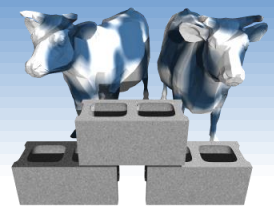
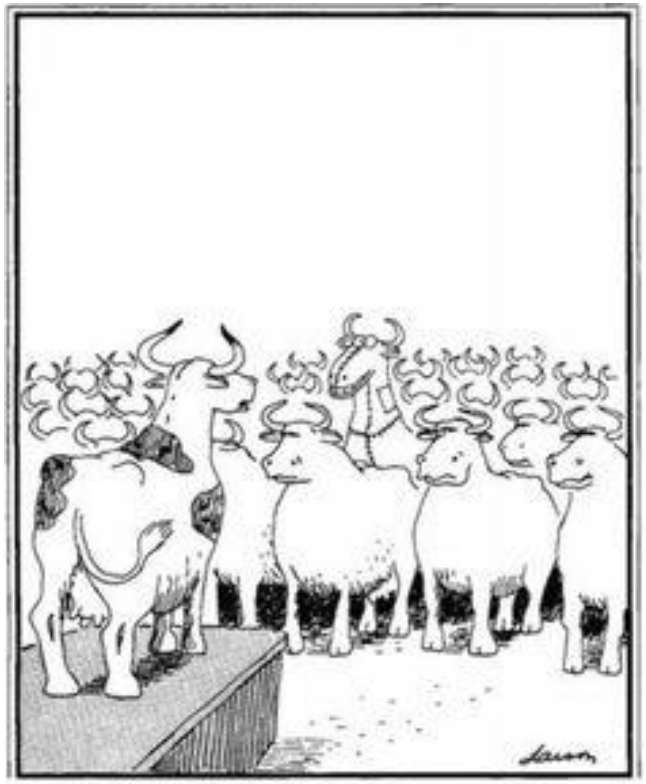


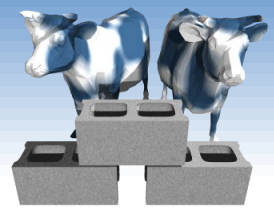
SQL: Advanced Queries

Chapter 5.4-5.5



"The revolution has been postponed . . . We've discovered a leak."

SQL's Aggregate Operators

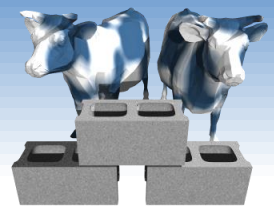


- Significant extension of relational algebra.
- Computation and summarization operations
- Appears in *target-list* of query
- Results *aggregate* rather than appear individually
- E.x. How many instances in the sailor relation?

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

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

single column



More examples

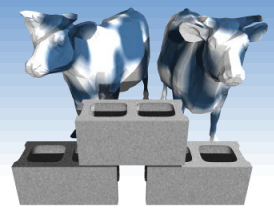
- Average age of Sailors with a rating of 10?

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

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

- Names of all Sailors who have achieved the maximum rating

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



More examples (cont)

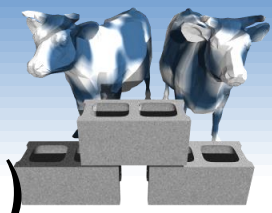
- How many distinct ratings for Sailors less than 40 years of age?

```
SELECT  COUNT(DISTINCT S.rating)
FROM  Sailors S
WHERE S.age < 40.0
```

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

- How many reservations were made by Sailors less than 40 years old?

```
SELECT  COUNT(*)
FROM  Sailors S, Reserves R
WHERE  S.sid = R.sid AND S.age < 40
```



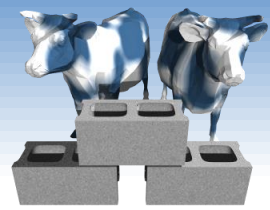
Find name and age of the oldest sailor(s)

- The first query is incorrect! (Switch the MAX to MIN to see)
- 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
```



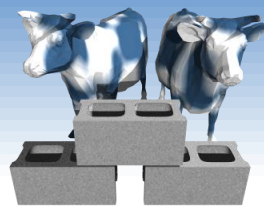
Motivation for Grouping

- So far, we've applied aggregate operators to *all* (qualifying) tuples. Sometimes, we want to apply them to *subgroups*.
- 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 (!):

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

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

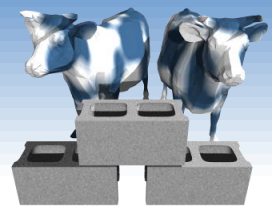
Queries With GROUP BY *and* HAVING



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

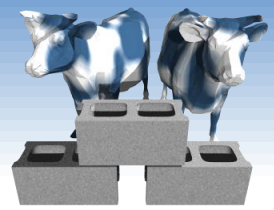
- The *target-list* contains
  (i) attribute names
  (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
- The attribute list (i) must be a subset of *grouping-list*. Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

Conceptual Evaluation



- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, *unnecessary* fields are deleted, and 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*. (SQL does not exploit primary key semantics here!)
- 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
```

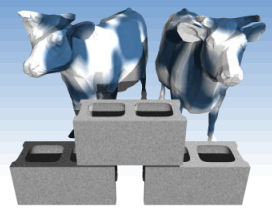
Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

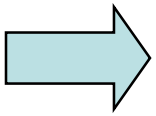
Sailors instance:

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

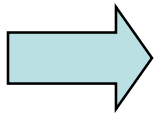
Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors



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

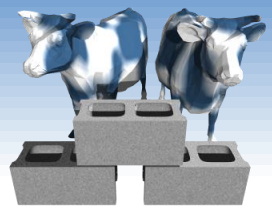


rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0



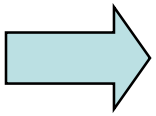
rating	minage
3	25.5
7	35.0
8	25.5

Find age of the youngest sailor with age ≥ 18, for each rating level with at least 2 such sailors, and where every sailor is under 60.

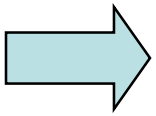


HAVING COUNT (*) > 1 AND MAX(S.age) < 60

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

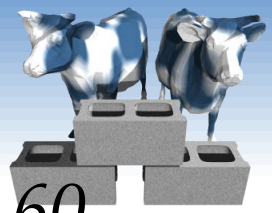


rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0



rating	minage
7	35.0
8	25.5

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
```

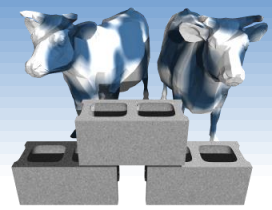
Answer relation:

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

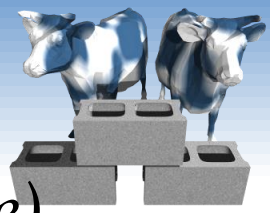
For each red boat, find the number of reservations for this boat



```
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?

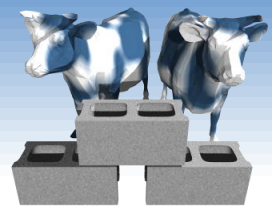
Find age of the youngest sailor with age > 18, for each rating with at least 2 sailors (of any age)



```
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



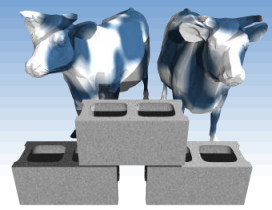
- Aggregate operations cannot be nested! WRONG:

```
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)
```

Summary



- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- Relationally complete; in fact, significantly more expressive power than relational algebra.
- Even queries that can be expressed in RA can often be expressed more naturally in SQL.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
  - In practice, users need to be aware of how queries are optimized and evaluated for best results.