SQL II

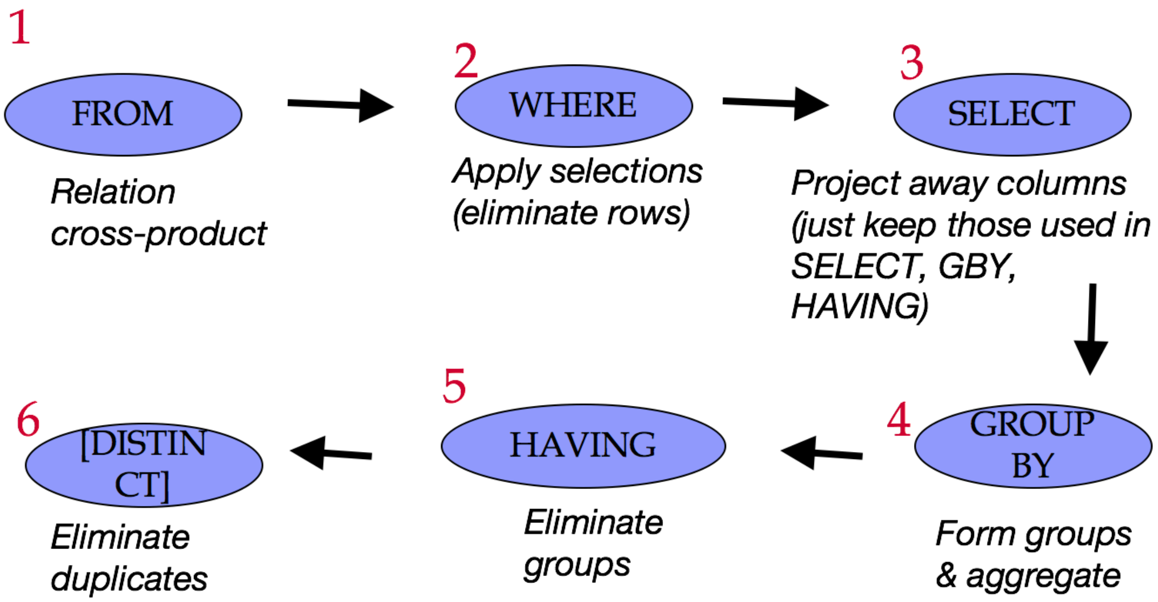
R & G - Chapter 5

Berkeley
cs186

SQL DML 1:
Basic Single-Table Queries

- **SELECT** [**DISTINCT**] *<column expression list>*
  FROM *<single table>*
  [**WHERE** *<predicate>*]
  [**GROUP BY** *<column list>*
  [**HAVING** *<predicate>*]]
  [**ORDER BY** *<column list>*]
  [**LIMIT** <integer>];

Conceptual SQL Evaluation



1 FROM

Relation cross-product

2 WHERE

Apply selections (eliminate rows)

3 SELECT

Project away columns (just keep those used in SELECT, GBY, HAVING)

4 GROUP BY

Form groups & aggregate

5 HAVING

Eliminate groups

6 [DISTINCT]

Eliminate duplicates

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

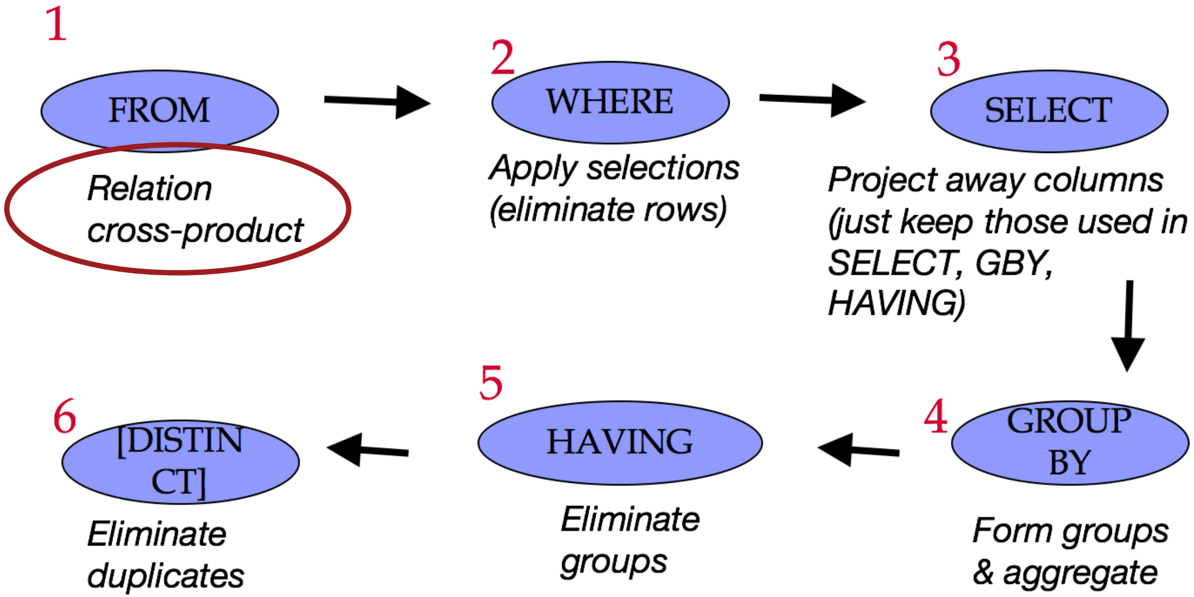
Putting it all together

```
SELECT S.dept, AVG(S.gpa), COUNT(*)
FROM Students S
WHERE S.gender = 'F'
GROUP BY S.dept
HAVING COUNT(*) >= 2
ORDER BY S.dept;
```

Join Queries

- SELECT [DISTINCT] <*column expression list*>
  FROM <*table1 [AS t1], ... , tableN [AS tn]*>
  [WHERE <*predicate*>]
  [GROUP BY <*column list*>[HAVING <*predicate*>]]
  [ORDER BY <*column list*>];

Conceptual SQL Evaluation, cont



1 FROM

Relation cross-product

2 WHERE

Apply selections (eliminate rows)

3 SELECT

Project away columns (just keep those used in SELECT, GBY, HAVING)

4 GROUP BY

Form groups & aggregate

5 HAVING

Eliminate groups

6 [DISTINCT]

Eliminate duplicates

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

Cross (Cartesian) Product

- All pairs of tuples, concatenated

Sailors

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sid	sname	rating	age	sid	bid	day
1	Popeye	10	22	1	102	9/12
1	Popeye	10	22	2	102	9/13
1	Popeye	10	22	1	101	10/01
2	OliveOyl	11	39	1	102	9/12
…	…	…	…	…	…	…

Find sailors who've reserved a boat

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

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sid	sname	rating	age	sid	bid	day
1	Popeye	10	22	1	102	9/12
1	Popeye	10	22	2	102	9/13
1	Popeye	10	22	1	101	10/01
~~2~~	~~OliveOyl~~	~~11~~	~~39~~	~~1~~	~~102~~	~~9/12~~
…	…	…	…	…	…	…

Slide Deck Title

Find sailors who’ve reserved a boat cont

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

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sid	sname	bid
1	Popeye	102
1	Popeye	101
2	OliveOyl	102

Column Names and Table Aliases

```
SELECT Sailors.sid, sname, bid
FROM Sailors, Reserves
WHERE Sailors.sid = Reserves.sid
```

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

More Aliases

```
SELECT x.sname, x.age,
             y.sname AS sname2,
             y.age AS age2
FROM Sailors AS x, Sailors AS y
WHERE x.age > y.age
```

sname	**age**	**sname2**	**age2**
Popeye	22	Bob	19
OliveOyl	39	Popeye	22
OliveOyl	39	Garfield	27
OliveOyl	39	Bob	19
Garfield	27	Popeye	22
Garfield	27	Bob	19

- Table aliases in the FROM clause
  - Needed when the same table used multiple times (“self-join”)
- Column aliases in the SELECT clause

Arithmetic Expressions

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
    FROM   Sailors AS S
    WHERE  S.sname = 'Popeye’
```

```
SELECT S1.sname AS name1, S2.sname AS name2
    FROM   Sailors AS S1, Sailors AS S2
    WHERE  2*S1.rating = S2.rating - 1
```

SQL Calculator!

```
SELECT
        log(1000) as three,
        exp(ln(2)) as two,
        cos(0) as one,
        ln(2*3) = ln(2) + ln(3) as sanity;
```

String Comparisons

- Old School SQL

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sname LIKE 'B_%’
```

- Standard Regular Expressions

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sname ~ 'B.*’
```

Combining Predicates

- Subtle connections between:
  - Boolean logic in WHERE (i.e., AND, OR)
  - Traditional Set operations (i.e. INTERSECT, UNION)
- Let’s see some examples…

Sid’s of sailors who reserved a red **OR** a green boat

```
SELECT R.sid
FROM   Boats B, Reserves R
WHERE  R.bid=B.bid AND
          (B.color='red' OR B.color='green')
```

Sid’s of sailors who reserved a red **OR** a green boat Pt 2

```
SELECT R.sid
FROM Boats B,Reserves R
WHERE R.bid=B.bid AND
                (B.color='red’ OR B.color='green')
```

VS…

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

UNION ALL

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

Sid’s of sailors who reserved a red **OR** a green boat Pt 3

```
SELECT R.sid
FROM Boats B,Reserves R
WHERE R.bid=B.bid AND
            (B.color='red’ AND B.color='green')
```

VS…

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

INTERSECT

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

Slide Deck Title

Find sailors who have **not** reserved a boat

```
SELECT S.sid
FROM   Sailors S

EXCEPT

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

Set Semantics

- Set: a collection of distinct elements
- Standard ways of manipulating/combining sets
  - Union
  - Intersect
  - Except
- Treat tuples within a relation as elements of a set

Default: Set Semantics

R = {A, A, A, A, B, B, C, D}
S = {A, A, B, B, B, C, E}

- UNION

  {A, B, C, D, E}
- INTERSECT

  {A, B, C}
- EXCEPT

  {D}

Note: Think of each letter as being a **tuple** in **relation.**

ex:
A: (Jim, 18, English, 4.0)
B: (Marcela , 20, CS, 3.8)
C: (Gail, 19, Statistics, 3.74)
D: (Goddard, 20, Math, 3.8

“ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

“UNION ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

- UNION ALL: sum of cardinalities
  {A(4+2), B(2+3), C(1+1), D(1+0), E(0+1)}
  = {A, A, A, A, A, A, B, B, B, B, B, C, C, D, E}

“INTERSECT ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

- INTERSECT ALL: min of cardinalities

  {A(min(4,2)), B(min(2,3)), C(min(1,1)),
  D(min(1,0)), E(min(0,1))}
  = {A, A, B, B, C}

“EXCEPT ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

- EXCEPT ALL: difference of cardinalities

  {A(4-2), B(2-3), C(1-1), D(1-0), E(0-1)}
  = {A, A, D, }

Nested Queries: IN

- *Names of sailors who've reserved boat #102:*

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN
      (SELECT  R.sid
       FROM    Reserves R
       WHERE   R.bid=102)
```

subquery

Nested Queries: NOT IN

- *Names of sailors who’ve **<u>not</u>** reserved boat #103:*

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

Nested Queries: EXISTS

- *This is a bit odd, but it is legal:*

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

Nested Queries with Correlation

- *Names of sailors who’ve reserved boat #102:*

```
SELECT  S.sname
FROM    Sailors S
WHERE EXISTS
        (SELECT  *
        FROM  Reserves R
        WHERE R.bid=102 AND S.sid=R.sid)
```

- Correlated subquery is recomputed for each Sailors tuple.

More on Set-Comparison Operators

- We’ve seen: IN, EXISTS
- Can also have: NOT IN, NOT EXISTS
- Other forms: op ANY, op ALL

Find sailors whose rating is greater than that of *some* sailor called Popeye:

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

A Tough One: “Division”

- Relational Division: “Find sailors who’ve reserved all boats.”
  Said differently: “sailors with no counterexample missing boats”

```
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 ))
```

ARGMAX? Pt 1

- The sailor with the highest rating
- Correct or Incorrect?

```
SELECT MAX(S.rating)
FROM Sailors S;
```

VS

```
SELECT S.*, MAX(S.rating)
FROM Sailors S;
```

ARGMAX? Pt 2

- The sailor with the highest rating
- Correct or Incorrect? Same or different?

```
SELECT *
FROM   Sailors S
WHERE  S.rating >= ALL
   (SELECT  S2.rating
   FROM  Sailors S2)
```

VS

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

ARGMAX? Pt 3

- The sailor with the highest rating
- Correct or Incorrect? Same or different?

```
SELECT *
FROM   Sailors S
WHERE  S.rating >= ALL
    (SELECT  S2.rating
    FROM  Sailors S2)
```

VS

```
SELECT *
FROM   Sailors S
ORDER BY rating DESC
LIMIT 1;
```

“Inner” Joins: Another Syntax

```
SELECT s.*, r.bid
FROM Sailors s, Reserves r
WHERE s.sid = r.sid
AND ...
```

```
SELECT s.*, r.bid
FROM Sailors s INNER JOIN Reserves r
ON s.sid = r.sid
WHERE ...
```

Join Variants

```
SELECT <column expression list>
FROM table_name
   [INNER | NATURAL
    | {LEFT |RIGHT | FULL } {OUTER}] JOIN table_name
   ON <qualification_list>
WHERE …
```

- INNER is default
- Inner join what we've learned so far
  - Same thing, just with different syntax.

Inner/Natural Joins

```
SELECT s.sid, s.sname, r.bid
FROM Sailors s, Reserves r
WHERE s.sid = r.sid
  AND s.age > 20;
```

SELECT s.sid, s.sname, r.bid
FROM Sailors s **INNER JOIN** Reserves r
ON s.sid = r.sid
WHERE s.age > 20;

SELECT s.sid, s.sname, r.bid
FROM Sailors s **NATURAL JOIN** Reserves r
WHERE s.age > 20;

- **ALL 3 ARE EQUIVALENT!**
- “NATURAL” means equi-join for pairs of attributes with the same name

Left Outer Join

- Returns all matched rows, and *preserves* all unmatched rows from the table on the left of the join clause
  - (use nulls in fields of non-matching tuples)

```
SELECT s.sid, s.sname, r.bid
FROM Sailors2 s LEFT OUTER JOIN Reserves2 r
ON s.sid = r.sid;
```

Returns all sailors & bid for boat in any of their reservations

Note: no match for s.sid? r.bid IS NULL!

Right Outer Join

- Returns all matched rows, and *preserves* all unmatched rows from the table on the right of the join clause
  - (use nulls in fields of non-matching tuples)

```
SELECT r.sid, b.bid, b.bname
FROM Reserves2 r RIGHT OUTER JOIN Boats2 b
ON r.bid = b.bid
```

Returns all boats and sid for any sailor associated with the reservation.

Note: no match for b.bid? r.sid IS NULL!

Full Outer Join

- Returns all (matched or unmatched) rows from the tables on both sides of the join clause

```
SELECT r.sid, b.bid, b.bname
FROM Reserves2 r FULL OUTER JOIN Boats2 b
ON r.bid = b.bid
```

- Returns all boats & all information on reservations
- No match for r.bid?
  - b.bid IS NULL AND b.bname IS NULL!
- No match for b.bid?
  - r.sid IS NULL!

Views: Named Queries

CREATE VIEW *view_name*
AS *select_statement*

- Makes development simpler
- Often used for security
- Not “materialized”

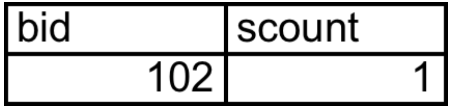
```
CREATE VIEW Redcount

AS SELECT B.bid, COUNT(*) AS scount
        FROM Boats2 B, Reserves2 R
        WHERE R.bid=B.bid AND B.color='red'
        GROUP BY B.bid
```

Views Instead of Relations in Queries

```
CREATE VIEW Redcount
AS SELECT B.bid, COUNT(*) AS scount
        FROM Boats2 B, Reserves2 R
        WHERE R.bid=B.bid AND B.color='red'
        GROUP BY B.bid;
```

```
SELECT * from redcount;
```



bid	scount
102	1

```
SELECT bname, scount
FROM Redcount R, Boats2 B
WHERE R.bid=B.bid
AND scount < 10;
```

Subqueries in FROM

Like a “view on the fly”!

```
SELECT  bname, scount
FROM Boats2 B,
(SELECT B.bid, COUNT (*)
        FROM Boats2 B, Reserves2 R
        WHERE R.bid = B.bid AND B.color = 'red'
        GROUP BY B.bid) AS Reds(bid, scount)

    WHERE  Reds.bid=B.bid
            AND scount < 10
```

WITH a.k.a. common table expression (CTE)

Another “view on the fly” syntax:

```
        WITH Reds(bid, scount) AS
        (SELECT B.bid, COUNT (*)
        FROM Boats2 B, Reserves2 R
        WHERE R.bid = B.bid AND B.color = 'red'
        GROUP BY B.bid)

SELECT bname, scount
FROM Boats2 B, Reds
WHERE Reds.bid=B.bid
AND scount < 10
```

Can have many queries in WITH

Another “view on the fly” syntax:

```
        WITH Reds(bid, scount) AS
        (SELECT B.bid, COUNT (*)
        FROM Boats2 B, Reserves2 R
        WHERE R.bid = B.bid AND B.color = 'red'
        GROUP BY B.bid),

        UnpopularReds AS
        SELECT bname, scount
        FROM Boats2 B, Reds
        WHERE Reds.bid=B.bid
        AND scount < 10

SELECT * FROM UnpopularReds;
```

ARGMAX GROUP BY?

- The sailor with the highest rating per age

```
WITH maxratings(age, maxrating) AS
(SELECT age, max(rating)
FROM Sailors
GROUP BY age)
```

```
SELECT S.*
  FROM Sailors S, maxratings m
 WHERE S.age = m.age
   AND S.rating = m.maxrating;
```

Brief Detour: Null Values

- Field values are sometimes unknown
  - SQL provides a special value NULL for such situations.
  - Every data type can be NULL
- The presence of null complicates many issues. E.g.:
  - Selection predicates (WHERE)
  - Aggregation
- But NULLs comes naturally from Outer joins

NULL in the WHERE clause

- Consider a tuple where rating IS NULL.

```
 INSERT INTO sailors VALUES
   (11, 'Jack Sparrow', NULL, 35);
```

```
SELECT * FROM sailors
WHERE rating > 8;
```

Is Jack Sparrow in the output?

NULL in comparators

- Rule: (x op NULL) evaluates to … NULL!

```
SELECT 100 = NULL;

SELECT 100 < NULL;

SELECT 100 >= NULL;
```

Explicit NULL Checks

```
SELECT * FROM sailors WHERE rating IS NULL;

SELECT * FROM sailors WHERE rating IS NOT NULL;
```

NULL at top of WHERE

- Rule: Do not output a tuple WHERE NULL

```
SELECT * FROM sailors;

SELECT * FROM sailors WHERE rating > 8;

SELECT * FROM sailors WHERE rating <= 8;
```

NULL in Boolean Logic

Three-valued logic:

NOT	T	F	N
	F	T	

AND	T	F	N
T	T	F	
F	F	F	
N			

OR	T	F	N
T	T	T	
F	T	F	
N			

SELECT * FROM sailors WHERE rating > 8 AND TRUE;

SELECT * FROM sailors WHERE rating > 8 OR TRUE;

SELECT * FROM sailors WHERE NOT (rating > 8);

General rule: NULL **column values are ignored by aggregate functions**

NULL in Boolean Logic

Three-valued logic:

NOT	T	F	N
	F	T	N

AND	T	F	N
T	T	F	N
F	F	F	F
N	N	F	N

OR	T	F	N
T	T	T	T
F	T	F	N
N	T	N	N

SELECT * FROM sailors WHERE rating > 8 AND TRUE;

SELECT * FROM sailors WHERE rating > 8 OR TRUE;

SELECT * FROM sailors WHERE NOT (rating > 8);

General rule: NULL **column values are ignored by aggregate functions**

NULL and Aggregation

```
SELECT count(*) FROM sailors;

SELECT count(rating) FROM sailors;

SELECT sum(rating) FROM sailors;

SELECT avg(rating) FROM sailors;
```

General rule: NULL **column values are ignored by aggregate functions**

NULLs: Summary

- NULL op NULL is NULL
- WHERE NULL: do not send to output
- Boolean connectives: 3-valued logic
- Aggregates ignore NULL-valued inputs

Testing SQL Queries

- SQL Fiddle pages we provide in this class will typically help you answer the questions in the worksheets and vitamins.
- But in real life:
  - not every database instance will reveal every bug in your query.
    - Eg: database instance without any rows in it!
  - Need to debug your queries
  - reasoning about them carefully
  - constructing test data.

Tips for Generating Test Data

- Generate **random data**
  - e.g. using a service like mockaroo.com
- Try to construct data that could check for the following potential errors:
  - Incorrect output schema
  - Output may be missing rows from the correct answer (false negatives)
  - Output may contain incorrect rows (false positives)
  - Output may have the wrong number of duplicates.
  - Output may not be ordered properly.

Summary

- You’ve now seen SQL—you are armed.
- A declarative language
  - Somebody has to translate to algorithms though…
  - The RDBMS implementor ... i.e. you!

Summary Cont

- The data structures and algorithms that make SQL possible also power:
  - NoSQL, data mining, scalable ML, network routing…
  - A toolbox for scalable computing!
  - That fun begins next week
- We skirted questions of good database (schema) design
  - a topic we’ll consider in greater depth later