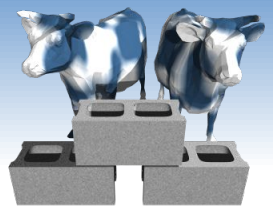
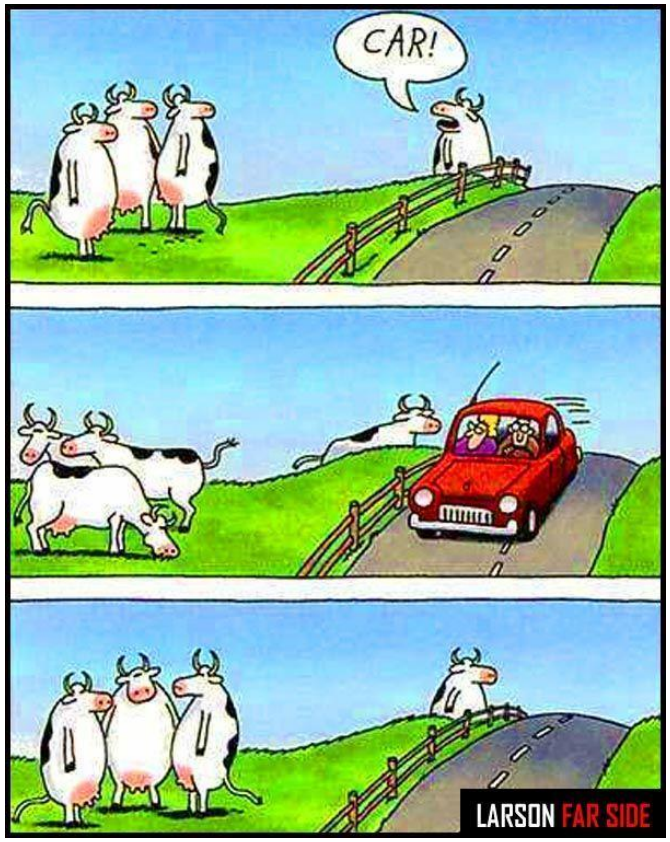
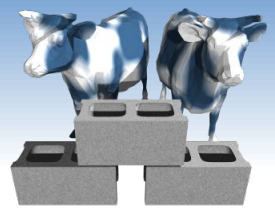


SQL: Basic Queries

Problem Set #1
due date has been
changed to 9/8

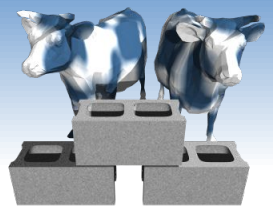


Structured Query Language (SQL)



- Introduced in 1974 by IBM
- "De facto" standard db query language
- Caveats
  - Standard has evolved (major revisions in 1992 and 1999)
  - Semantics, Syntax, and Extentions may vary slightly among DBMS implementations

"Baby" Example Instances



- We will start with these instances of the Sailors and Reserves relations in our examples.
- If the key for the Reserves relation contained only the attributes *sid* and *bid*, how would the semantics differ?

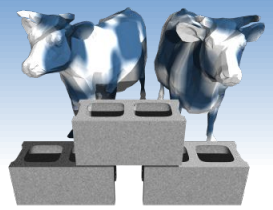
Sailors:

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

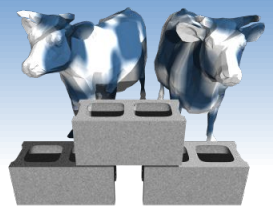
Basic SQL Query



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

- *target-list* A list of attributes of relations in *relation-list*
- *relation-list* A list of relation names (possibly with a *range-variable* after each name).
- *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of <, >, =, <=, >=, <>) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. By default duplicates are *not* eliminated!

Conceptual Evaluation Strategy



- Semantics of an SQL query defined in terms of the following *conceptual evaluation strategy*:
  - Compute the cross-product of the *relation-list*.
  - Select tuples (rows) if they satisfy *qualifications*.
  - Select attributes (columns) in the *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

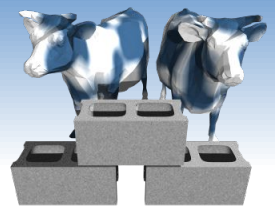
```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

(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

Outputs:

sname
rusty

Table Aliases (Variables)



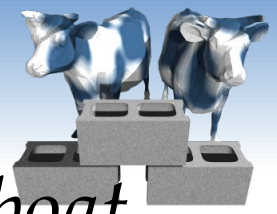
- Really needed only if the same relation appears more than once in the FROM clause. The same query can also be written as:

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

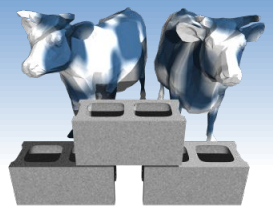
Aliases provide a convenient shorthand for referencing tables



Find sailors who've reserved at least one boat

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

- Why is the DISTINCT keyword useful here?
- What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause?
- Does DISTINCT work as expected in this case?
- Just because a query appears to gives a correct answer on a specific database instance, does not mean that it is correct!

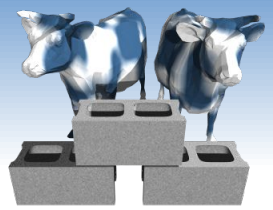


Expressions and Strings

```
SELECT S.age, S.age*12.0 AS ageMonths, 10-S.rating AS revRating
FROM   Sailors S
WHERE  S.sname LIKE '_us%'
```

age	ageMonths	revRating
45.0	540.0	3
35.0	420.0	0

- Illustrates use of arithmetic expressions and string pattern matching: *Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names have 'us' as the second and third letter of their name.*
- AS renames fields in result. (Some SQL implementations allow the use of '*newalias*=*expr*' as well)
- LIKE is used for approximate string matching. "_" stands for any one character and "%" stands for 0 or more arbitrary characters.



A more extensive example

- "Infant" Sailors/Reserves/Boats instance

Sailors:

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

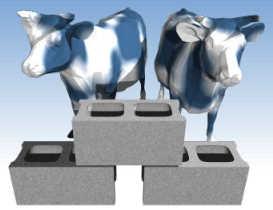
Reserves:

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats:

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

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



- Two approaches
- If we replace OR by AND in the first version, what do we get?
- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?)

```
SELECT DISTINCT S.sname, 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")
```

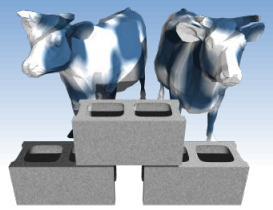
sname	sid
Dustin	22
Lubber	31
Horatio	64
Horatio	74

```
SELECT S.sname, 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.sname, S.sid
FROM   Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
AND    B.color="green"
```

EXCEPT

sname	sid
Horatio	64

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



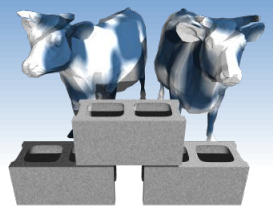
- Solution 1: Multiple instancing of the same relation in the relation-list using another variable
- Solution 2: INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.
- Consider the symmetry of the UNION, EXCEPT, and INTERSECT queries versus the first, multiple instancing version.

```
SELECT DISTINCT S.sname, 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")
```

```
SELECT S.sname, 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.sname, S.sid
FROM   Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
  AND  B.color="green"
```

sname	sid
Dustin	22
Lubber	31

Nested Queries



Find names of sailors who've never reserved boat #103:

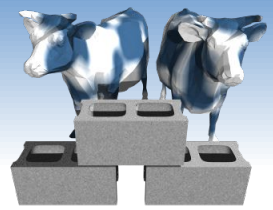
```
SELECT S.sid, S.sname
FROM   Sailors S
WHERE  S.sid NOT IN  (SELECT  DISTINCT R.sid
                      FROM  Reserves R
                      WHERE  R.bid=103)
```

sid
22
31
74

sid	sname
29	Brutus
32	Andy
58	Rusty
64	Horatio
71	Zorba
85	Art
95	Bob

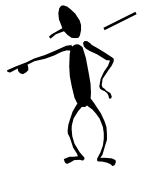
- *A very powerful feature of SQL*: a WHERE clause can itself contain an SQL query!
- To find sailors who've reserved #103, use IN.
- To understand semantics of nested queries, think of a *nested loops* evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*

Nested Queries with Correlation



Find names of sailors who've reserved any boat:

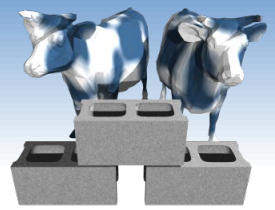
```
SELECT  S.sid, S.sname
FROM    Sailors S
WHERE   EXISTS (SELECT *
                FROM   Reserves R
                WHERE  S.sid=R.sid)
```



Correlation is when an inner SELECT references relation variables of outer SELECT

- EXISTS is another set comparison operator, like IN.
- Illustrates why, in general, a subquery must be re-evaluated for each Sailors tuple.

sid	sname
22	Dustin
31	Lubber
64	Horatio
74	Horatio



More on Set-Comparison Operators

- We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL, *op* IN
- Find sailors whose rating is greater than that of some sailor called Horatio:

Not every SQL dialect supports ANY and ALL. However, min() and max() functions can usually be used to achieve the desired effect

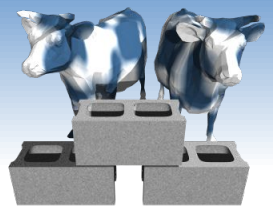
```
SELECT  *
  FROM  Sailors S
 WHERE  S.rating > ANY (SELECT  S2.rating
                          FROM  Sailors S2
                         WHERE  S2.sname='Horatio')
```

rating
7
9

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

sid	sname	rating	age
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0

Rewriting INTERSECT Using "IN"



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

```
SELECT  DISTINCT S.sid, S.sname
  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')
```

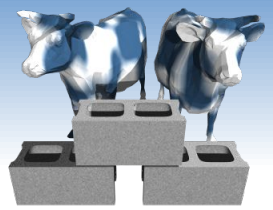
- Similarly, EXCEPT queries re-written using NOT IN.

sid
22
31
74

sid	sname
22	Dustin
31	Lubber

Division in SQL

Find sailors who've reserved all boats.



- The hard way, without EXCEPT:

(1)

```
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)
```

All boats

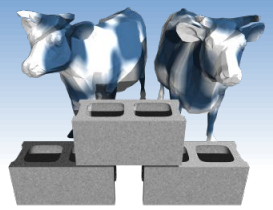
Boats reserved by a given Sailor

(2)

```
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

sname
Dustin

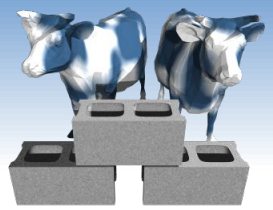


"*Relationally*" *Pure SQL*

Thus far all of the SQL commands I have used (except one) take one or more relations (tables) as an input and produce a new relation as an output.

This has limitations. Sometimes we'd like to compute summaries of our tables such as…

- how many rows were returned
- averages over all outputs



SQL's Aggregate Operators

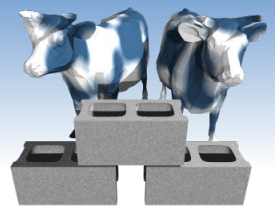
- Significant SQL extension
- Computation and summarization operations
- Appears in *target-list* of query
- Results *aggregate* rather than appear individually
- E.x. How many instances in the sailor relation?

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

single column

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

COUNT (*)
10



More examples

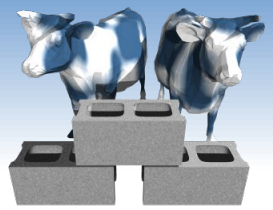
- Average age of Sailors with a rating of 10?

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

- Names of Sailors having the maximum rating

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

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



More examples (cont)

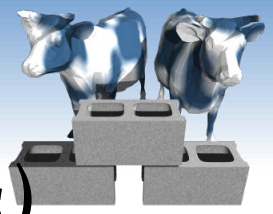
- How many distinct ratings for Sailors less than 40 years of age?

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

- 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
```

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 name and age of the oldest sailor(s)

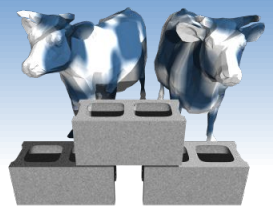
- The first query is incorrect! (Switch the S.age to S.rating to see why)
- The third query is equivalent to the second query, but may not be supported in some systems.

```
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
```

Next Time



- We've covered the portion of SQL that strictly returns "tuples from tables" and "aggregate" table summaries
- Next time we will consider some important extensions, that partition sets of tuples. They are useful and a natural additions to query specification.