SQL: The Query Language

R & G - Chapter 5

Relational Tables

- *Schema* is fixed:
  - attribute names, *atomic* types
  - `students(name text, gpa float, dept text)`
- *Instance* can change
  - a *multi*set of "rows" ("tuples")

    -
      ```
      {('Bob Snob', 3.3,'CS'),
       ('Bob Snob', 3.3,'CS'),
       ('Mary Contrary', 3.8, 'CS')}
      ```

Basic Single-Table Queries

```
SELECT [DISTINCT] <column expression list>
  FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>] ]
[ORDER BY <column list>];
```

Basic Single-Table Queries

```
SELECT [DISTINCT] <column expression list>
  FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>] ]
[ORDER BY <column list>] ;
```

Simplest version is straightforward

- Produce all tuples in the table that satisfy the predicate
- Output the expressions in the SELECT list
- Expression can be a column reference, or an arithmetic expression over column refs

Basic Single-Table Queries

```
SELECT S.name, S.gpa
  FROM students AS S
 WHERE S.dept = 'CS'
[GROUP BY <column list>
 [HAVING <predicate>] ]
[ORDER BY <column list>] ;
```

Simplest version is straightforward

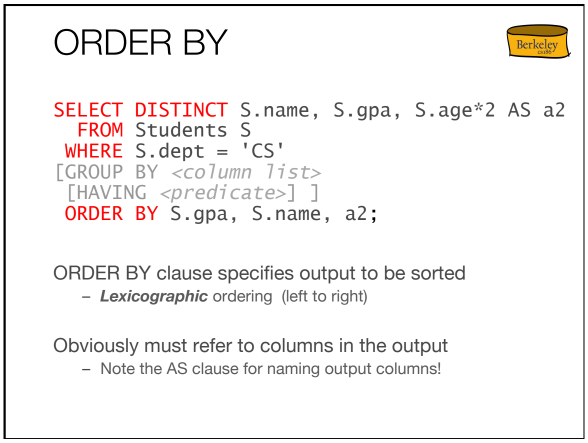
- Produce all tuples in the table that satisfy the predicate
- Output the expressions in the SELECT list
- Expression can be a column reference, or an arithmetic expression over column refs

SELECT DISTINCT

```
SELECT DISTINCT S.name, S.gpa
  FROM students S
 WHERE S.dept = 'CS'
[GROUP BY <column list>
 [HAVING <predicate>] ]
[ORDER BY <column list>] ;
```

DISTINCT flag specifies removal of duplicates before output

Removed the "AS" from FROM clause --- it's optional



ORDER BY

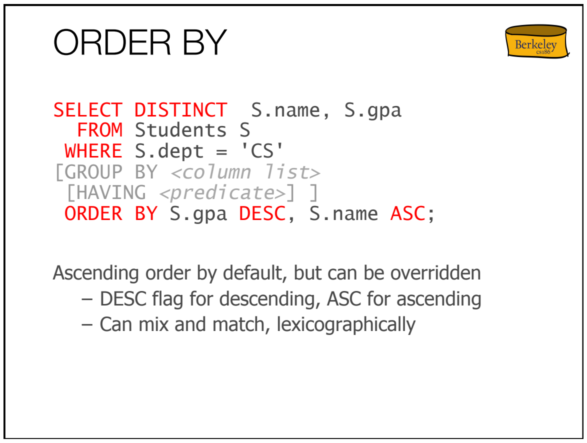
```
SELECT DISTINCT S.name, S.gpa, S.age*2 AS a2
  FROM Students S
 WHERE S.dept = 'CS'
[GROUP BY <column list>
 [HAVING <predicate>] ]
 ORDER BY S.gpa, S.name, a2;
```

ORDER BY clause specifies output to be sorted

- ***Lexicographic*** ordering (left to right)

Obviously must refer to columns in the output

- Note the AS clause for naming output columns!

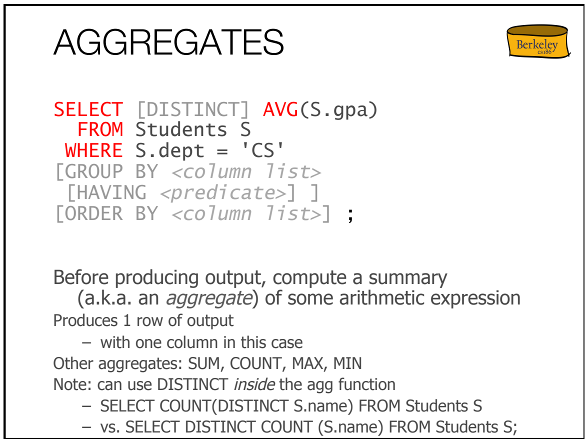


ORDER BY

```
SELECT DISTINCT  S.name, S.gpa
  FROM Students S
 WHERE S.dept = 'CS'
[GROUP BY <column list>
 [HAVING <predicate>] ]
 ORDER BY S.gpa DESC, S.name ASC;
```

Ascending order by default, but can be overridden

- DESC flag for descending, ASC for ascending
- Can mix and match, lexicographically



AGGREGATES

```
SELECT [DISTINCT] AVG(S.gpa)
  FROM Students S
 WHERE S.dept = 'CS'
[GROUP BY <column list>
 [HAVING <predicate>] ]
[ORDER BY <column list>] ;
```

Before producing output, compute a summary
(a.k.a. an *aggregate*) of some arithmetic expression

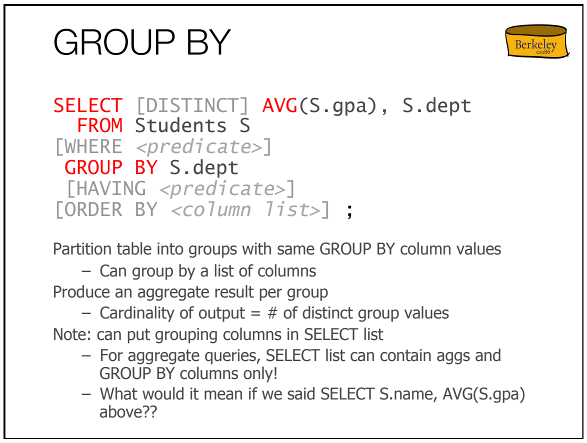
Produces 1 row of output

- with one column in this case

Other aggregates: SUM, COUNT, MAX, MIN

Note: can use DISTINCT *inside* the agg function

- SELECT COUNT(DISTINCT S.name) FROM Students S
- vs. SELECT DISTINCT COUNT (S.name) FROM Students S;



GROUP BY

```
SELECT [DISTINCT] AVG(S.gpa), S.dept
  FROM Students S
[WHERE <predicate>]
 GROUP BY S.dept
 [HAVING <predicate>]
[ORDER BY <column list>] ;
```

Partition table into groups with same GROUP BY column values

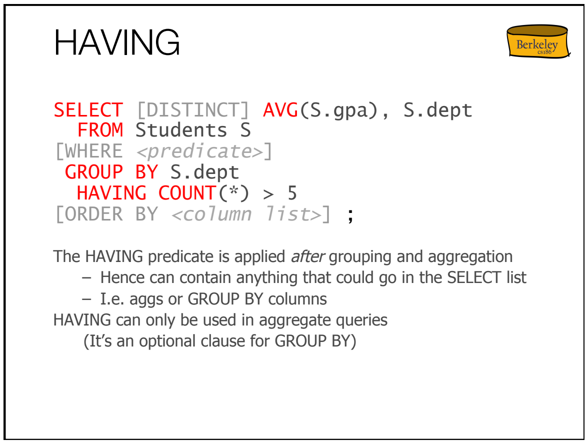
- Can group by a list of columns

Produce an aggregate result per group

- Cardinality of output = # of distinct group values

Note: can put grouping columns in SELECT list

- For aggregate queries, SELECT list can contain aggs and GROUP BY columns only!
- What would it mean if we said SELECT S.name, AVG(S.gpa) above??



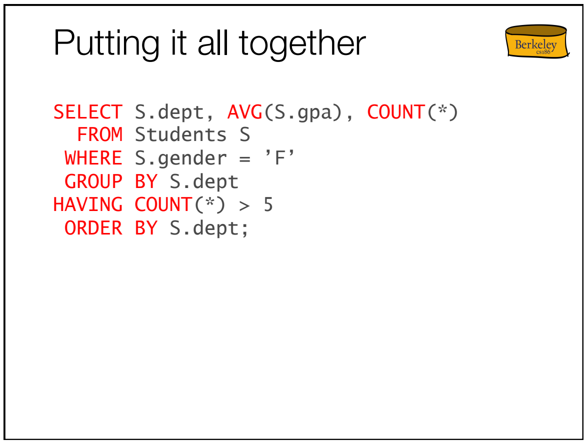
HAVING

```
SELECT [DISTINCT] AVG(S.gpa), S.dept
  FROM Students S
[WHERE <predicate>]
 GROUP BY S.dept
  HAVING COUNT(*) > 5
[ORDER BY <column list>] ;
```

The HAVING predicate is applied *after* grouping and aggregation

- Hence can contain anything that could go in the SELECT list
- I.e. aggs or GROUP BY columns

HAVING can only be used in aggregate queries
(It’s an optional clause for GROUP BY)



Putting it all together

```
SELECT S.dept, AVG(S.gpa), COUNT(*)
  FROM Students S
 WHERE S.gender = ’F’
 GROUP BY S.dept
HAVING COUNT(*) > 5
 ORDER BY S.dept;
```

Relational Query Languages

Two sublanguages:

- DDL – Data Definition Language
  - Define and modify schema
- DML – Data Manipulation Language
  - Write declarative queries/updates
  - We just covered basic queries is the SQL DML

DBMS is responsible for efficient evaluation

- Semantics are precise (more on that later)
- Declarative language => room for optimization
- Optimizer can re-order operations
  - Won't affect query answer

Example Database

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Boats

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

Reserves

sid	bid	day
1	102	9/12/2015
2	102	9/13/2015

SQL DDL

```
CREATE TABLE Sailors (
   sid INTEGER,
   sname CHAR(20),
   rating INTEGER,
   age REAL,
   PRIMARY KEY (sid));

CREATE TABLE Boats (
   bid INTEGER,
   bname CHAR (20),
   color CHAR(10),
   PRIMARY KEY (bid));

 CREATE TABLE Reserves (
   sid INTEGER,
   bid INTEGER,
   day DATE,
  PRIMARY KEY (sid, bid, day),
  FOREIGN KEY (sid) REFERENCES Sailors,
  FOREIGN KEY (bid) REFERENCES Boats);
```

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

sid	**bid**	**day**
1	102	9/12
2	102	9/13

Querying Multiple Relations

```
SELECT S.sname
FROM   Sailors AS S, Reserves AS R
WHERE  S.sid=R.sid AND R.bid=102
```

Sailors

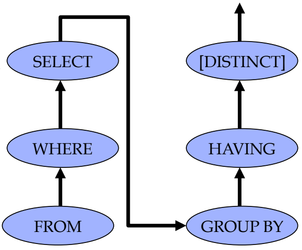
sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Conceptual SQL Evaluation

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



Project away columns (just keep those used in SELECT, GROUP BY, HAVING)

Apply selections (eliminate rows)

Relation cross-product

Eliminate duplicates

Eliminate groups

Form groups & aggregate

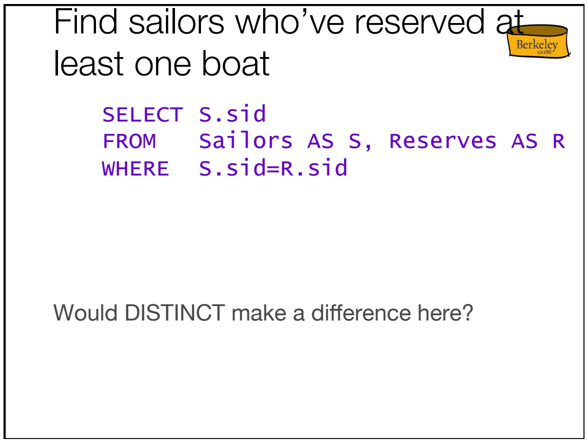
Query Semantics

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

1. FROM : compute cross product of tables.
2. WHERE : Check conditions, discard tuples that fail.
3. SELECT : Delete unwanted fields.
4. DISTINCT (optional) : eliminate duplicate rows.

Note: likely a terribly inefficient strategy!

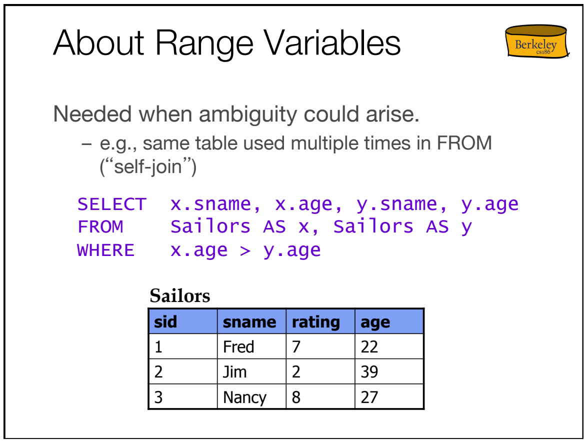
- Query optimizer will find more efficient plans.



Find sailors who've reserved at least one boat

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

Would DISTINCT make a difference here?



About Range Variables

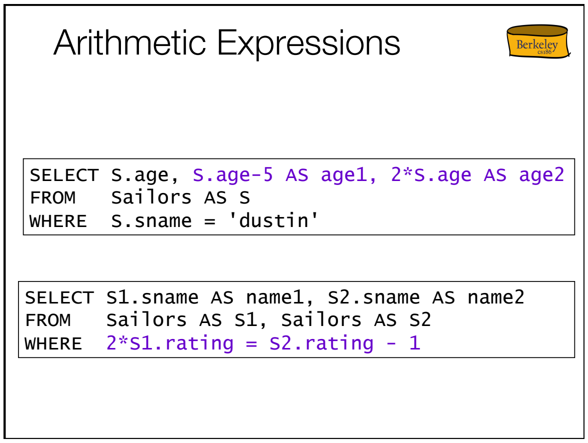
Needed when ambiguity could arise.

- e.g., same table used multiple times in FROM ("self-join")

```
SELECT  x.sname, x.age, y.sname, y.age
FROM    Sailors AS x, Sailors AS y
WHERE   x.age > y.age
```

Sailors

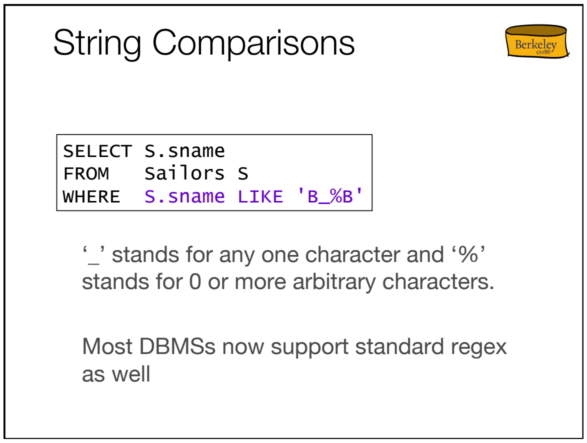
sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27



Arithmetic Expressions

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
FROM   Sailors AS S
WHERE  S.sname = 'dustin'
```

```
SELECT S1.sname AS name1, S2.sname AS name2
FROM   Sailors AS S1, Sailors AS S2
WHERE  2*S1.rating = S2.rating - 1
```

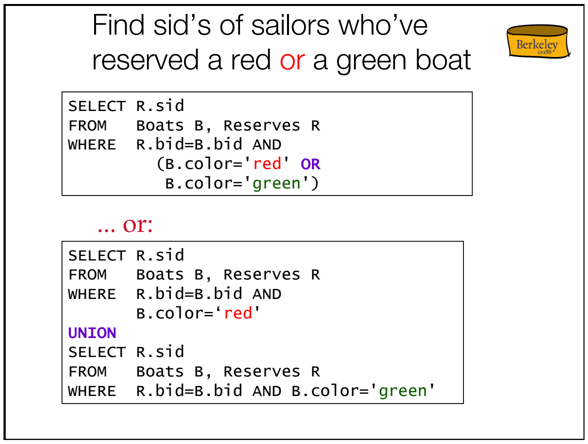


String Comparisons

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sname LIKE 'B_%B'
```

'_' stands for any one character and '%' stands for 0 or more arbitrary characters.

Most DBMSs now support standard regex as well

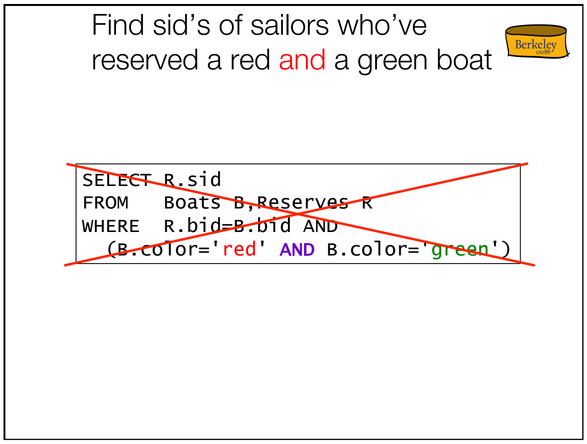


Find sid's of sailors who've reserved a red or a green boat

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

... or:

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



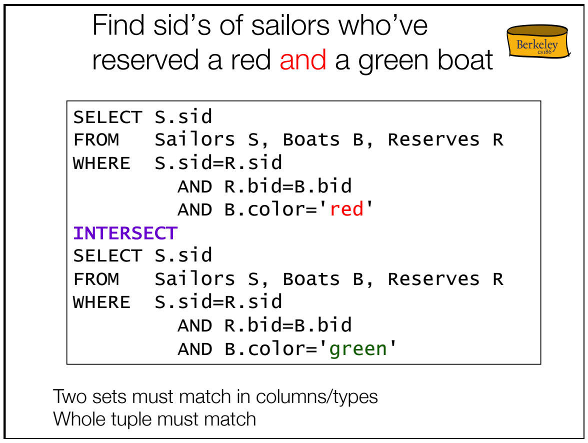
Find sid's of sailors who've reserved a red and a green boat

~~SELECT R.sid~~

~~FROM Boats B, Reserves R~~

~~WHERE R.bid=B.bid AND~~

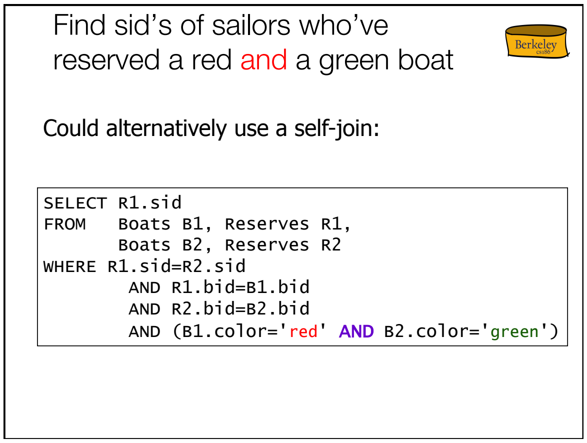
~~(B.color='red' AND B.color='green')~~



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'
```

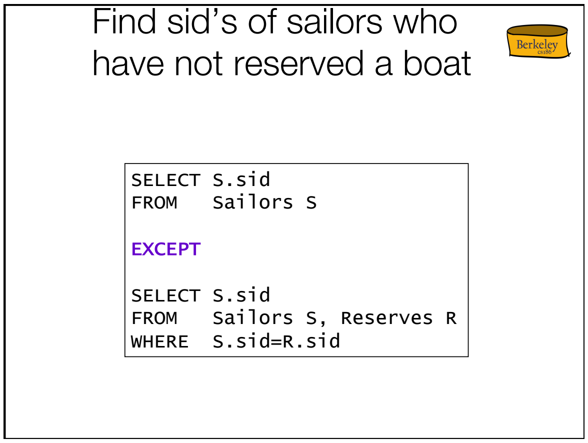
Two sets must match in columns/types
Whole tuple must match



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

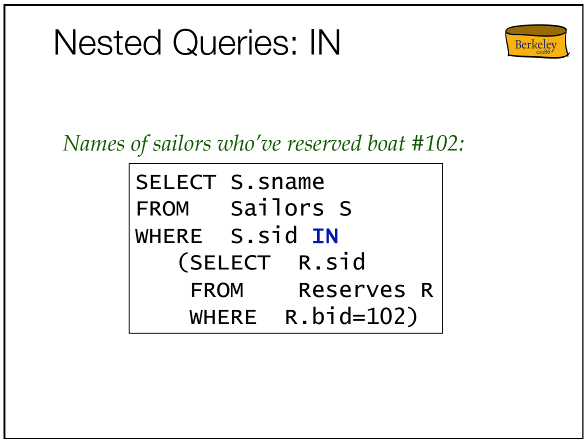
Could alternatively use a self-join:

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



Find sid's of sailors who have not reserved a boat

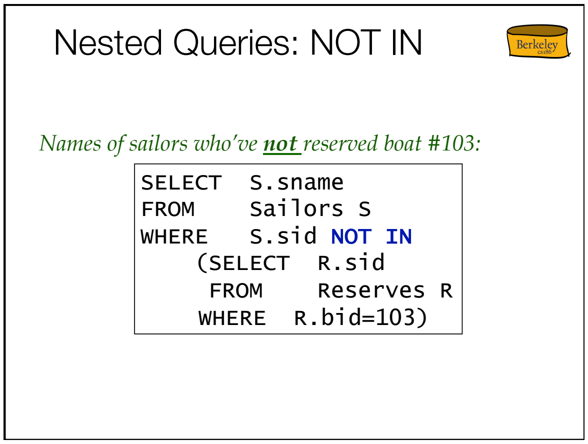
```
SELECT S.sid
FROM   Sailors S

EXCEPT

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



Nested Queries: IN

Names of sailors who've reserved boat #102:

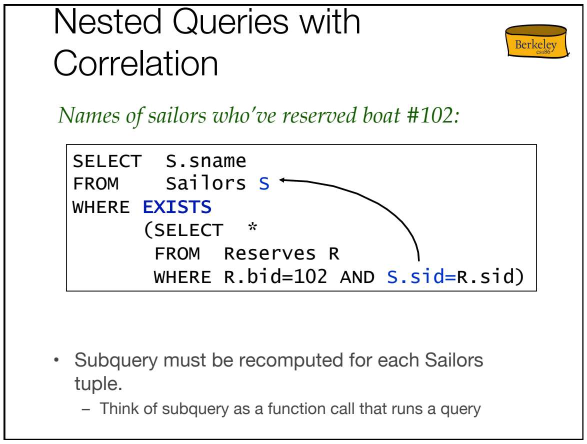
```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN
   (SELECT  R.sid
    FROM    Reserves R
    WHERE   R.bid=102)
```



Nested Queries: NOT IN

Names of sailors who've ***not*** *reserved boat #103:*

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



Nested Queries with Correlation

Names of sailors who've reserved boat #102:

```
SELECT   S.sname
FROM     Sailors S
WHERE EXISTS
      (SELECT  *
       FROM  Reserves R
       WHERE R.bid=102 AND S.sid=R.sid)
```

- Subquery must be recomputed for each Sailors tuple.
  - Think of subquery as a function call that runs a query

More on Set-Comparison Operators

- we've seen: IN, EXISTS
- can also have: NOT IN, NOT EXISTS
- other forms: op ANY, op ALL

Find sailors whose rating is greater than that of *some* sailor called Fred:

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

A Tough One

Find sailors who've reserved all boats.

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

ARGMAX?

The sailor with the highest rating

– what about ties for highest?!

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

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

```
SELECT *
FROM   Sailors S
ORDER BY rating DESC
LIMIT 1;
```

Null Values

Field values are sometimes unknown or inapplicable

- SQL provides a special value null for such situations.

The presence of null complicates many issues. E.g.:

- Special syntax "IS NULL" and "IS NOT NULL"
- Assume rating = NULL. Consider predicate "rating>8".
  - True? False? (answer is always *false*)
  - What about AND, OR and NOT connectives?
  - SUM?
- We need a 3-valued logic (true, false and unknown).
- Meaning of constructs must be defined carefully. (e.g., WHERE clause eliminates rows that don't evaluate to true.)
- New operators (in particular, outer joins) possible/needed.