to eliminate some groups
  - One answer tuple is generated per qualifying group
- Note: Does not imply query will actually be evaluated this way!



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
```

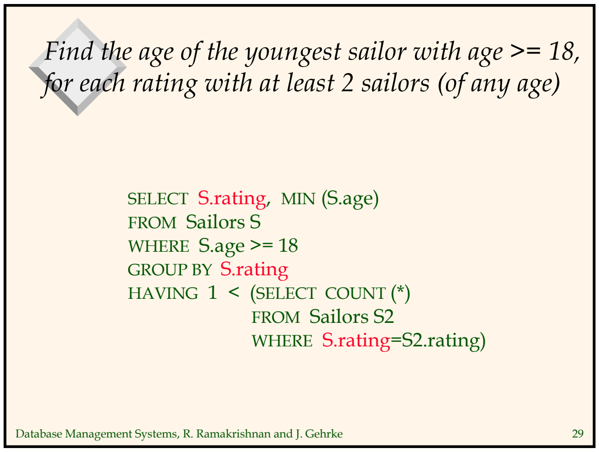
sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

- Only S.rating and S.age are mentioned in the SELECT, GROUP BY or HAVING clauses; other attributes `*unnecessary*'.
- 2nd column of result is unnamed. (Use AS to name it.)

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

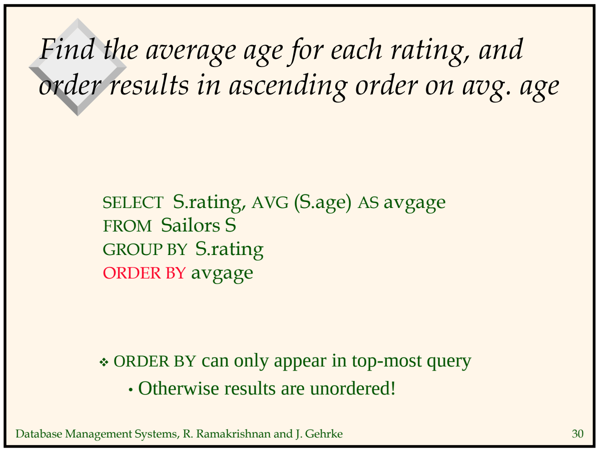
rating	
7	35.0

Answer relation



Find the age of the youngest sailor with age >= 18, for each rating with at least 2 sailors (of any age)

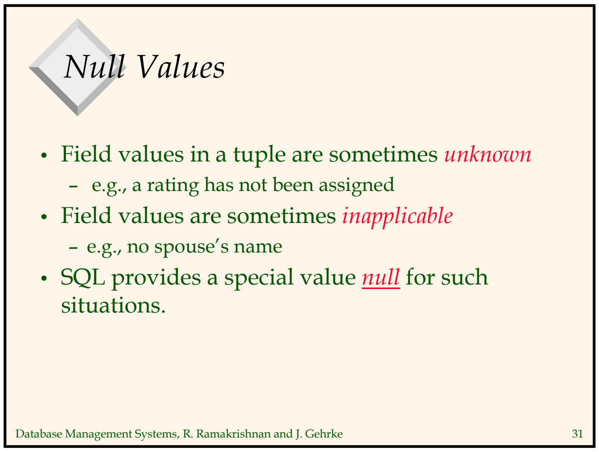
```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  1  <  (SELECT  COUNT (*)
               FROM  Sailors S2
               WHERE  S.rating=S2.rating)
```



Find the average age for each rating, and order results in ascending order on avg. age

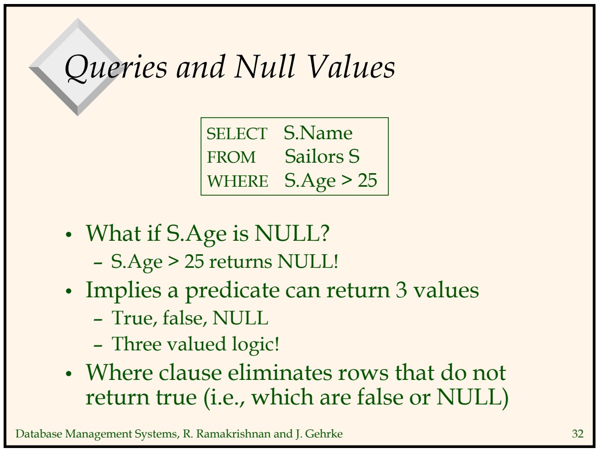
```
SELECT  S.rating, AVG (S.age) AS avgage
FROM  Sailors S
GROUP BY  S.rating
ORDER BY  avgage
```

- ORDER BY can only appear in top-most query
  - Otherwise results are unordered!



Null Values

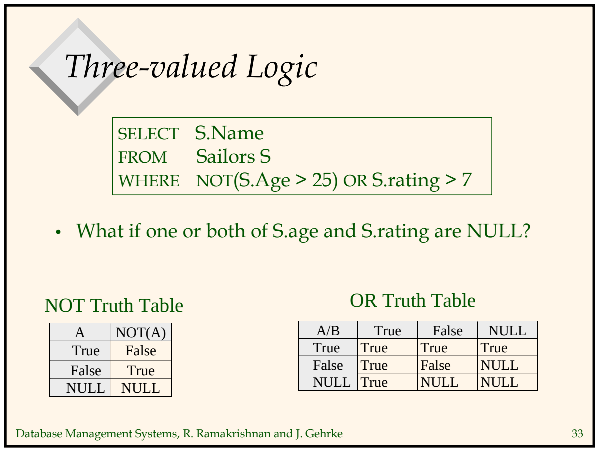
- Field values in a tuple are sometimes *unknown*
  - e.g., a rating has not been assigned
- Field values are sometimes *inapplicable*
  - e.g., no spouse's name
- SQL provides a special value *null* for such situations.



Queries and Null Values

```
SELECT   S.Name
FROM     Sailors S
WHERE    S.Age > 25
```

- What if S.Age is NULL?
  - S.Age > 25 returns NULL!
- Implies a predicate can return 3 values
  - True, false, NULL
  - Three valued logic!
- Where clause eliminates rows that do not return true (i.e., which are false or NULL)



Three-valued Logic

```
SELECT   S.Name
FROM     Sailors S
WHERE    NOT(S.Age > 25) OR S.rating > 7
```

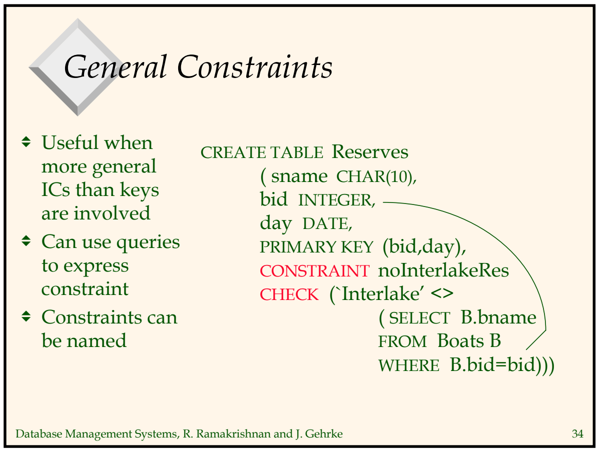
- What if one or both of S.age and S.rating are NULL?

NOT Truth Table

A	NOT(A)
True	False
False	True
NULL	NULL

OR Truth Table

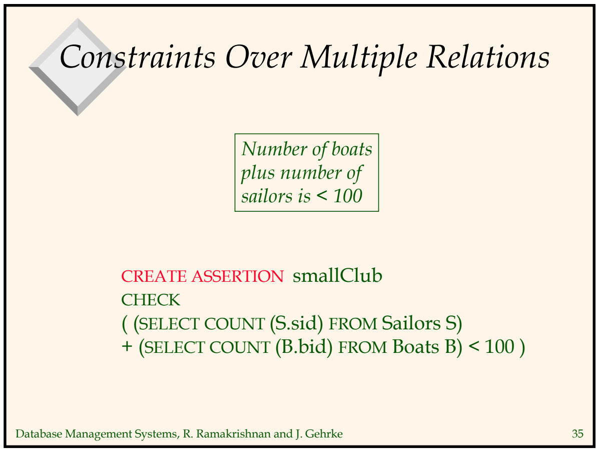
A/B	True	False	NULL
True	True	True	True
False	True	False	NULL
NULL	True	NULL	NULL



General Constraints

- Useful when more general ICs than keys are involved
- Can use queries to express constraint
- Constraints can be named

```
CREATE TABLE Reserves
        ( sname  CHAR(10),
        bid  INTEGER,
        day  DATE,
        PRIMARY KEY  (bid,day),
        CONSTRAINT  noInterlakeRes
        CHECK  (`Interlake' <>
                        ( SELECT  B.bname
                        FROM  Boats B
                        WHERE  B.bid=bid)))
```



Constraints Over Multiple Relations

Number of boats plus number of sailors is < 100

```
CREATE ASSERTION  smallClub
CHECK
( (SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid) FROM Boats B) < 100 )
```