Basic SQL

Module 6

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*

Basic SQL Query

- *qualification* Comparisons ("Attr *op* const" or "Attr1 *op* Attr2", where *op* is one of =, >, >=, <, <=, <>) combined using AND, OR and NOT.

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*.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

Set vs. Bag (Multiset)

- Set: {1, 2, 3}
  - No duplicates, no order
- Bag: {1, 2, 2, 2, 3, 3}
  - Duplicate possible, no order
- Membership test
  - Same
- SQL uses "Bag Semantics"
- Relational algebra uses "Set Semantics"

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

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 Range Variables

- 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
```


- Is it different from algebra query below?

$$\pi_{\text{Sid}}(\text{Sailor} \rhd\lhd \text{Reserve})$$

- How many times the same sid may appear?
- What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause?

Expressions and Strings

```
SELECT  S.age, age1=S.age-5, 2*S.age AS age2
FROM  Sailors S
WHERE  S.sname LIKE 'B_%B'
```


- Illustrates use of arithmetic expressions and string pattern matching: *for each pair (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.*
- AS and = are two ways to name fields in result.
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.

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 …

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

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

```
SELECT  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')
```

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

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

- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.
- Included in the SQL/92 standard, but some systems don't support it.
- Contrast symmetry of the UNION and INTERSECT queries with how much the other versions differ.

```
SELECT  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.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'
```

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.*

Two "equivalent" queries?

```
Select S.Sid
From Sailor S, Reserve R
Where S.Sid = R.Sid;
```



```
Select Sid
From Sailor
Where Sid in (Select Sid
              From Reserve);
```

Nested Queries with Correlation

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

```
SELECT  S.sname
FROM  Sailors S
WHERE  EXISTS  (SELECT  *
                FROM  Reserves R
                WHERE  R.bid=103 AND S.sid=R.sid)
```


- EXISTS is another set comparison operator, like IN.
- If UNIQUE is used, and * is replaced by *R.bid*, finds sailors with exactly one reservation for boat #103. (UNIQUE checks for duplicate tuples; * denotes all attributes. Why do we have to replace * by *R.bid*?)
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple.

More on Set-Comparison Operators

- Find sailors whose rating is greater than that of some sailor called Horatio:


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

More

- Find sailors whose rating is greater than that of every other sailor.


```
SELECT  *
FROM  Sailors S1
WHERE  S1.rating > All  (SELECT  S2.rating
                         FROM  Sailors S2
                         WHERE S2.sid <> S1.Sid);
```

Rewriting INTERSECT Queries Using IN

Find sid's of sailors who've reserved both 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'
      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.
- To find *names* (not *sid*'s) of Sailors who've reserved both red and green boats, just replace *S.sid* by *S.sname* in SELECT clause. (What about INTERSECT query?)

Division in SQL

Find sailors who've reserved all boats.

(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))
```

- Let's do it the hard way, without EXCEPT:

(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

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!