

SQL Basics

Lecture 6

EECS 339

Example Instances

R1

<u>sid</u>	<u>bid</u>	<u>day</u>
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

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

S2

<u>sid</u>	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	[DISTINCT] <i>target-list</i>
FROM	<i>relation-list</i>
WHERE	<i>qualification</i>

- *relation-list* A list of relation names (possibly with a *range-variable* after each name).
- *target-list* A list of attributes of relations in *relation-list*
- *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of $<$, $>$, $=$, \leq , \geq , \neq) combined using AND, OR and NOT.
- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates. Default is that 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 *relation-list*.
 - Discard resulting tuples if they fail *qualifications*.
 - Delete attributes that are not in *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

A Note on Aliases

- Really needed only if the same relation appears twice in the FROM clause. The previous 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
```

*It is good style,
however, to use
range variables
always!*

Find sailors who've reserved at least one boat

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

- Would adding DISTINCT to this query make a difference?
- What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause? Would adding DISTINCT to this variant of the query make a difference?

Nested Queries

- Queries can contain multiple select blocks and name each one
 - Called a common table expression, denoted using `WITH`
- Nested queries also allowed using parentheses in `WHERE` clause

Find names of sailors who've reserved boat #103:

```
WITH reserved_103 AS (SELECT R.sid
                        FROM Reserves R
                        WHERE R.bid=103)
SELECT S.sname
FROM Sailors S, reserved_103 R
WHERE S.sid = r.sid;
```

OR

```
SELECT S.sname
FROM Sailors S, reserved_103 R
WHERE S.sid = (SELECT R.sid
                FROM Reserves R
                WHERE R.bid=103)
```


Study Break

- Given the relations from our hospital example:

hospital(hosp_id, h_name, h_addr)

doctor(doctor_id, d_name, d_specialty, hosp_id)

patient(patient_id, p_name, p_dob, doctor_id)

- Express these queries using SQL:
 - Find the patients who are seeing the doctor with id 5
 - Find the patient ids for individuals that are being treated by doctors specializing in pediatrics.
 - Find the patient ids who see both a pediatrician and a surgeon

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

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

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

```
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
```

Motivation for Grouping

- 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 $i = 1, 2, \dots, 10$:

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

Queries With GROUP BY and HAVING

SELECT	[DISTINCT] <i>target-list</i>
FROM	<i>relation-list</i>
WHERE	<i>qualification</i>
GROUP BY	<i>grouping-list</i>
HAVING	<i>group-qualification</i>

- 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:

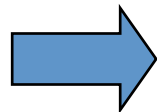
rating	minage
3	25.5
7	35.0
8	25.5

Sailors instance:

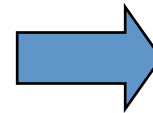
<u>sid</u>	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

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

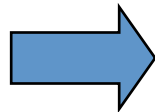


rating	minage
3	25.5
7	35.0
8	25.5

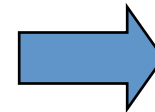
Find age of the youngest sailor with age \geq 18, for each rating with at least 2 such sailors and with every sailor under 60.

HAVING COUNT (*) > 1 AND EVERY (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

What is the result of changing EVERY to ANY?

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:

<u>sid</u>	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 scout
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)
```

Study Break

- Given the relations from our hospital example:

hospital(hosp_id, h_name, h_addr)

doctor(doctor_id, d_name, d_specialty, hosp_id)

patient(patient_id, p_name, p_dob, doctor_id)

- Express these queries using SQL:
 - Calculate the number of patients associated with each doctor
 - Find the average age of a patient seeing a surgeon
 - Find the patient_ids who have been seen in all hospitals

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.