Advanced SQL

EECS 339

Lecture 7

The story so far

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

- Select-project- (simple) join
- Aggregation

Example Instances

- We will use these instances of the Sailors and Reserves relations in our examples.

R1

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

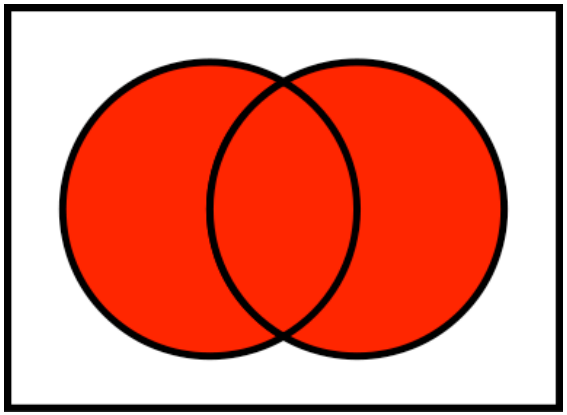
S1

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

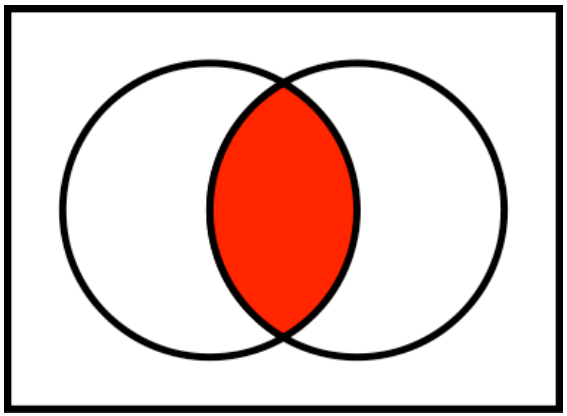
S2

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

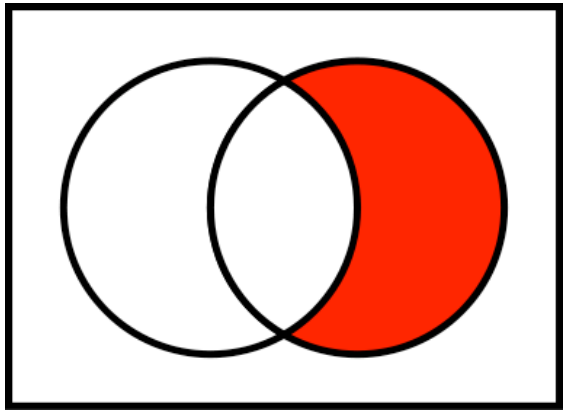
Set Comparisons



UNION



INTERSECT



EXCEPT

Image source: wikipedia.org

Find sid's of sailors who've reserved a red <u>or</u> 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 <u>and</u> a green boat

- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.

- 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'
```

Key field!

```
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'
```

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.*

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 at most 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

- We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS, UNION ALL and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL, *op* IN
- 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' )
```

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.

Division in SQL

(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))
```

Find sailors who've reserved all boats.

- 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

Top-K Querying

- Look at the first k results from a query
- Use LIMIT clause
- Example: Select the top 10 sailors with the largest number of reservations

```
SELECT sid
FROM sailors s, reservations r
WHERE s.sid = r.rid
GROUP BY sid
ORDER BY COUNT(*)
LIMIT 10;
```

Study Break: Advanced SQL

- Consider the following schema:
  - Suppliers(<u>sid</u>, sname, address)
  - Parts(<u>pid</u>, color)
  - Catalog(<u>sid</u>, <u>pid</u>, cost)
- Express the following as SQL queries:
  - Find the sids of suppliers who supply a red part and a green part.
  - For each part, find the sname of the supplier who charges the most for that part.
  - For every supplier that supplies a green part and a red part, print the name and price of the most expensive part that she supplies

Summary

- SQL permits us to reason about data a tuple at a time
- We also work with data using set comparisons
- Combine the two to express a rich set of query semantics