More on SQL

Module 7

Prof. Alex Brodsky

Database Systems

Example Instances

R1

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

- We will use 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?

S1

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

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Basic SQL Query

```
SELECT    target-list
FROM      relation-list
WHERE     qualification
```


- *relation-list* A list of relation names (possibly with a *range-variable* after each name). One name can appear more than once, with different range-variable names.
- *target-list* A list of attributes of the relations in *relation-list*

Conceptual Evaluation Strategy

Nested loops evaluation:

```
SELECT    target-attribute-list
FROM      R1, ..., Rn
WHERE     qualification
```


Foreach tuple t1 in R1

…

Foreach tuple tn in Rn

1. Substitute the attribute names in the qualification part with values from t1, …, tn
2. If the modified qualification part evaluates True
   then output target-attribute-***values***
   else do nothing

end

…

end

Set Operations (back to set semantics)

- Set-ops:
  - All duplicates removed!
  - Union
  - Intersect
  - Except (difference)
- *Bag version*
  - Union all

Select …
From …
Where …

Set-op

Select …
From …
Where …

Qualification involving Sets

- Value **IN** *SET*
  - Value NOT **IN** *SET*
- **EXISTS** *SET*
  - NOT **EXISTS** *SET*
- **UNIQUE** *SET*
- Value θ ANY *SET*
- Value θ ALL *SET*
  - θ is one of =, >, >=, <, <=, <>
- Where do we get *SET* from?
  - *Select statement!*
  - *Nested queries.*

Nested Queries

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)
```


- A very powerful feature of SQL: a WHERE clause can itself contain an SQL query! (Actually, so can FROM and HAVING clauses.)
- To find sailors who've *not* reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a *nested loops* evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*

Conclusions for now

- This is "core" part of SQL
- Bag, nested loop
- Similar to Relational Algebra, but not quite
- A lot of extensions coming up!

Post Processing

- Processing on the result of an SQL query:
  - Sorting: can sort the tuples in the output by any column (even the ones not appearing the the SELECT clause)
  - Duplicate removal
  - Example:


```
SELECT Distinct S.sname
FROM  Sailors S, Reserves R
WHERE  S.sid=R.sid and R.bid=103
Sort by S.sid asc, S.sname desc;
```

- ***Aggregation operators***

Aggregate Operators

- Significant extension of relational algebra.

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

single column


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

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

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

```
SELECT  COUNT (DISTINCT S.rating)
FROM  Sailors S
WHERE S.sname='Bob'
```

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

Find name and age of the oldest sailor(s)

- The first query is illegal! (We'll look into the reason a bit later, when we discuss GROUP BY.)
- 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
```

GROUP BY and HAVING

- So far, we've applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several *groups* of tuples.
- 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 the age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors

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


sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

- Only S.rating and S.age are mentioned in the SELECT, GROUP BY or HAVING clauses; other attributes `*unnecessary*'.
- 2nd column of result is unnamed. (Use AS to name it.)

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

rating	
7	35.0

Answer relation

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 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  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)
```

Continue from previous

However, this should work on Oracle 8:

```
SELECT  S.rating
FROM  Sailors S
Group by S.rating
Having AVG(S.age) = (SELECT MIN (AVG (S2.age))
                     FROM Sailors S2
                     Group by rating);
```

Conclusion

- Post processing on the result of queries is supported.
- Aggregation is the most complex "post processing"
  - "Group by" clause partition the results into groups
  - "Having" clause puts condition on groups (just like Where clause on tuples).