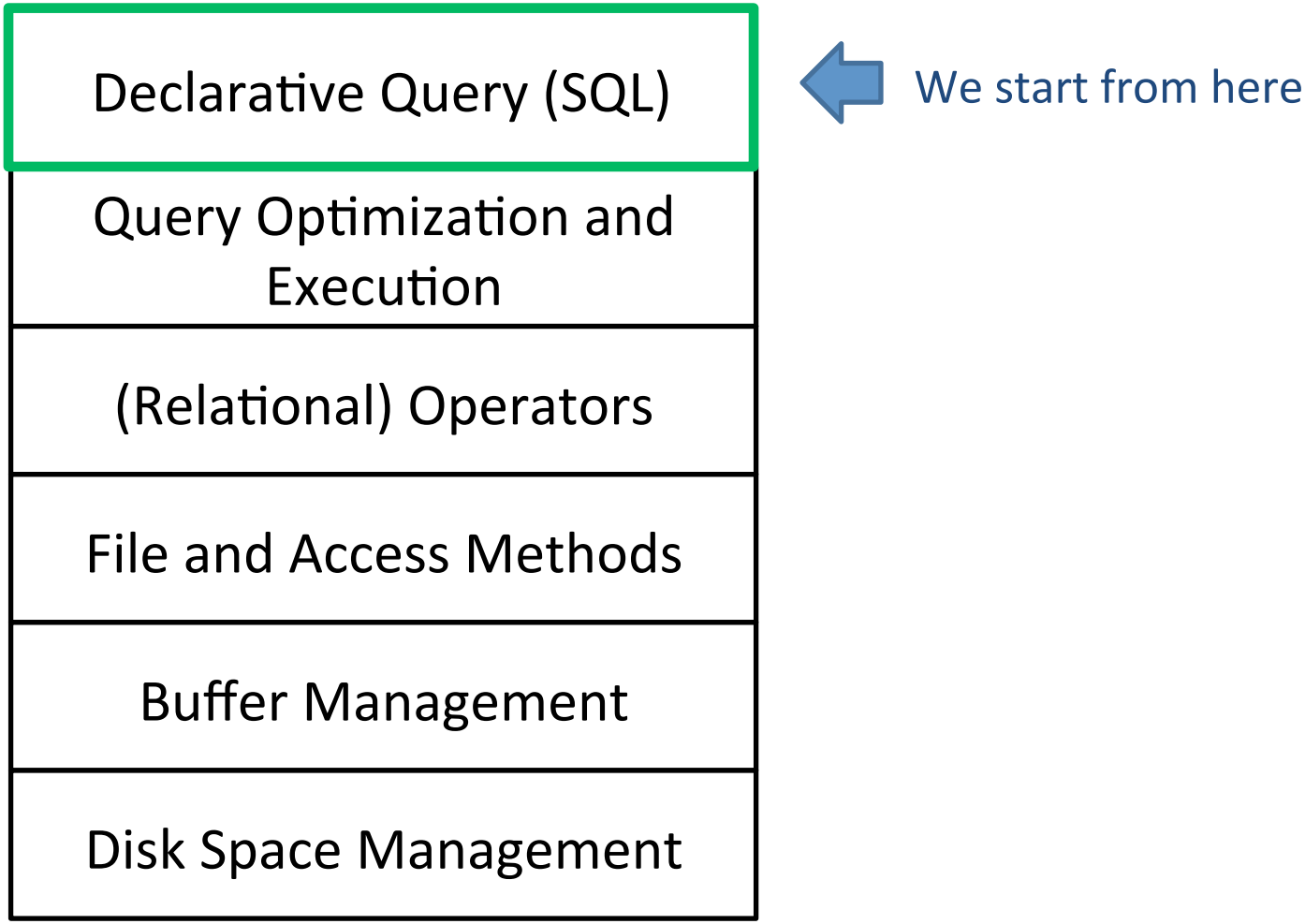
SQL

The Query Language

R & G - Chapter 5

Based on Slides from UC Berkeley and book.

Query Execution



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>] ;
```

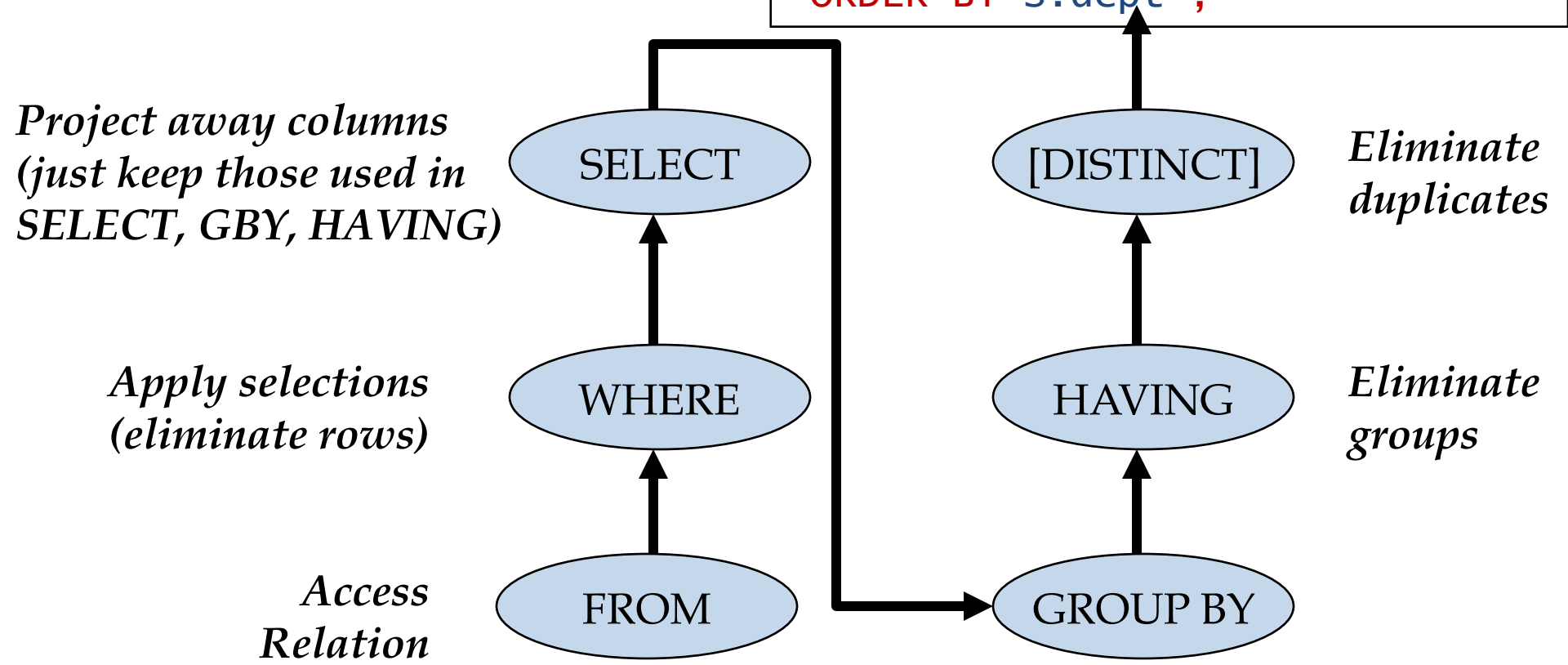
- The HAVING predicate is applied after grouping and aggregation
  - Hence can contain anything that could go in the SELECT list
  - That is, aggs or GROUP BY columns
- HAVING can only be used in aggregate queries
- It's an optional clause

Putting it all together

```
SELECT S.dept, AVG(S.gpa), COUNT(*)
  FROM students S
 WHERE S.gender = 'F'
 GROUP BY S.dept
 HAVING COUNT(*) > 2
 ORDER BY S.dept ;
```

Conceptual SQL Evaluation


```
SELECT S.dept, AVG(S.gpa),
       COUNT(*)
  FROM students S
 WHERE S.gender = 'F'
 GROUP BY S.dept
 HAVING COUNT(*) > 2
 ORDER BY S.dept ;
```



Multi-relation Queries

Querying Multiple Relations

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

Sailors

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

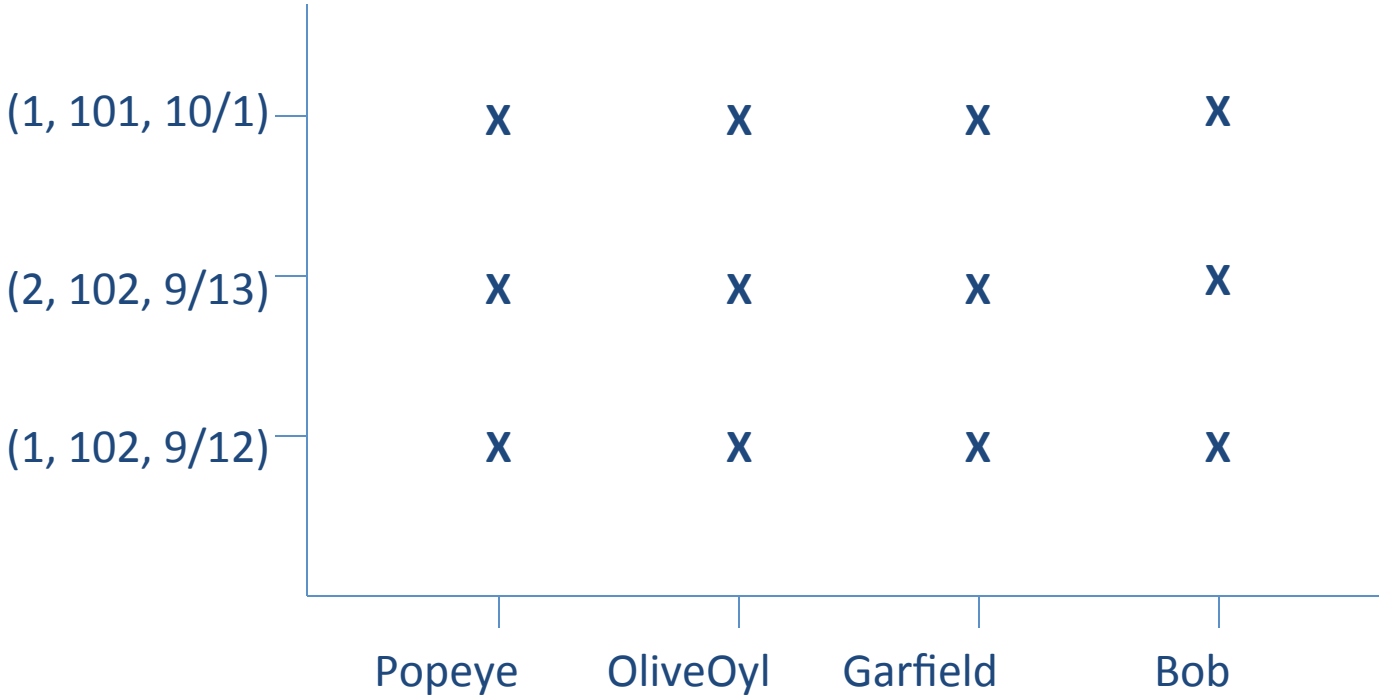
Reserves

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01
  ```

Querying Multiple Relations

```
SELECT S.sname
  FROM Sailors S, Reserves R
```

- Cartesian product



Meaning (Semantics) of SQL Queries

```
SELECT x1.a1, x1.a2, …, xn.ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions(x1,…, xn)
```

Almost never the *fastest* way to compute it!

```
Answer = {}
for x1 in R1 do
    for x2 in R2 do
        …..
            for xn in Rn do
                if Conditions(x1 …, xn)
                    then Answer = Answer ∪
   {(x1.a1, x1.a2, …, xn.ak)}
return Answer
```

Note: this is a *multiset* union

Join Queries

- SELECT [DISTINCT] *<column expression list>*
  FROM *<table1 [AS t1], ... , tableN [AS tn]>*
  [WHERE *<predicate>*]
  [GROUP BY *<column list>*]
  [HAVING *<predicate>*]
  [ORDER BY *<column list>*] ;

Query Semantics

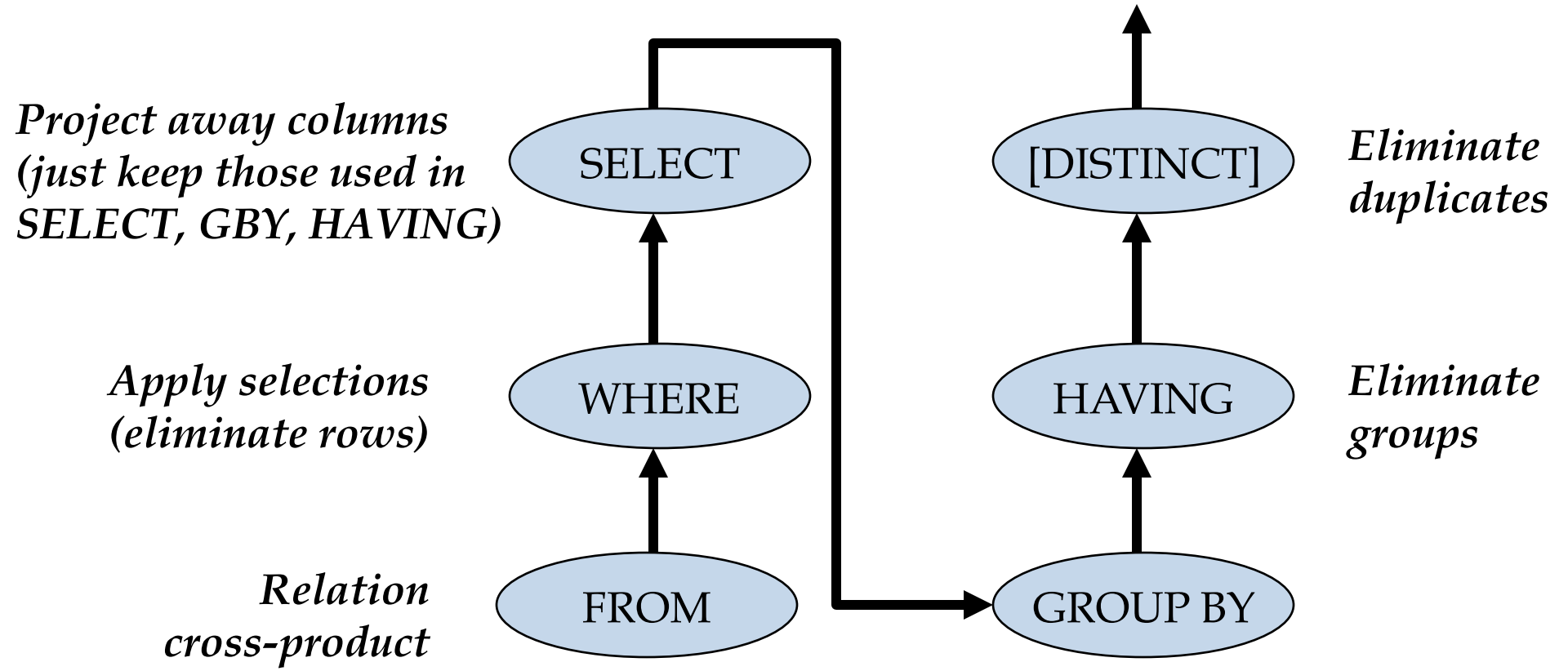
```
SELECT  [DISTINCT]  target-list
FROM        relation-list
WHERE       qualification
```

- FROM: compute cross product of tables.
- WHERE: Check conditions, discard tuples that fail.
- SELECT: Specify desired fields in output.
- DISTINCT (optional): eliminate duplicate rows.

- Note: this is likely a terribly inefficient strategy!
  - Query optimizer will find more efficient plans.

Conceptual SQL Evaluation


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



Find sailors who have reserved at least one boat

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

Will 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	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

Arithmetic Expressions

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

```
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 'P_p%'
```

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

Most DBMSs now support standard regex as well (incl. PostgreSQL)

Find sid of sailors who've reserved a red or 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 ALL
SELECT R.sid
FROM   Boats B, Reserves R
WHERE  R.bid=B.bid AND B.color='green'
```

Find sid 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 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 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'
```

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

- Could 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 sids 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 have 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 'Popeye'

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

A Tougher Query

Find sailors who've reserved ALL boats

(relational division: no "counterexample 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 that ...

... is missing a Reserves tuple showing that S reserved B

A Tougher Query

Find sailors who've reserved ALL boats

(here we use set difference: from all the boats remove the ones that Sailor S has reserved. If empty, then S is good)

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

Sailors *S* such that...

```
WHERE  NOT EXISTS
     ((SELECT  B.bid
       FROM  Boats B)
```

there is no boat B that ...

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

that has not been reserved by S

A Tougher Query

Find sailors who've reserved ALL boats

(here we use count aggregates: count the total number of boats and the number of boats reserved by S)

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

Sailors *S* such that...

```
WHERE (SELECT  COUNT(B.bid)
       FROM  Boats B) =
      (SELECT COUNT (DISTINCT R.bid)
       FROM Reserves R
       WHERE R.sid=S.sid)
```

total number of boats equal to the number of distinct boats reserved by S

ARGMAX?

- The Sailor with the highest rating
  - What about ties for highest?

```
SELECT  MAX(S.rating)
  FROM  Sailors S; -- OK

SELECT  S.*, MAX(S.rating)
  FROM  Sailors S; -- Not OK
```

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 IS NULL. Consider predicate “rating>8”.
    - True? 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.

NULL Values: Truth table

p	*q*	*p* OR *q*	*p* AND *q*	*p* = *q*
TRUE	**TRUE**	TRUE	TRUE	TRUE
TRUE	**FALSE**	TRUE	FALSE	FALSE
TRUE	**Unknown**	TRUE	Unknown	Unknown
FALSE	**TRUE**	TRUE	FALSE	FALSE
FALSE	**FALSE**	FALSE	FALSE	TRUE
FALSE	**Unknown**	Unknown	FALSE	Unknown
Unknown	**TRUE**	TRUE	Unknown	Unknown
Unknown	**FALSE**	Unknown	FALSE	Unknown
Unknown	**Unknown**	Unknown	Unknown	Unknown