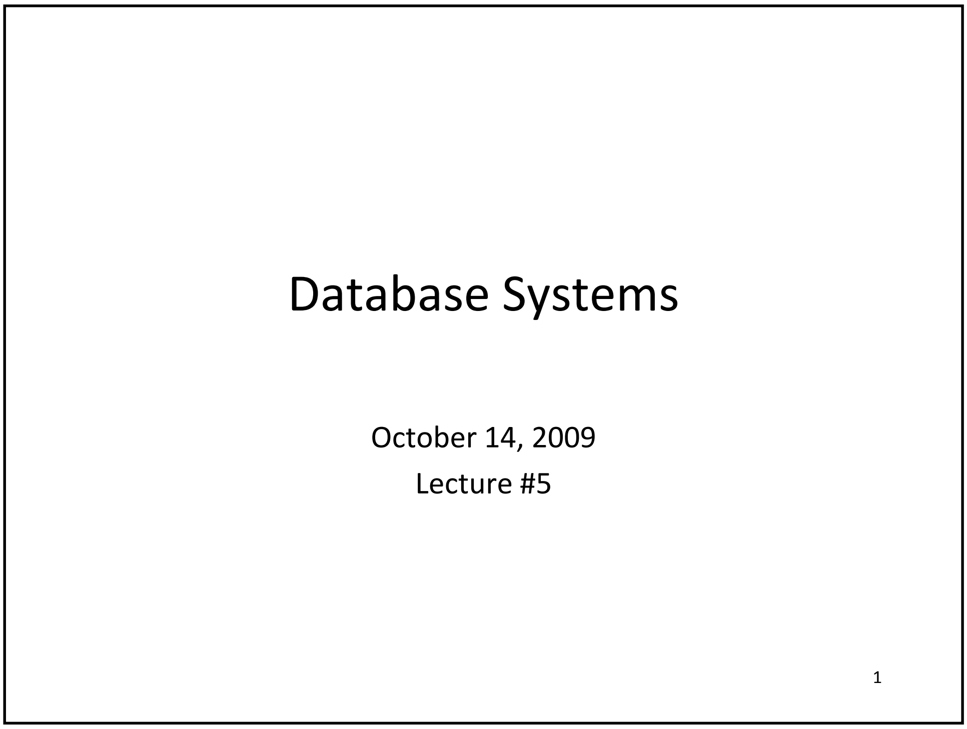
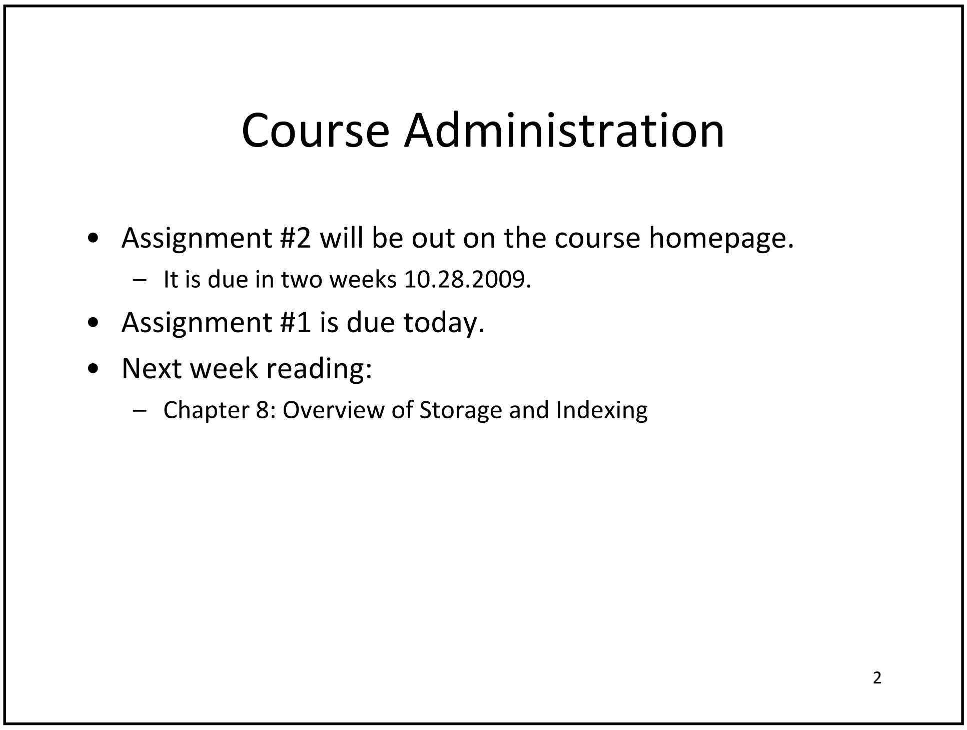


Database Systems

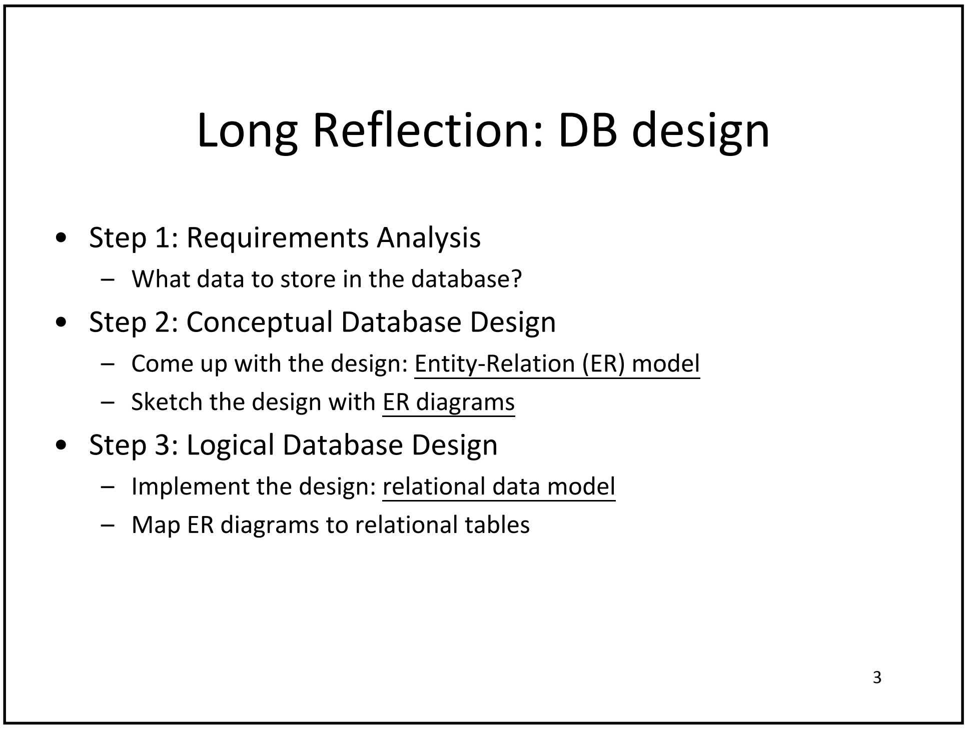
October 14, 2009

Lecture #5



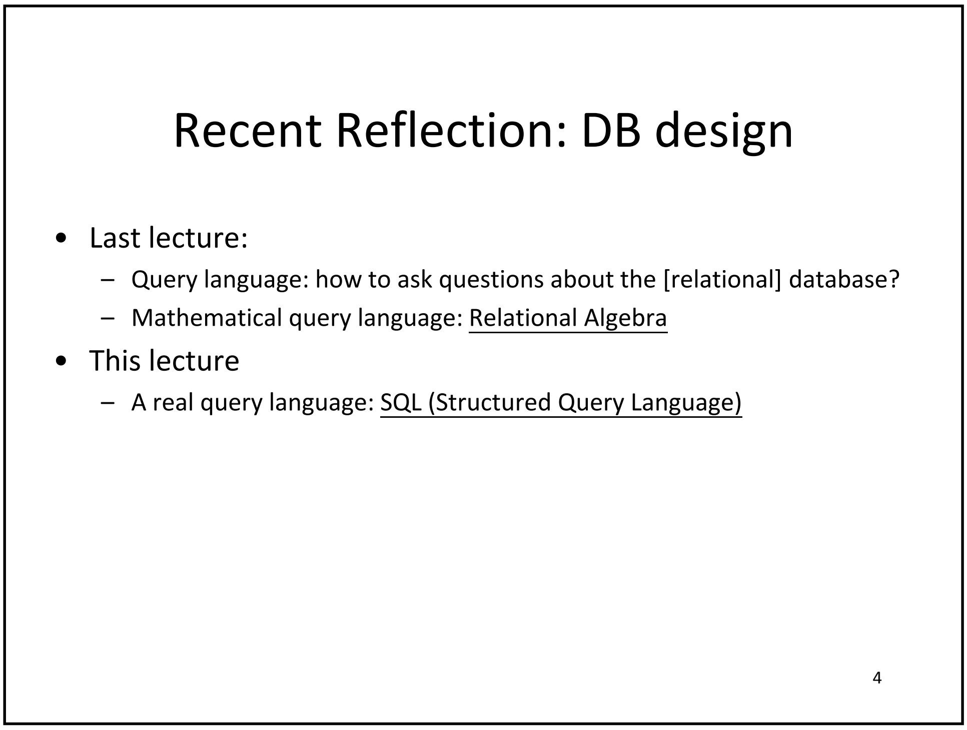
Course Administration

- Assignment #2 will be out on the course homepage.
  - It is due in two weeks 10.28.2009.
- Assignment #1 is due today.
- Next week reading:
  - Chapter 8: Overview of Storage and Indexing



Long Reflection: DB design

- Step 1: Requirements Analysis
  - What data to store in the database?
- Step 2: Conceptual Database Design
  - Come up with the design: Entity-Relation (ER) model
  - Sketch the design with ER diagrams
- Step 3: Logical Database Design
  - Implement the design: relational data model
  - Map ER diagrams to relational tables



Recent Reflection: DB design

- Last lecture:
  - Query language: how to ask questions about the [relational] database?
  - Mathematical query language: Relational Algebra
- This lecture
  - A real query language: SQL (Structured Query Language)

Review: Relational Algebra

- A query is applied to table(s), and the result of a query is also a table.
- Find the names of sailors who have reserved boat 103

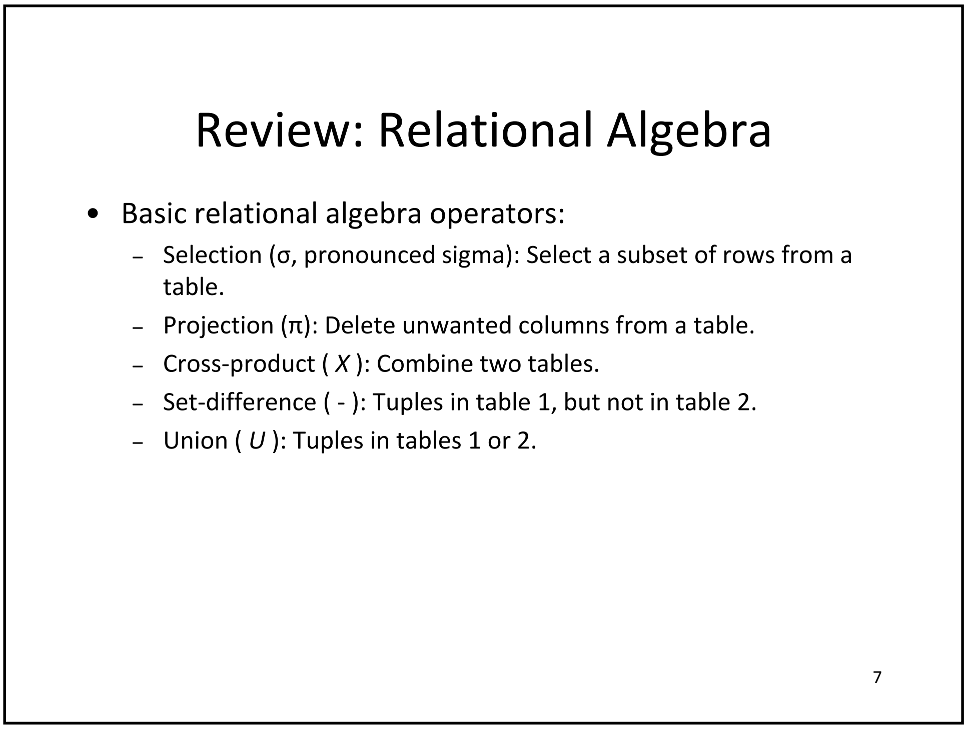
$\pi_{sname}((\sigma_{bid = 103}\ Reserves) \infty\ Sailors)$

5

Example Table Definitions

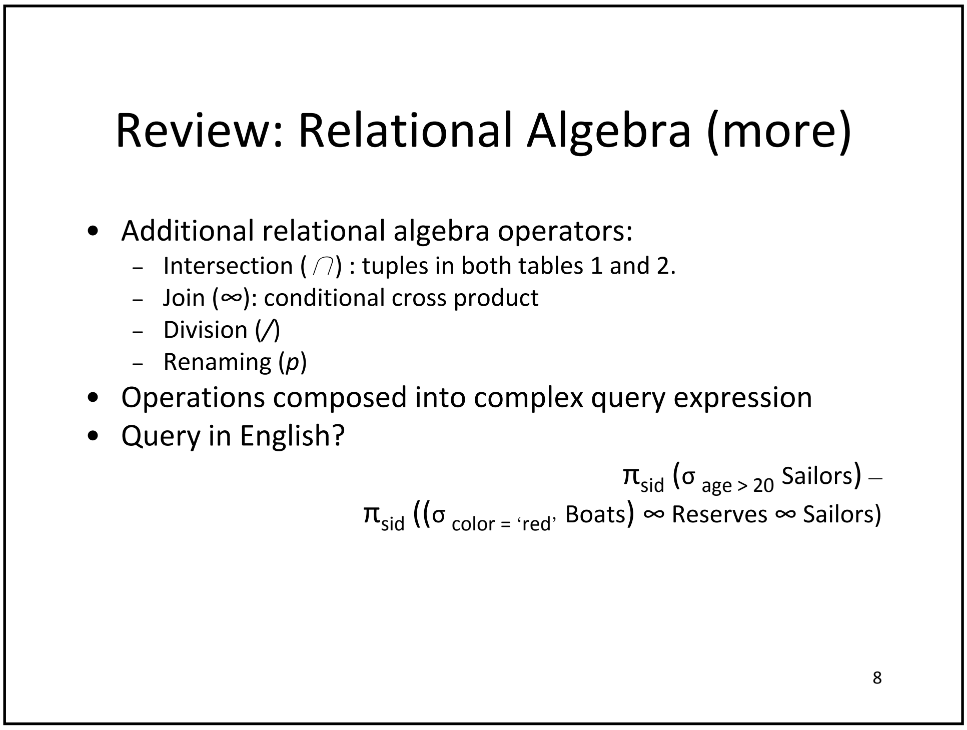
Sailors(sid: integer, sname: string, rating: integer, age: real)
Boats(bid: integer, bname: string, color: string)
Reserves(sid: integer, bid: integer, day: date)

6



Review: Relational Algebra

- Basic relational algebra operators:
  - Selection (σ, pronounced sigma): Select a subset of rows from a table.
  - Projection (π): Delete unwanted columns from a table.
  - Cross-product (*X*): Combine two tables.
  - Set-difference (-): Tuples in table 1, but not in table 2.
  - Union (*U*): Tuples in tables 1 or 2.



Review: Relational Algebra (more)

- Additional relational algebra operators:
  - Intersection (*∩*) : tuples in both tables 1 and 2.
  - Join (∞): conditional cross product
  - Division (*/*)
  - Renaming (*p*)
- Operations composed into complex query expression
- Query in English?

$$\pi_{sid} (\sigma_{age > 20} \text{ Sailors}) - \pi_{sid} ((\sigma_{color = 'red'} \text{ Boats}) \infty \text{ Reserves} \infty \text{ Sailors})$$

Relational Algebra to SQL

- Relational operators → SQL commands

Relational Algebra:

$\pi_{sname}\,(\sigma_{bid\,=\,103}\,(Sailors \infty\ Reserves))$

SQL:

SELECT S.sname

FROM ***Sailors S, Reserves R***

WHERE S.sid=R.sid AND R.bid=103

- Guess the mapping?
  - Notice the difference between SELECT (SQL) and σ

W

SQL: Queries, Constraints, Triggers

Chapter 5

Lecture Outline

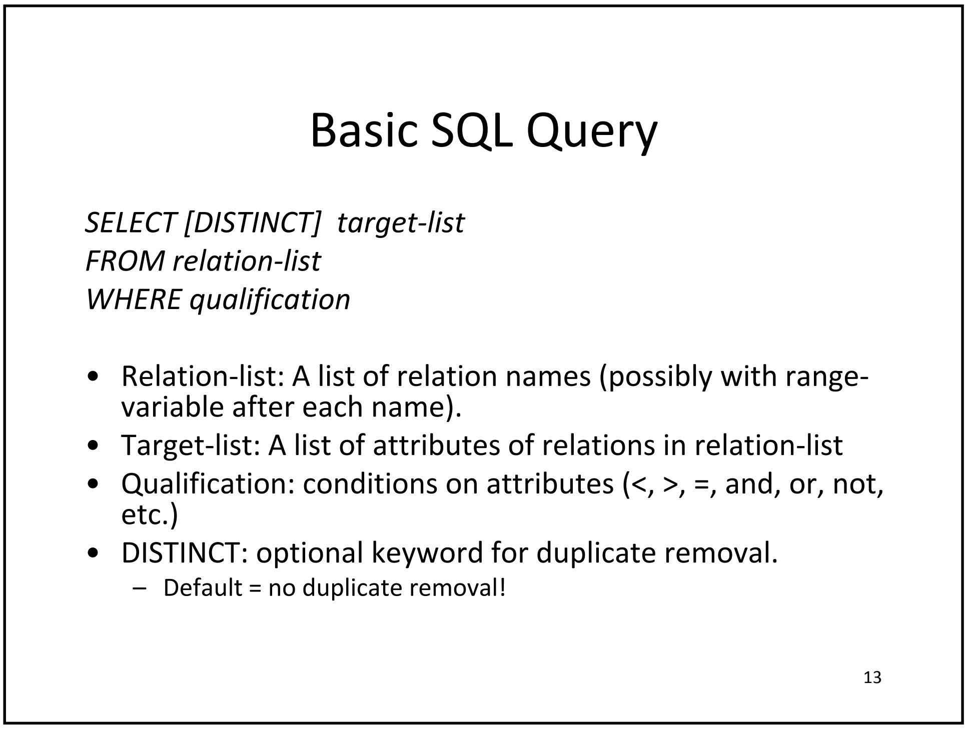
- Basic Query
  - *SELECT*
- Set Constructs
  - *UNION, INTERSECT, EXCEPT, IN, ANY, ALL, EXISTS*
- Nested Queries
- Aggregate Operators
  - *COUNT, SUM, AVG, MAX, MIN, GROUP BY, HAVING*
- Null Values
- Integrity Constraints
  - *CHECK, CREATE ASSERTION*
- Triggers
  - *CREATE TRIGGER, FOR EACH ROW*

Example Table Definitions

Sailors(sid: integer, sname: string, rating: integer, age: real)
Boats(bid: integer, bname: string, color: string)
Reserves(sid: integer, bid: integer, day: date)

- Find names of sailors who’ve reserved boat #103

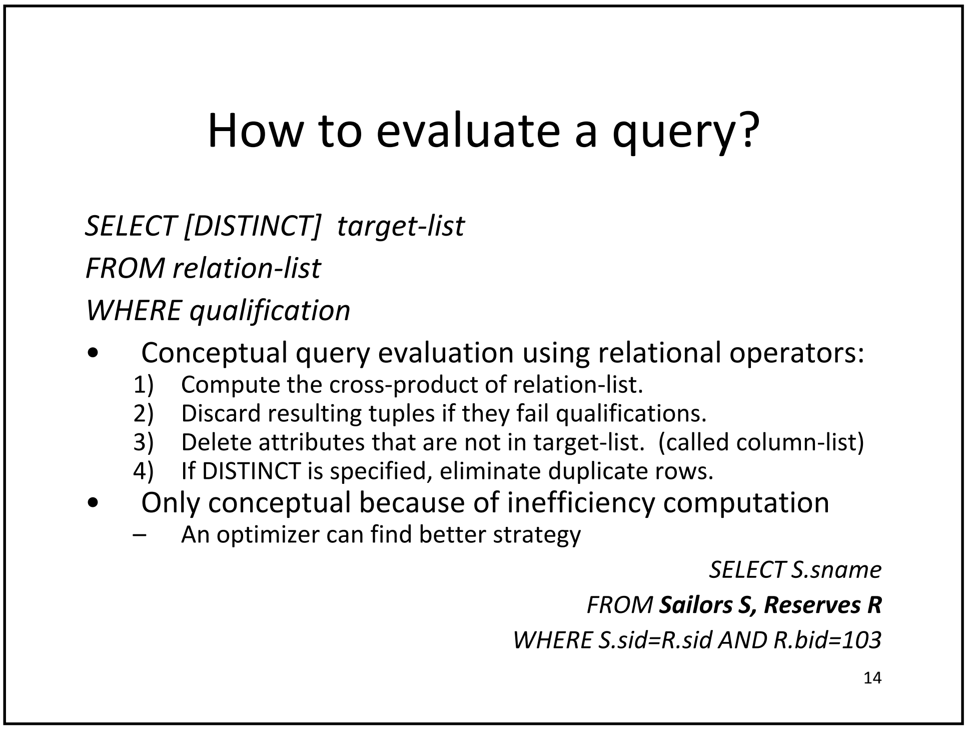
```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103
```



Basic SQL Query

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

- Relation-list: A list of relation names (possibly with range-variable after each name).
- Target-list: A list of attributes of relations in relation-list
- Qualification: conditions on attributes (<, >, =, and, or, not, etc.)
- DISTINCT: optional keyword for duplicate removal.
  - Default = no duplicate removal!



How to evaluate a query?

SELECT [DISTINCT] target-list

FROM relation-list

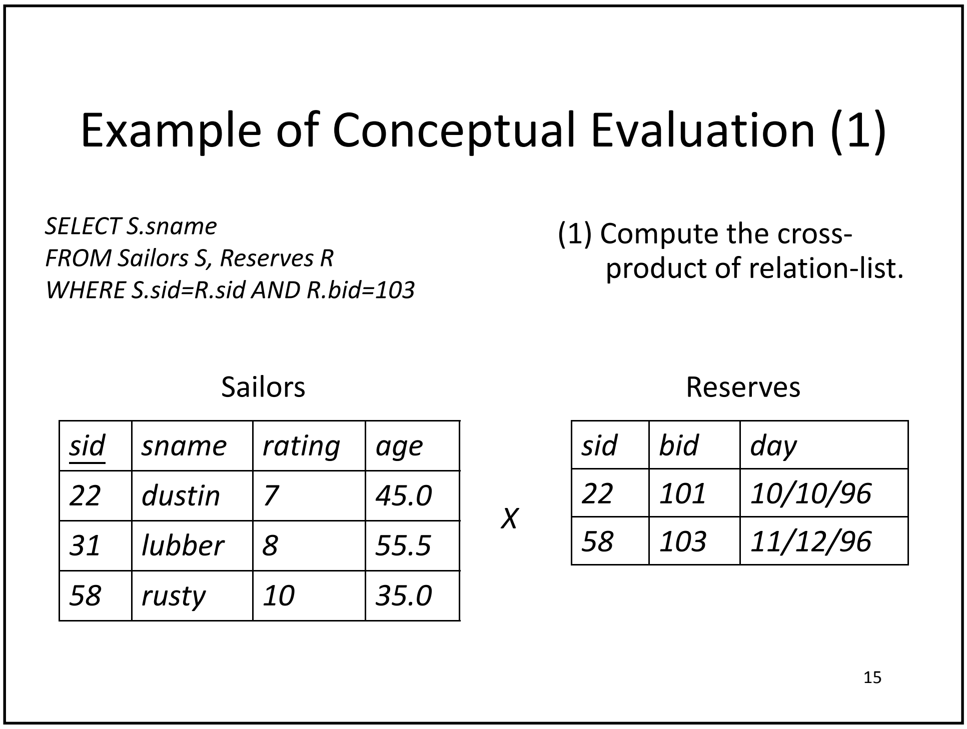
WHERE qualification

- Conceptual query evaluation using relational operators:
  1) Compute the cross-product of relation-list.
  2) Discard resulting tuples if they fail qualifications.
  3) Delete attributes that are not in target-list. (called column-list)
  4) If DISTINCT is specified, eliminate duplicate rows.
- Only conceptual because of inefficiency computation
  - An optimizer can find better strategy

SELECT S.sname

FROM ***Sailors S, Reserves R***

WHERE S.sid=R.sid AND R.bid=103



Example of Conceptual Evaluation (1)

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

(1) Compute the cross-product of relation-list.

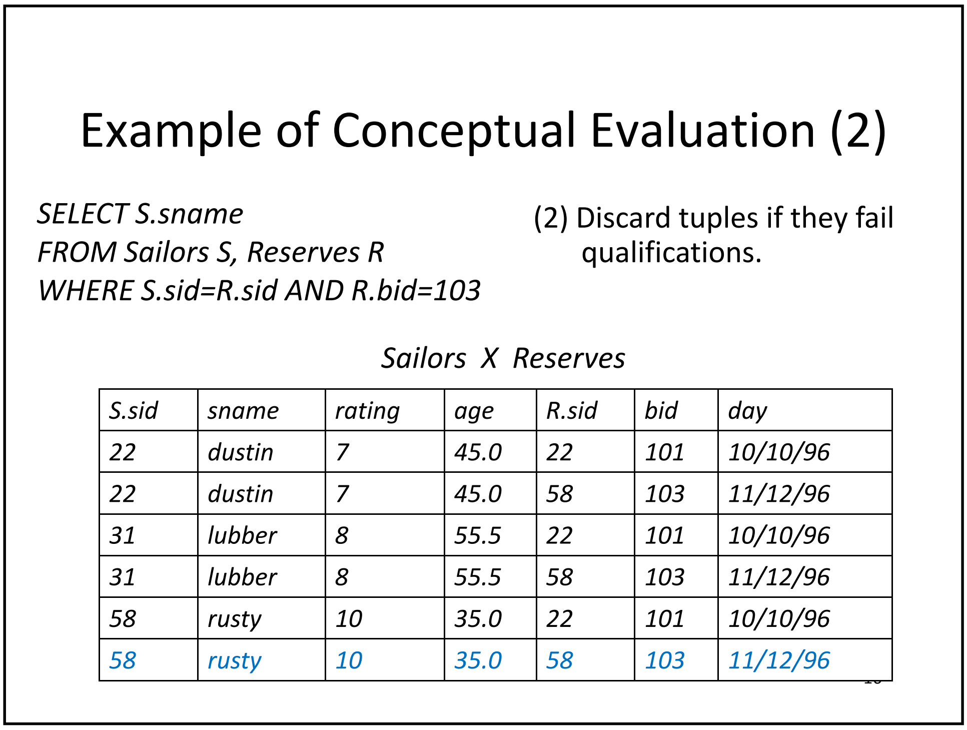
Sailors

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

X

Reserves

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



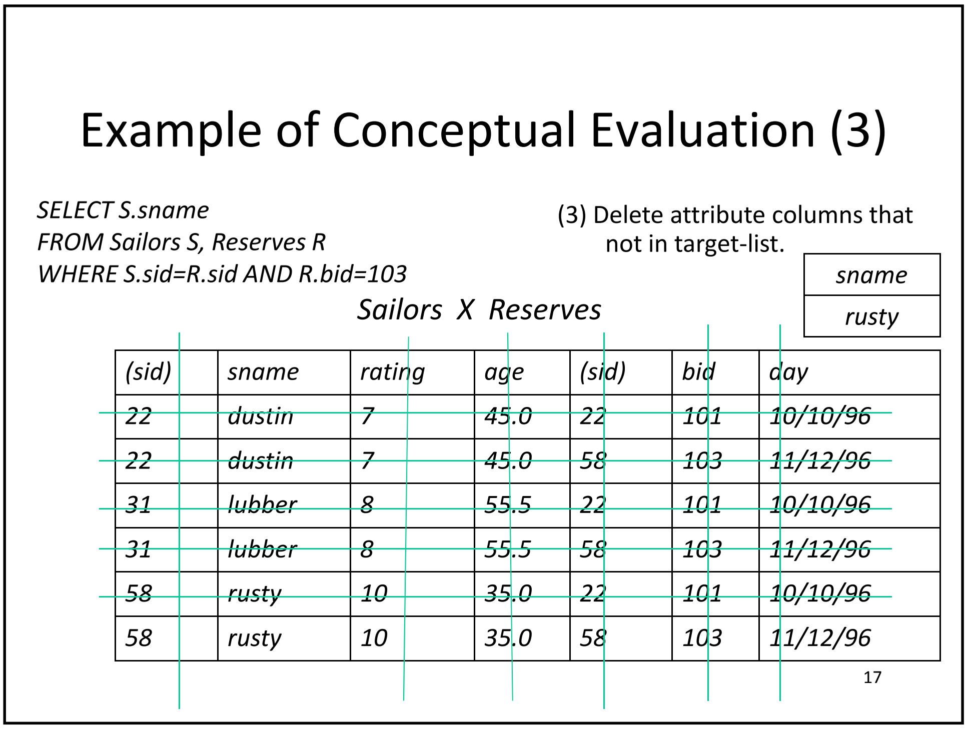
Example of Conceptual Evaluation (2)

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

(2) Discard tuples if they fail qualifications.

Sailors X Reserves

S.sid	sname	rating	age	R.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



Example of Conceptual Evaluation (3)

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

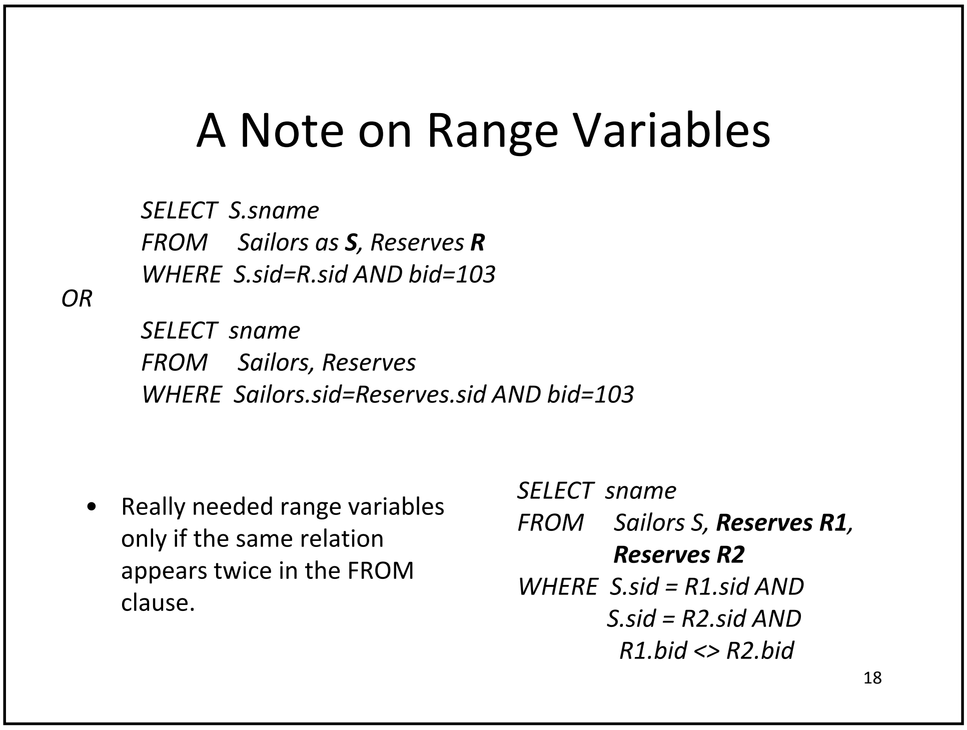
(3) Delete attribute columns that not in target-list.

sname
rusty

Sailors X Reserves

~~*(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*~~

17



A Note on Range Variables

```
SELECT  S.sname
FROM    Sailors as 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
```

- Really needed range variables only if the same relation appears twice in the FROM clause.

```
SELECT  sname
FROM    Sailors S, Reserves R1,
        Reserves R2
WHERE   S.sid = R1.sid AND
        S.sid = R2.sid AND
        R1.bid <> R2.bid
```

18

Find the sids of sailors who've reserved at least one boat

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

Sailors X Reserves

(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*

DISTINCT

- Find the names and ages of all sailors
  SELECT S.sname, S.age
  FROM Sailors S
- Add *DISTINCT* to this query?
- Replace S.sname by S.sid in the *SELECT* clause?
- Add *DISTINCT* to the above?

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

Find sailors whose names begin and end with B and contain at least three characters.

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

- *AS* and = are two ways to name fields in result.
- *LIKE* for string matching.
  - `_' for one character
  - `%' for 0 or more characters.

Sid	Sname	Rating	Age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	20

Age	Age1	Age2
20	15	40

21

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

```
SELECT
FROM
WHERE
```

- *UNION*: work on two *union-compatible* sets of tuples

```
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'
```

22

Find sid's of sailors who've reserved 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'
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'
```

- *(A Except B)* returns tuples in A but not in B.
- What is the query in English if we replace *INTERSECT* by *EXCEPT*?
  - Find sids of all sailors who have reserved a red boat but not a green boat.

SET Construct: *UNION ALL*

- *UNION, INTERSECT*, and *EXCEPT* delete duplicate by default.
- To retain duplicates, use *UNION ALL, INTERSECT ALL, or EXCEPT ALL*.

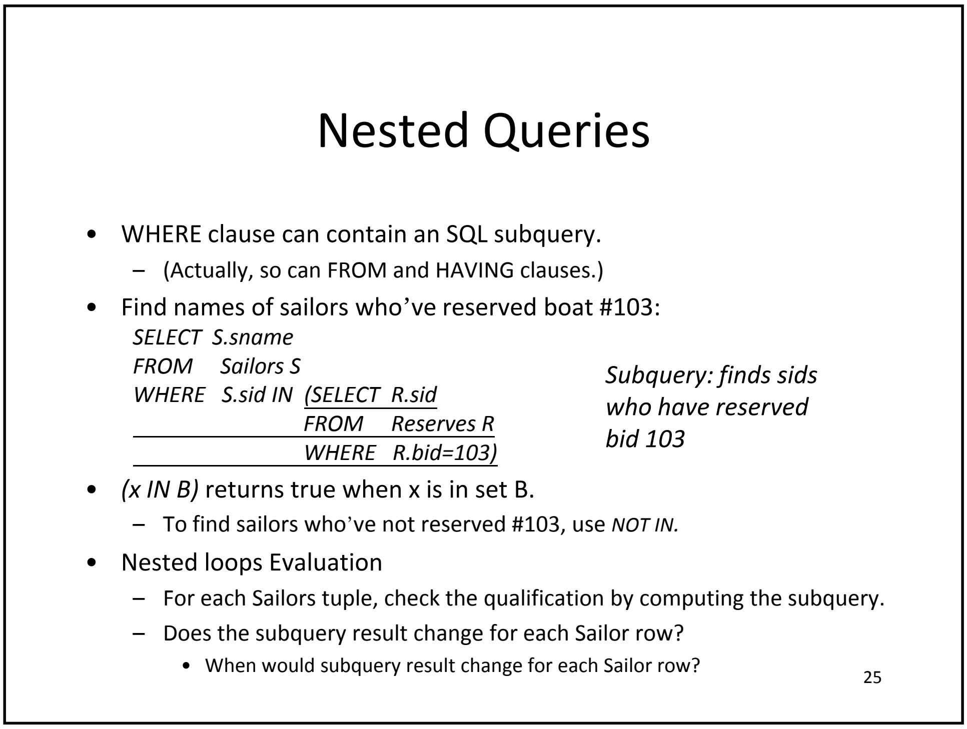
Sid	*Sname*
71	*Zorba*
74	*Horatio*
74	*Horatio*
95	*Bob*

INTERSECT ALL

Sid	*Sname*
22	*Dustin*
71	*Zorba*
74	*Horatio*
74	*Horatio*

=

Sid	*Sname*
71	*Zorba*
74	*Horatio*
74	*Horatio*



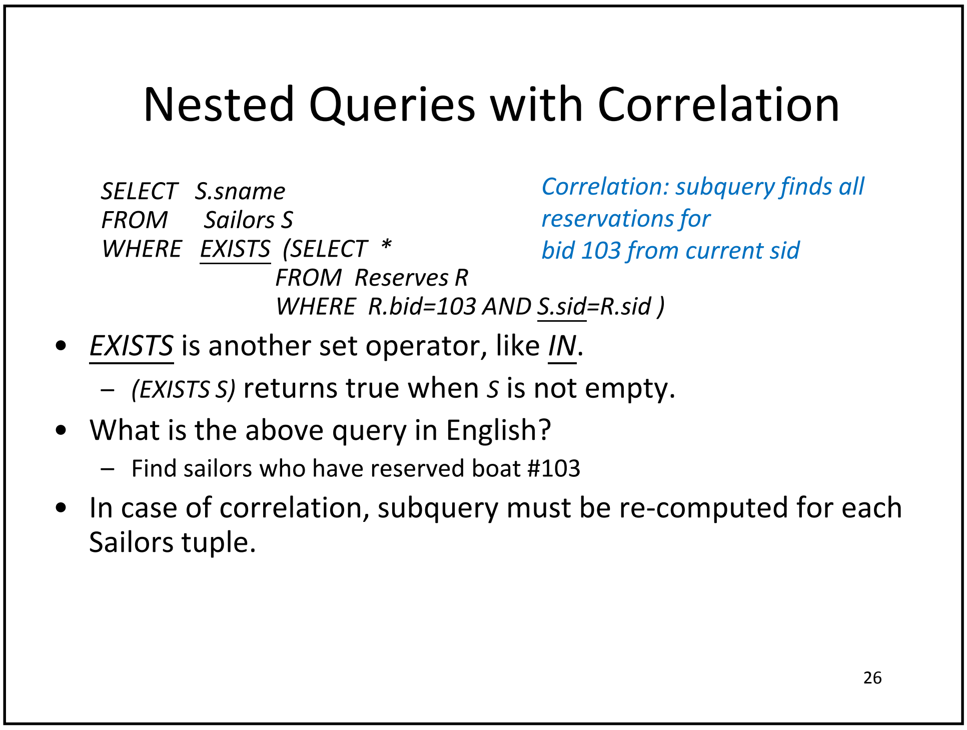
Nested Queries

- WHERE clause can contain an SQL subquery.
  - (Actually, so can FROM and HAVING clauses.)
- 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)
```

Subquery: finds sids who have reserved bid 103

- *(x IN B)* returns true when x is in set B.
  - To find sailors who've not reserved #103, use *NOT IN.*
- Nested loops Evaluation
  - For each Sailors tuple, check the qualification by computing the subquery.
  - Does the subquery result change for each Sailor row?
    - When would subquery result change for each Sailor row?

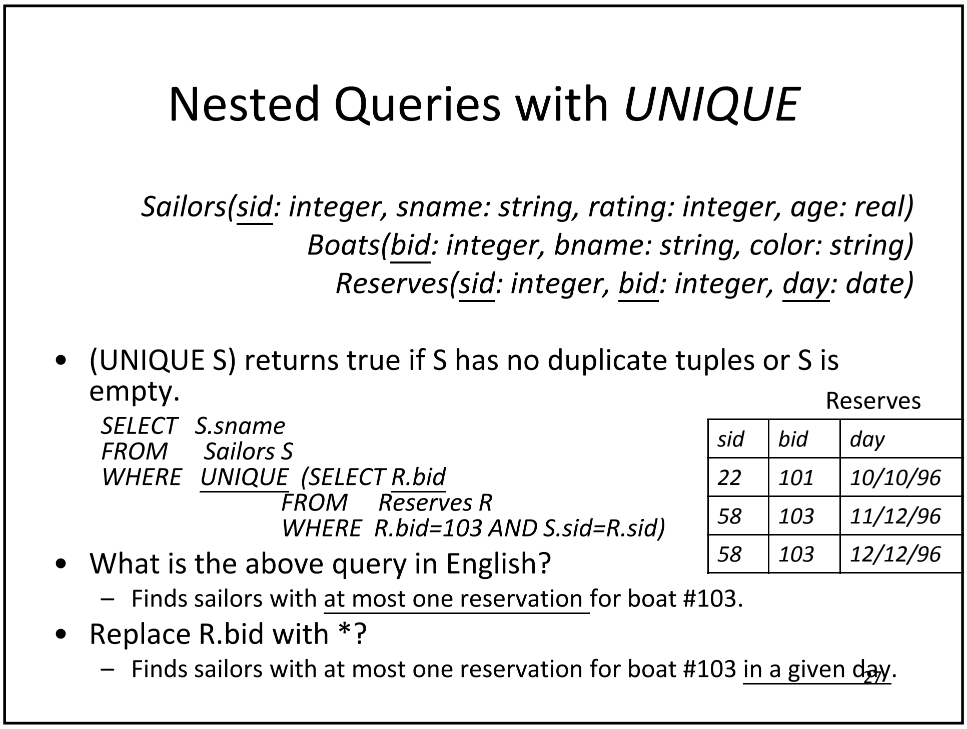


Nested Queries with Correlation

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

Correlation: subquery finds all reservations for bid 103 from current sid

- *EXISTS* is another set operator, like *IN*.
  - *(EXISTS S)* returns true when *S* is not empty.
- What is the above query in English?
  - Find sailors who have reserved boat #103
- In case of correlation, subquery must be re-computed for each Sailors tuple.



Nested Queries with *UNIQUE*

Sailors(sid: integer, sname: string, rating: integer, age: real)
Boats(bid: integer, bname: string, color: string)
Reserves(sid: integer, bid: integer, day: date)

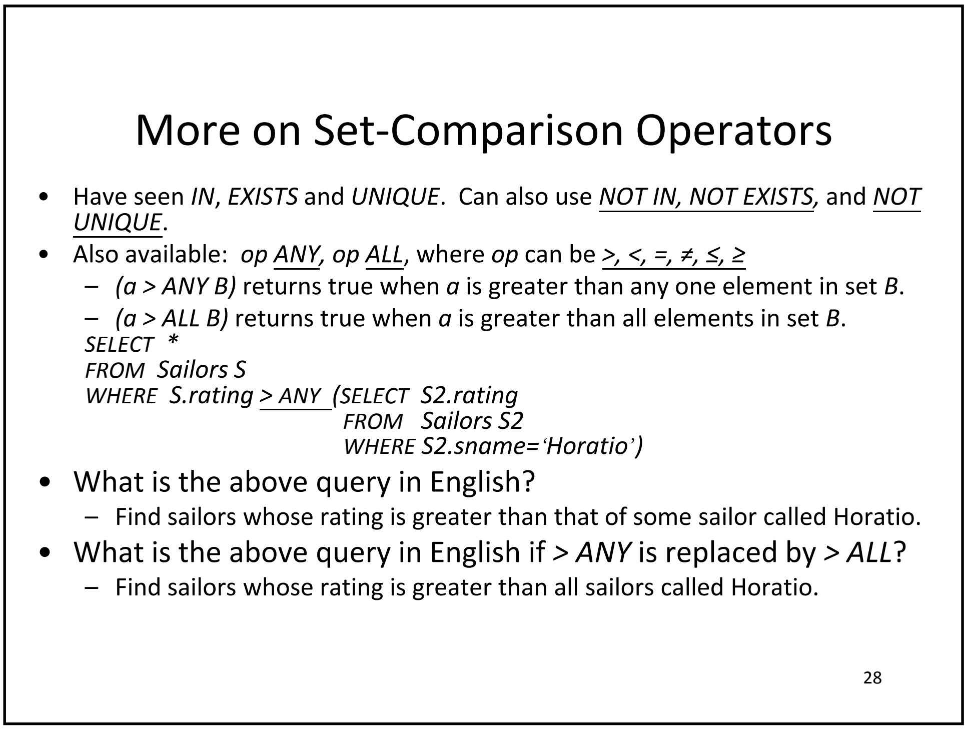
- (UNIQUE S) returns true if S has no duplicate tuples or S is empty.

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

Reserves

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

- What is the above query in English?
  - Finds sailors with at most one reservation for boat #103.
- Replace R.bid with *?
  - Finds sailors with at most one reservation for boat #103 in a given day.

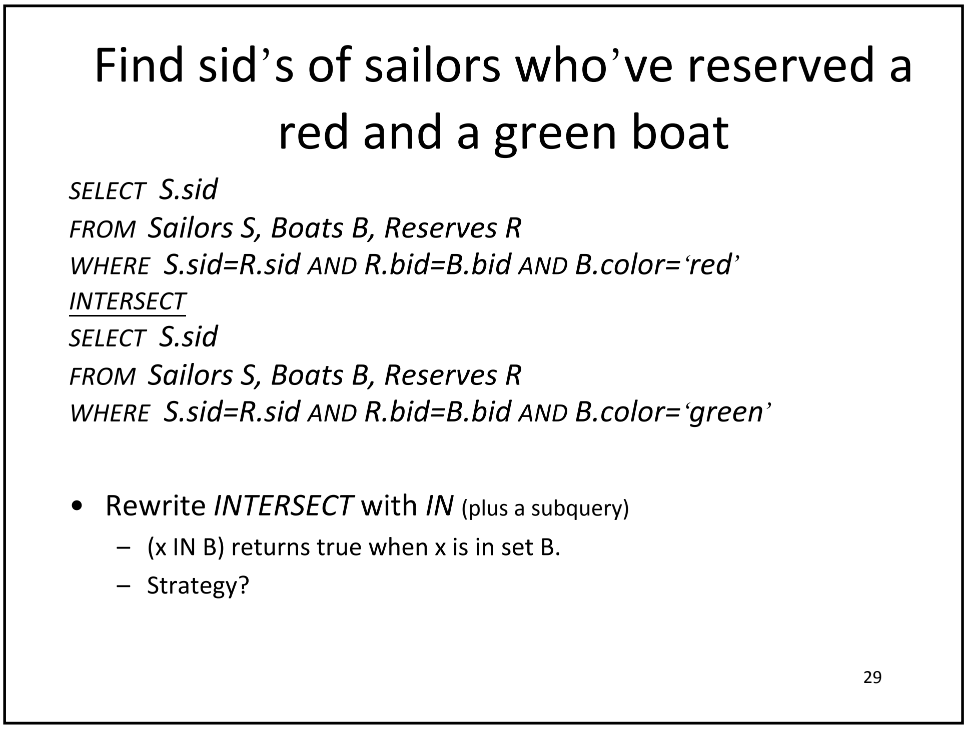


More on Set-Comparison Operators

- Have seen *IN*, *EXISTS* and *UNIQUE*. Can also use *NOT IN, NOT EXISTS,* and *NOT UNIQUE*.
- Also available: *op ANY, op ALL*, where *op* can be >, <, =, ≠, ≤, ≥
  - *(a > ANY B)* returns true when *a* is greater than any one element in set *B*.
  - *(a > ALL B)* returns true when *a* is greater than all elements in set *B*.

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

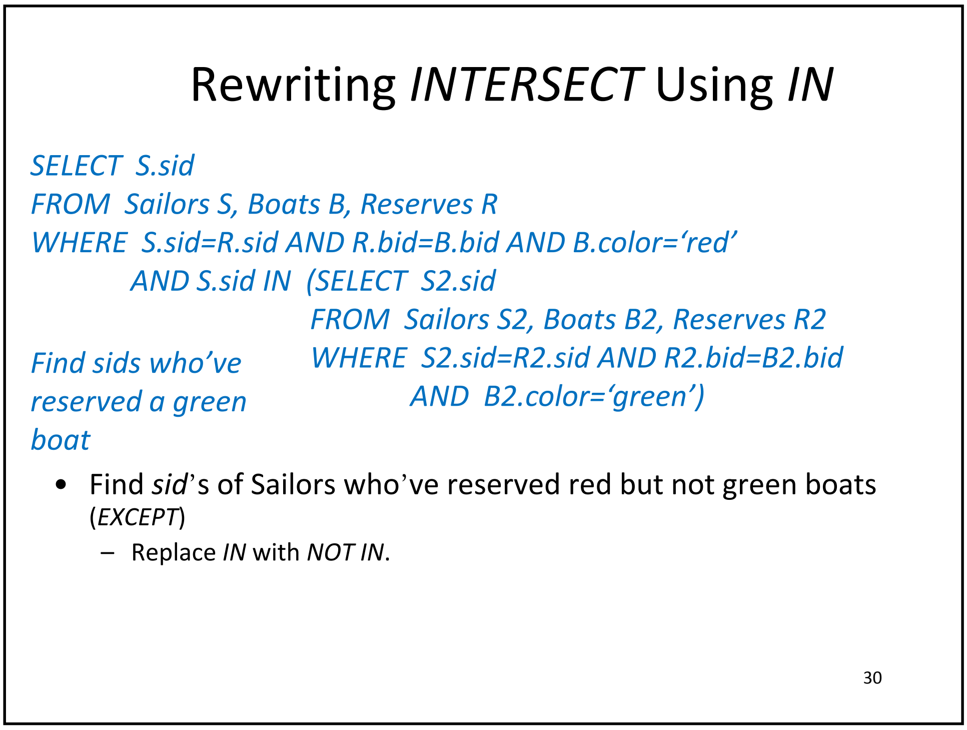
- What is the above query in English?
  - Find sailors whose rating is greater than that of some sailor called Horatio.
- What is the above query in English if *> ANY* is replaced by *> ALL*?
  - Find sailors whose rating is greater than all sailors called Horatio.



Find sid's of sailors who've reserved 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'
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'
```

- Rewrite *INTERSECT* with *IN* (plus a subquery)
  - (x IN B) returns true when x is in set B.
  - Strategy?



Rewriting *INTERSECT* Using *IN*

```
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')
```

Find sids who've reserved a green boat

- Find *sid*'s of Sailors who've reserved red but not green boats (*EXCEPT*)
  - Replace *IN* with *NOT IN*.

Division in SQL

- Find sailors who’ve reserved all boats.
- Strategy?
  - Find all boats that have been reserved by a sailor
  - Compare with all boats
  - Do the sailor’s reserved boats include all boats?
    - Yes → include this sailor
    - No → exclude this sailor

```
SELECT  S.sname
FROM  Sailors S
WHERE [redacted]
```

(A EXCEPT B) returns tuples in A but not in B.

Division in SQL

- Can you do it the hard way, without *EXCEPT* & with *NOT EXISTS*?
- Strategy:
  - For each sailor, check that there is no boat that has not been reserved by this sailor.

```
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

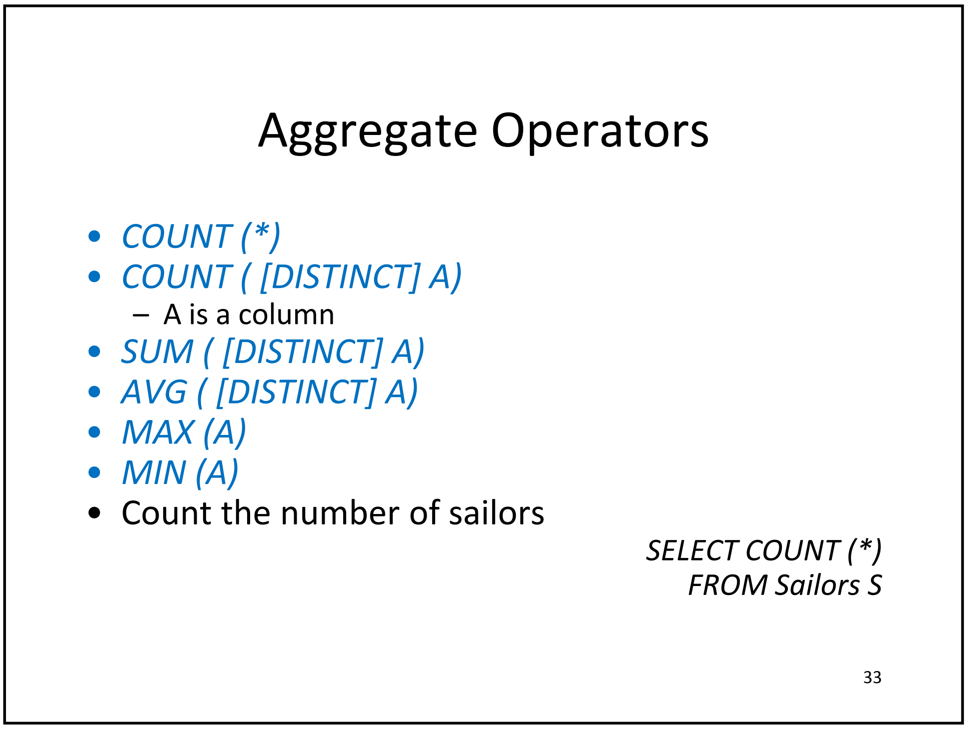
sid	*sname*	*rating*	*age*
22	*dustin*	*7*	*45.0*
31	*lubber*	*8*	*55.5*

Boats

bid	*bname*	*color*
101	*xyz*	*red*
103	*abc*	*green*

Reserves

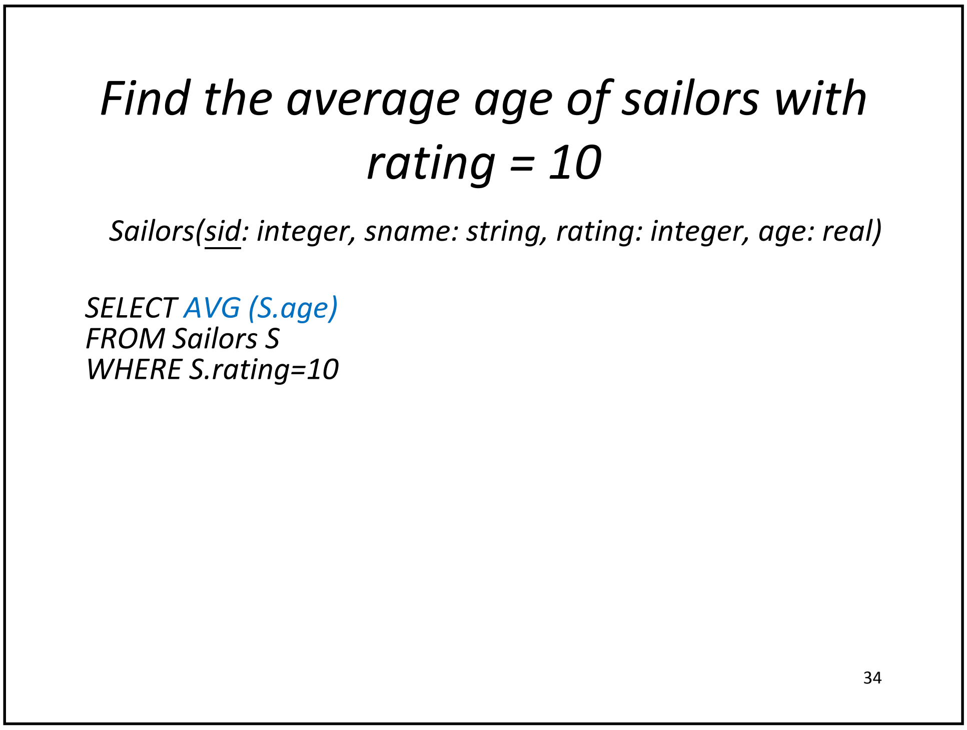
sid	*bid*	*day*
22	*101*	*10/10/96*
31	*101*	*11/12/96*
31	*103*	*12/12/96*



Aggregate Operators

- *COUNT (*)*
- *COUNT ([DISTINCT] A)*
  - A is a column
- *SUM ([DISTINCT] A)*
- *AVG ([DISTINCT] A)*
- *MAX (A)*
- *MIN (A)*
- Count the number of sailors

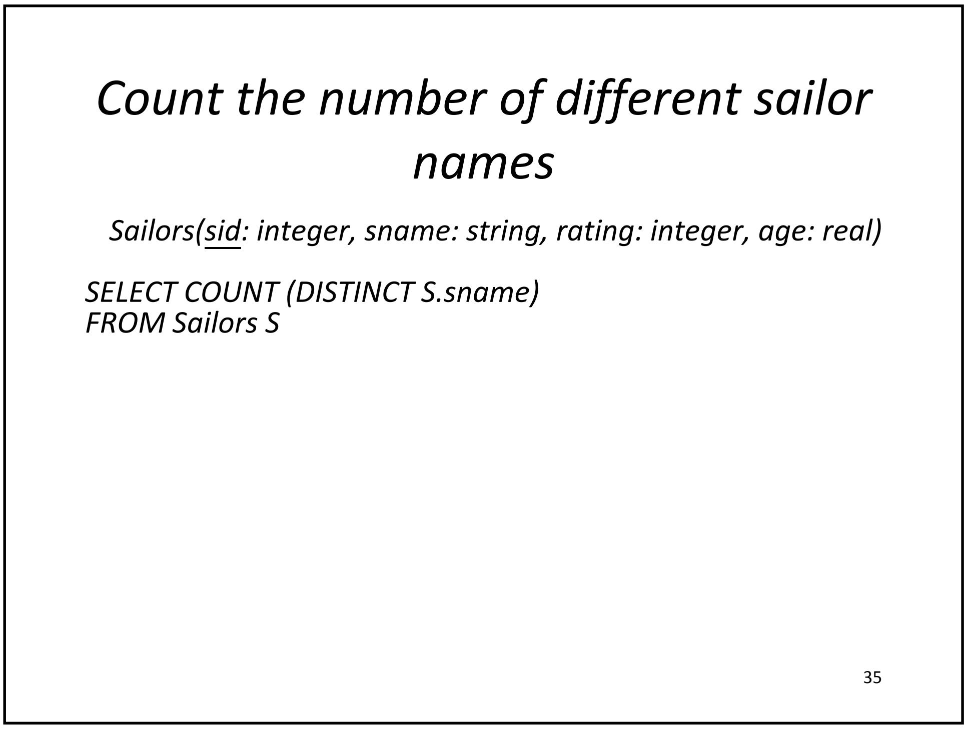
```
SELECT COUNT (*)
FROM Sailors S
```



Find the average age of sailors with rating = 10

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

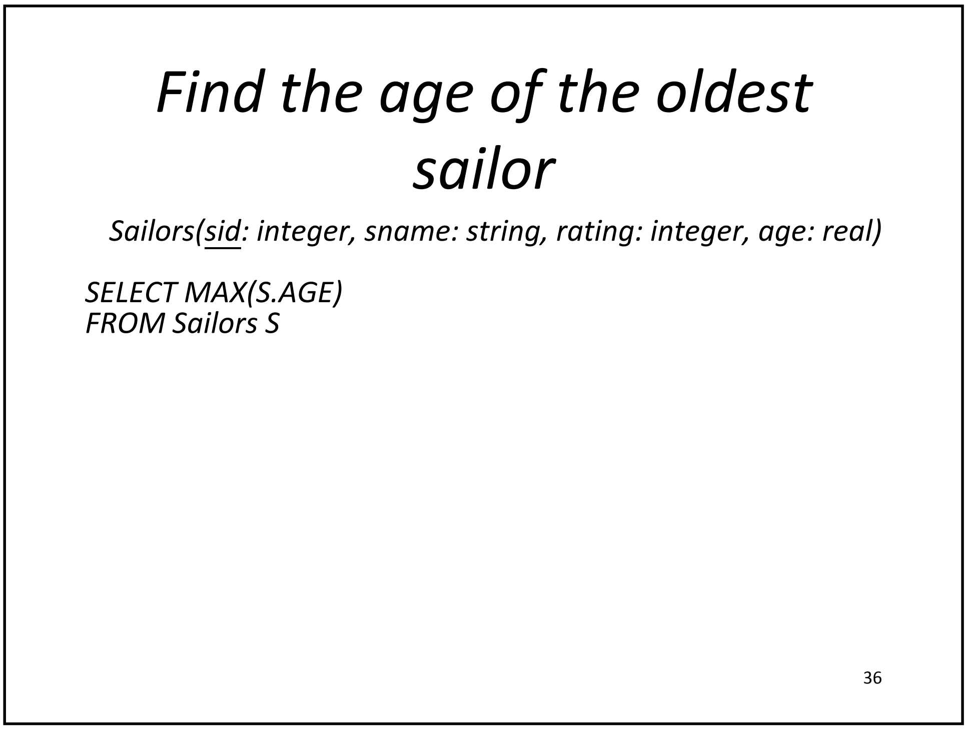
```
SELECT AVG (S.age)
FROM Sailors S
WHERE S.rating=10
```



Count the number of different sailor names

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

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



Find the age of the oldest sailor

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

```
SELECT MAX(S.AGE)
FROM Sailors S
```

Find name and age of the oldest sailor(s)

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

- This is illegal, but why?
  - Cannot combine a column with a value (unless we use *GROUP BY*)

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

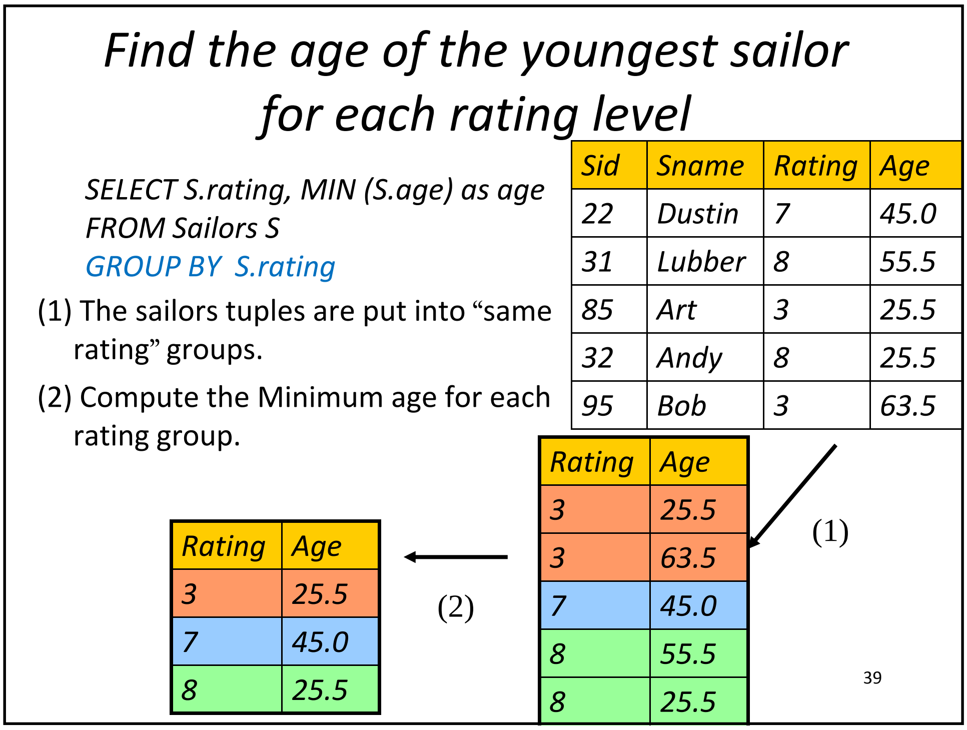
- Okay, but not supported in every system
  - Convert a table (of a single aggregate value) into a single value for comparison

GROUP BY and *HAVING*

- So far, aggregate operators are applied to all (qualifying) tuples.
  - Can we apply them to each of several groups of tuples?
- Example: 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) as age
FROM Sailors S
GROUP BY S.rating

(1) The sailors tuples are put into "same rating" groups.

(2) Compute the Minimum age for each rating group.

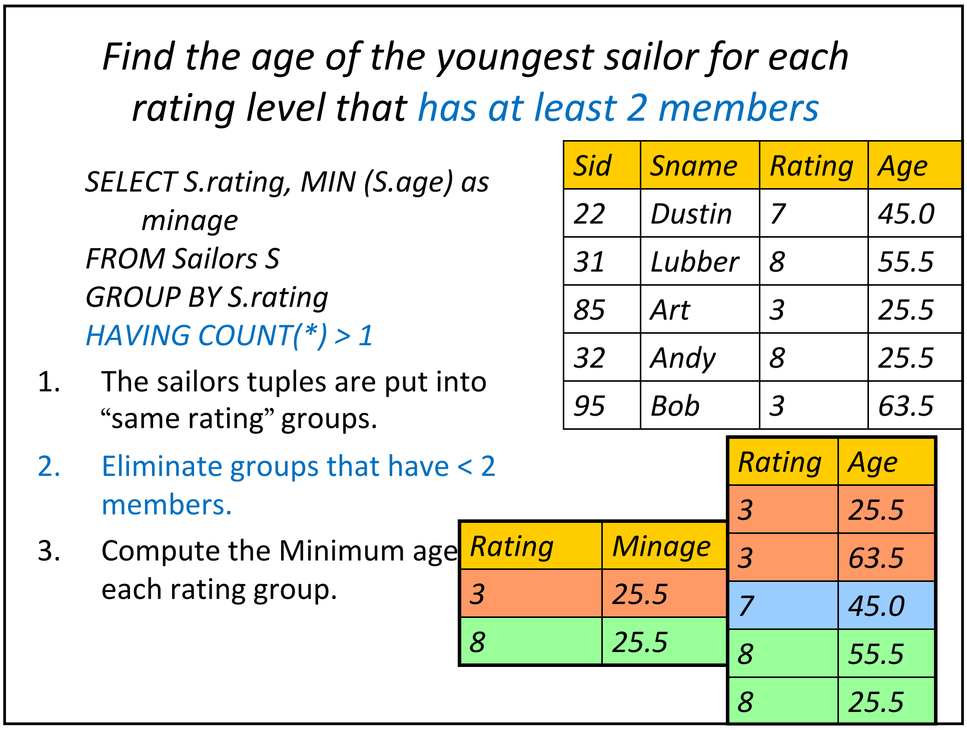
Sid	*Sname*	*Rating*	*Age*
22	*Dustin*	*7*	*45.0*
31	*Lubber*	*8*	*55.5*
85	*Art*	*3*	*25.5*
32	*Andy*	*8*	*25.5*
95	*Bob*	*3*	*63.5*

(1)

Rating	*Age*
3	*25.5*
3	*63.5*
7	*45.0*
8	*55.5*
8	*25.5*

(2)

Rating	*Age*
3	*25.5*
7	*45.0*
8	*25.5*



Find the age of the youngest sailor for each rating level that has at least 2 members

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

1. The sailors tuples are put into "same rating" groups.
2. Eliminate groups that have < 2 members.
3. Compute the Minimum age each rating group.

Sid	*Sname*	*Rating*	*Age*
22	*Dustin*	*7*	*45.0*
31	*Lubber*	*8*	*55.5*
85	*Art*	*3*	*25.5*
32	*Andy*	*8*	*25.5*
95	*Bob*	*3*	*63.5*

Rating	*Age*
3	*25.5*
3	*63.5*
7	*45.0*
8	*55.5*
8	*25.5*

Rating	*Minage*
3	*25.5*
8	*25.5*

Queries With *GROUP BY* and *HAVING*

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

```
SELECT S.rating, MIN (S.age) as age
FROM Sailors S
GROUP BY S.rating
HAVING S.rating > 5
```

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., AVG (*S.age*)).
- The attribute list (e.g., *S.rating*) in *target-list* must be in *grouping-list*.
- The attributes in group-qualification must be in *grouping-list*.

Say if Attribute list is not in grouping-list

```
SELECT S.sname, S.rating,
    AVG (S.age) as age
FROM Sailors S
GROUP BY S.rating
HAVING COUNT(S.rating) > 1
```

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
85	Art	3	25.5
32	Andy	8	25.5
95	Bob	3	63.5

Sname	Rating	Age
Art	3	25.5
Bob	3	63.5
Dustin	7	45.0
Lubber	8	55.5
Andy	8	25.5

Sname	Rating	Age
?	3	44.5
?	8	40.5

Say if attributes in the Group qualification is not in grouping-list

SELECT S.rating, AVG (S.age)
as age
FROM Sailors S
GROUP BY S.rating
HAVING S.sname ≠ 'Dustin'

Sid	*Sname*	*Rating*	*Age*
22	*Dustin*	*7*	*45.0*
31	*Lubber*	*8*	*55.5*
85	*Art*	*3*	*25.5*
32	*Andy*	*8*	*25.5*
95	*Bob*	*3*	*63.5*

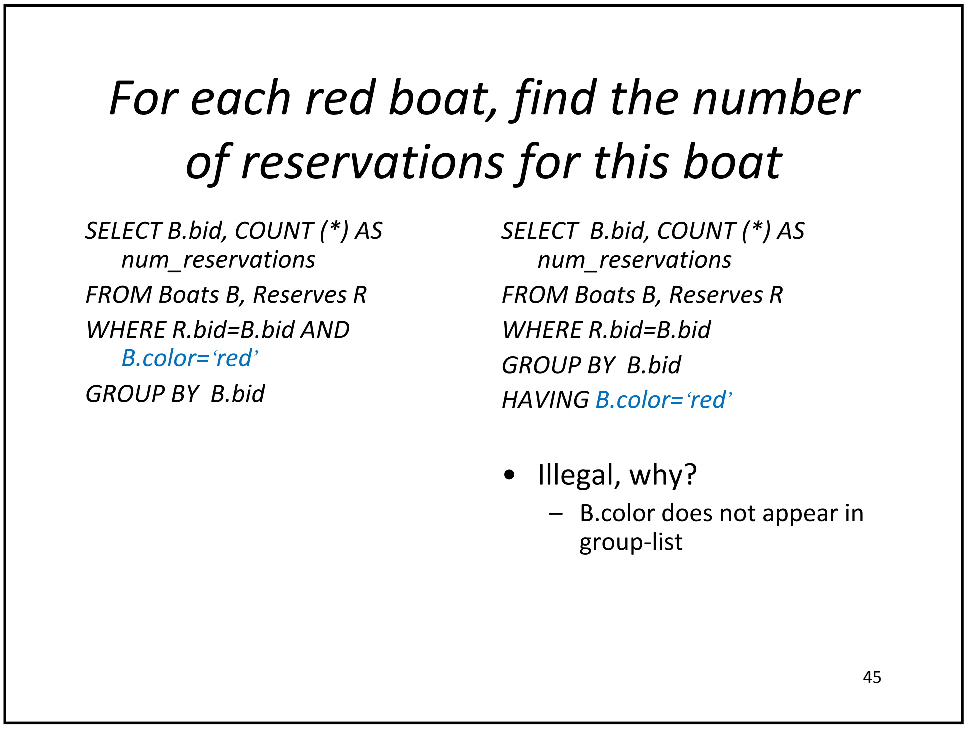
Sname	*Rating*	*Age*
Art	*3*	*25.5*
Bob	*3*	*63.5*
Dustin	*7*	*45.0*
Lubber	*8*	*55.5*
Andy	*8*	*25.5*

?

Rating	*Age*
🟧	🟧
🟩	🟩

Conceptual Evaluation

- Without *GROUP BY* and *HAVING*:
  - Compute cross-product of *relation-list*
  - Remove tuples that fail *qualification*
  - Delete unnecessary columns
- With *GROUP BY* and *HAVING*, continue with
  - Partition remaining tuples into groups by the value of attributes in *grouping-list* (specified in *GROUP-BY* clause)
  - Remove groups that fail *group-qualification* (specified in *HAVING* clause).
  - Compute one answer tuple per qualifying group.

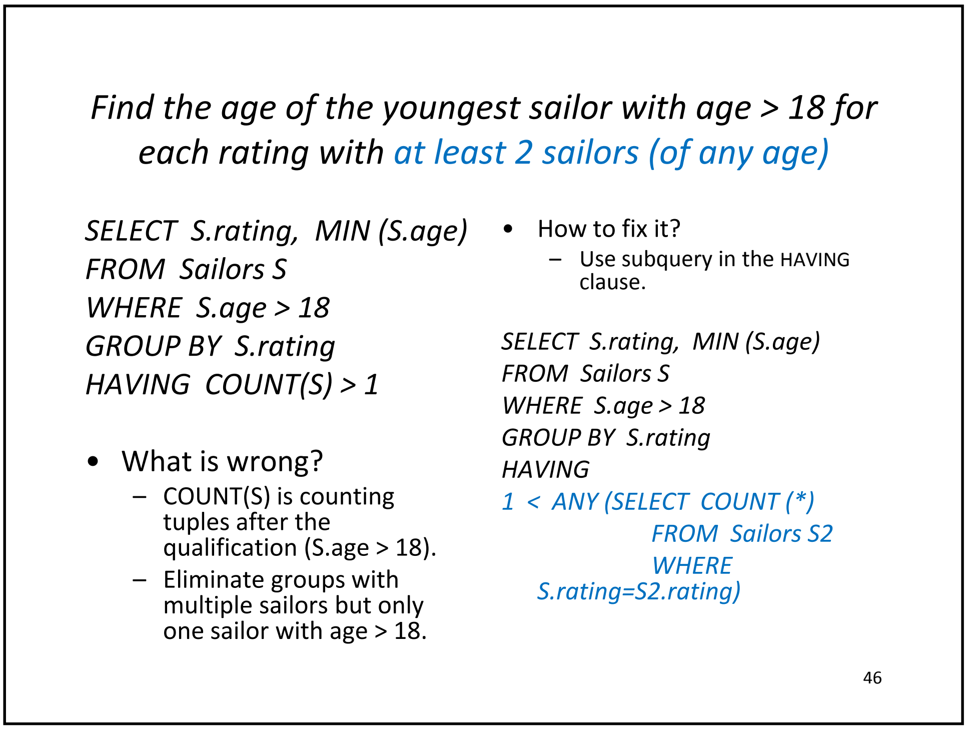


For each red boat, find the number of reservations for this boat

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

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

- Illegal, why?
  - B.color does not appear in group-list



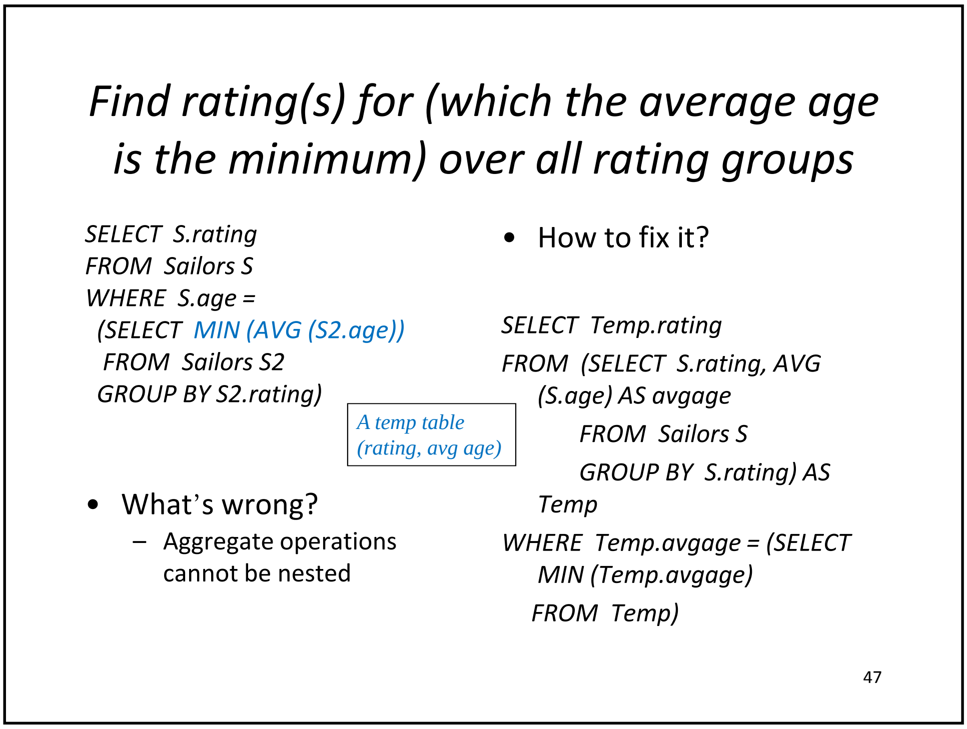
Find the age of the youngest sailor with age > 18 for each rating with at least 2 sailors (of any age)

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

- What is wrong?
  - COUNT(S) is counting tuples after the qualification (S.age > 18).
  - Eliminate groups with multiple sailors but only one sailor with age > 18.

- How to fix it?
  - Use subquery in the HAVING clause.

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



Find rating(s) for (which the average age is the minimum) over all rating groups

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

A temp table (rating, avg age)

- What's wrong?
  - Aggregate operations cannot be nested

- How to fix it?

```
SELECT  Temp.rating
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)
```

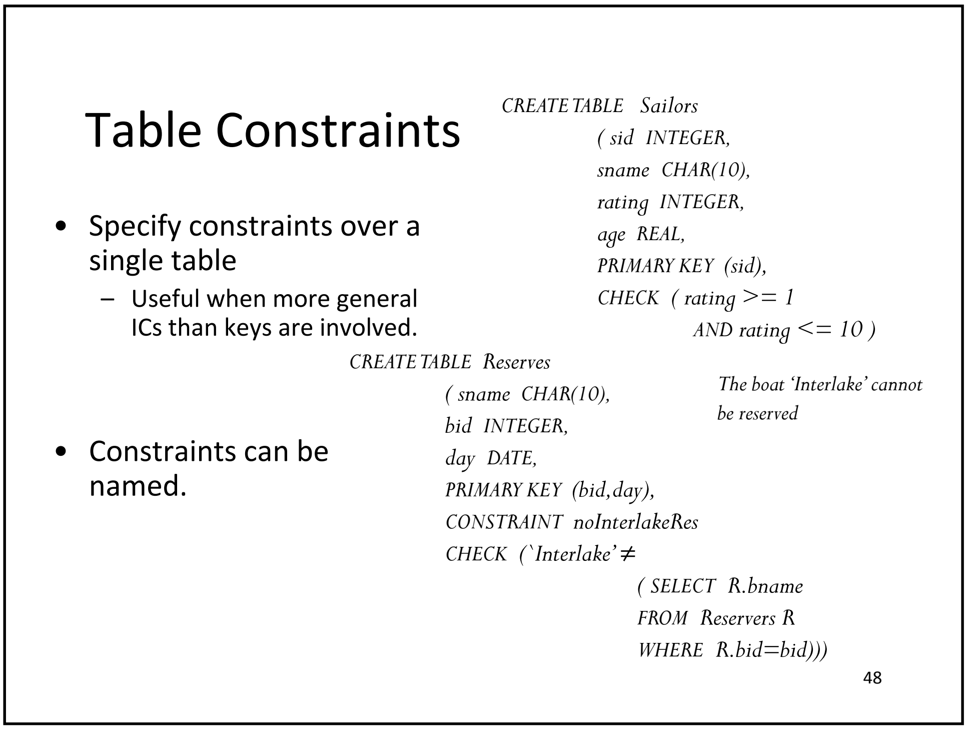


Table Constraints

- Specify constraints over a single table
  - Useful when more general ICs than keys are involved.

- Constraints can be named.

```
CREATE TABLE   Sailors
            ( sid  INTEGER,
            sname  CHAR(10),
            rating  INTEGER,
            age  REAL,
            PRIMARY KEY  (sid),
            CHECK  ( rating >= 1
                        AND rating <= 10 )
CREATE TABLE  Reserves
            ( sname  CHAR(10),
            bid  INTEGER,
            day  DATE,
            PRIMARY KEY  (bid,day),
            CONSTRAINT  noInterlakeRes
            CHECK  (`Interlake'≠
                        ( SELECT  R.bname
                        FROM  Reservers R
                        WHERE  R.bid=bid)))
```

The boat 'Interlake' cannot be reserved

Assertions: Constraints Over Multiple Tables

- Awkward and wrong!
  - If Sailors is empty, the number of Boats tuples can be anything!
- ASSERTION is the right solution; not associated with either table.

Number of boats plus number of sailors is < 100

```
CREATE TABLE   Sailors
        ( sid  INTEGER,
        sname  CHAR(10),
        rating  INTEGER,
        age  REAL,
        PRIMARY KEY  (sid),
        CHECK
        ( (SELECT COUNT (S.sid) FROM Sailors S)
        + (SELECT COUNT (B.bid) FROM Boats B) < 100 )
```

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

Triggers

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

```
CREATE TRIGGER incr_count AFTER INSERT ON Students // Event
WHEN (new.age < 18)  // Condition
FOR EACH ROW
   BEGIN   // ACTION: a procedure in Oracle's PL/SQL syntax
        count := count + 1
   END
```

Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

Which planet does Princess Leia go to in movie3?

```
SELECT distinct pname
FROM timetable
WHERE cname ='Princess Leia' and movie=3
```

51

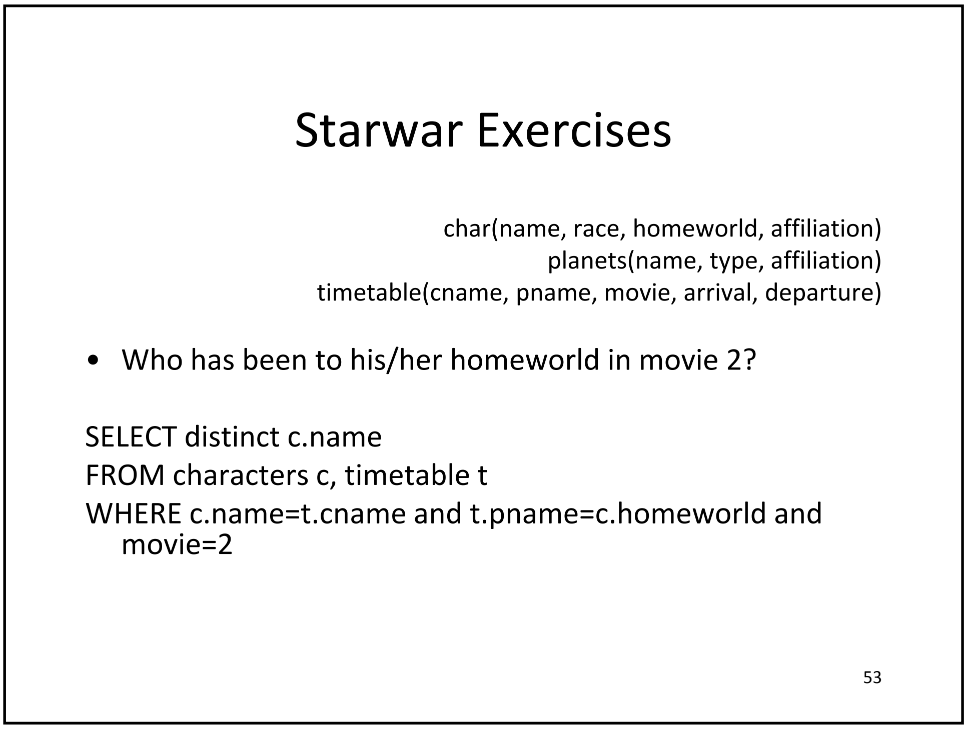
Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

- How many people stay on Dagobah in movie 3?

```
SELECT count(*)
FROM timetable, characters
WHERE movie=3 and pname ='Dagobah' and
    timetable.cname=characters.name and
    characters.race='Human'
```

52

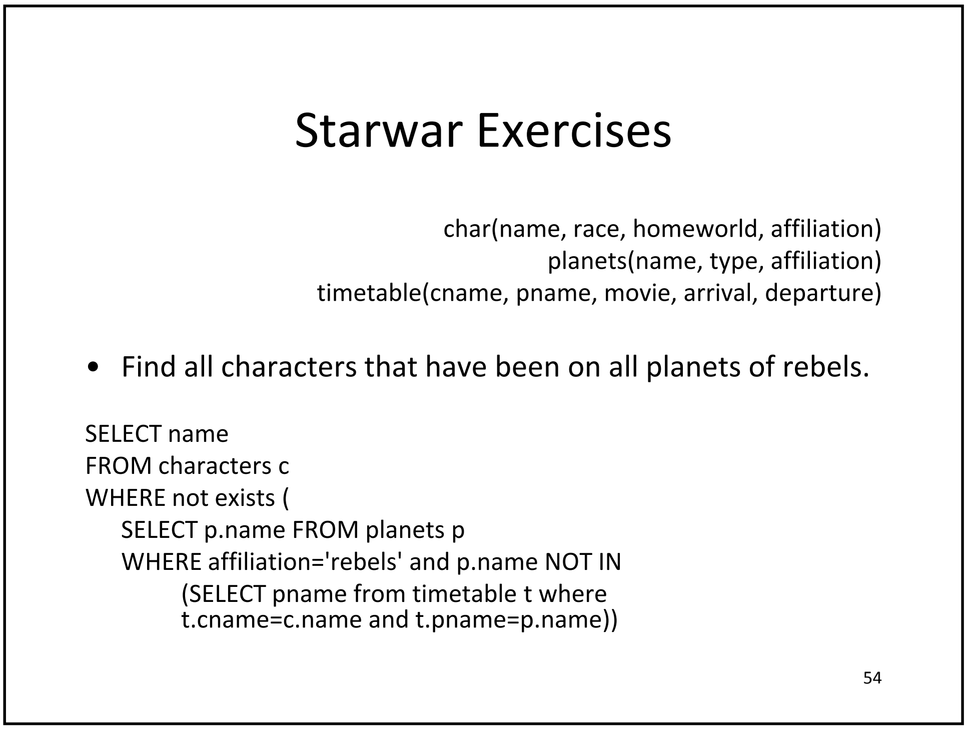


Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

- Who has been to his/her homeworld in movie 2?

```
SELECT distinct c.name
FROM characters c, timetable t
WHERE c.name=t.cname and t.pname=c.homeworld and
    movie=2
```



Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

- Find all characters that have been on all planets of rebels.

```
SELECT name
FROM characters c
WHERE not exists (
    SELECT p.name FROM planets p
    WHERE affiliation='rebels' and p.name NOT IN
          (SELECT pname from timetable t where
          t.cname=c.name and t.pname=p.name))
```

Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

- Find distinct names of the planets visited by those of race “droid”.

```
SELECT distinct t.pname
FROM char c, timetable t
WHERE c.name=t.cname and c.race='droid'
```

Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

- For each character and for each neutral planet, how much time total did the character spend on the planet?

```
SELECT c.name, p.name, SUM(t.departure-t.arrival+1) as amount
FROM characters c, timetable t, planets p
WHERE t.cname=c.name and t.pname=p.name and p.affiliation='neutral'
GROUP BY c.name, p.name
```