Administrivia

HW1 Part 2 grades: Thursday class

AA1 grades: Monday

HW2 out: Thursday

Hacker News new | threads | comments | ask | show | jobs | submit

▲ SQL: One of the most valuable skills (craigkerstiens.com)

816 points by duck 6 days ago | hide | past | web | favorite | 378 comments

▲ slap_shot 6 days ago [-]

SQL is one the most amazing concepts I've ever experienced. It's nearly 5 decades old and there is no sign of a replacement. We've created countless other technologies to store and process data, and we always seem to try to re-create SQL in those technologies (e.g. Hive, Presto, KSQL, etc).

I run a (SQL, and Craigs p

SQL: One of the Most Valuable Skills

I've learned a lot of skills over the course of my career, but no technical skill more useful than SQL. SQL stands out to me as the most valuable skill for a few reasons:

1. It is valuable across different roles and disciplines
2. Learning it once doesn't really require re-learning
3. You seem like a superhero. *You seem extra powerful when you know it because of the amount of people that aren't fluent*

https://news.ycombinator.com/item?id=19149792

Didn’t Lecture 3 Go Over SQL?

haha

Didn't Lecture 3 Go Over SQL?

Two sublanguages

DDL Data Definition Language

define and modify schema (physical, logical, view)

CREATE TABLE, Integrity Constraints

DML Data Manipulation Language

get and modify data

simple SELECT, INSERT, DELETE

human-readable language

Gritty Details

DDL

NULL, Views

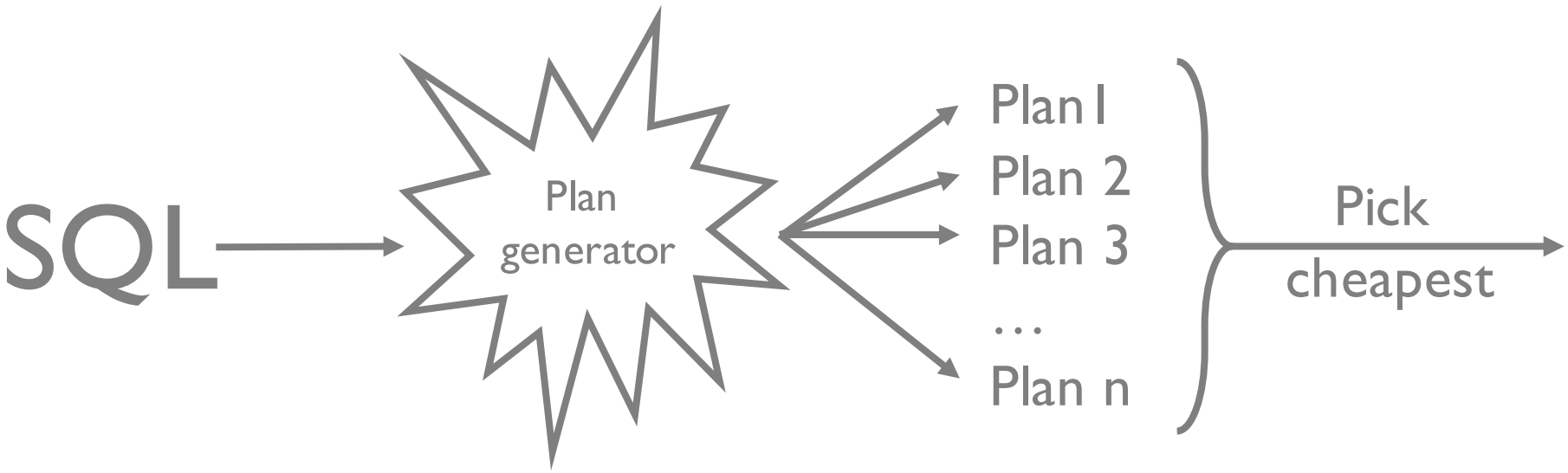
DML

Basics, SQL Clauses, Expressions, Joins, Nested Queries, Aggregation, With, Triggers

Why a declarative language (SQL)?

DBMS makes it run efficiently

- Key: precise query semantics
- Reorder/modify queries while answers stay same
- DBMS estimates costs for different evaluation plans



SQL Extends Relational Algebra

More expressive power than Rel Alg

- Multisets (bags) rather than sets (allow duplicates)
- Ordering
- NULLs
- Aggregates

Most widely used *query language*,
not just relational query language

Today's Database

Sailors

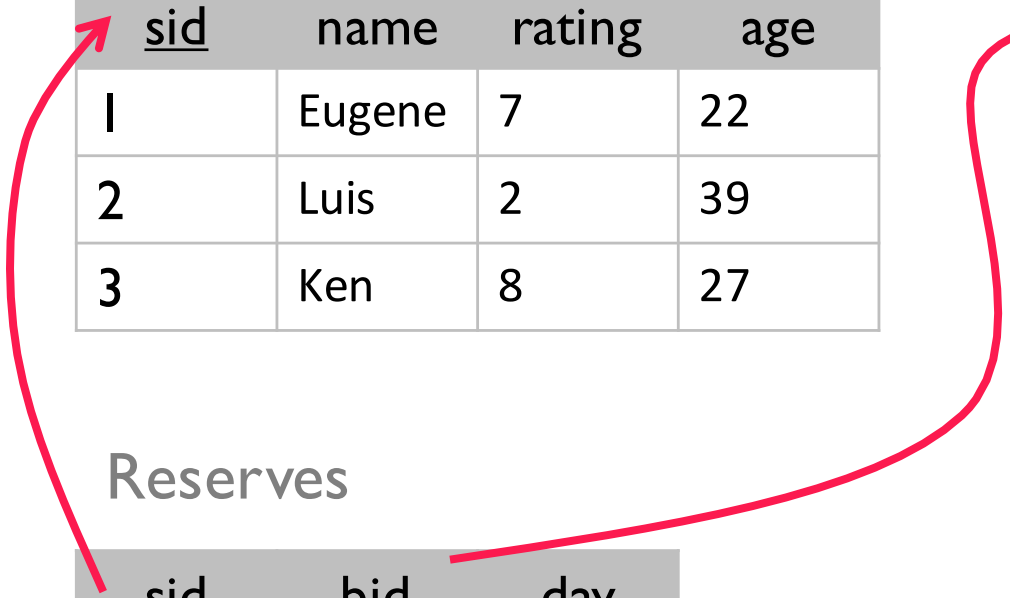
sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Boats

bid	name	color
101	Legacy	red
102	Melon	blue
103	Mars	red

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
2	103	9/14



Is Reserves table correct?

Today's Database

Sailors

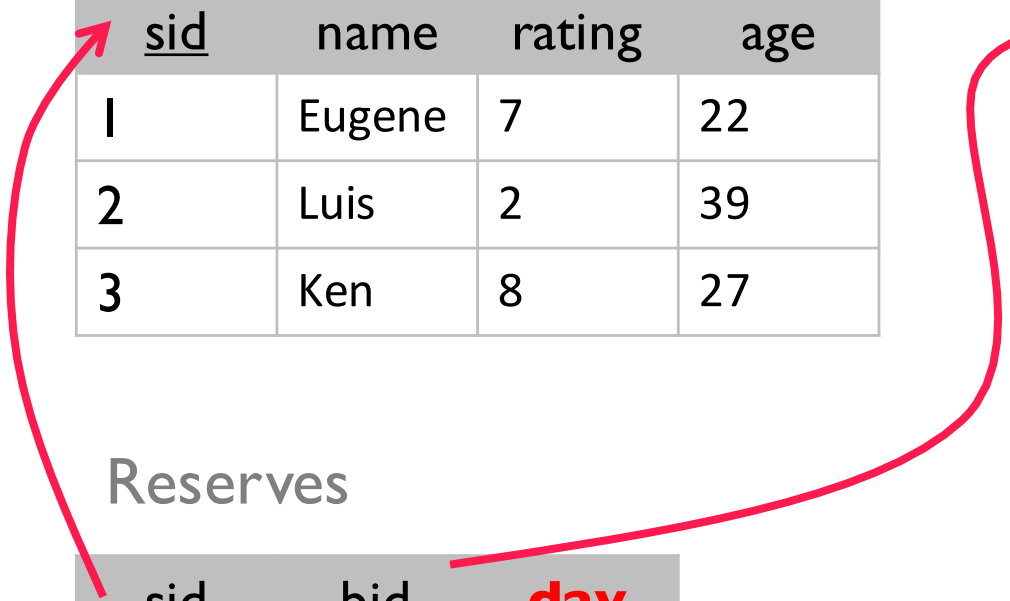
sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Boats

bid	name	color
101	Legacy	red
102	Melon	blue
103	Mars	red

Reserves

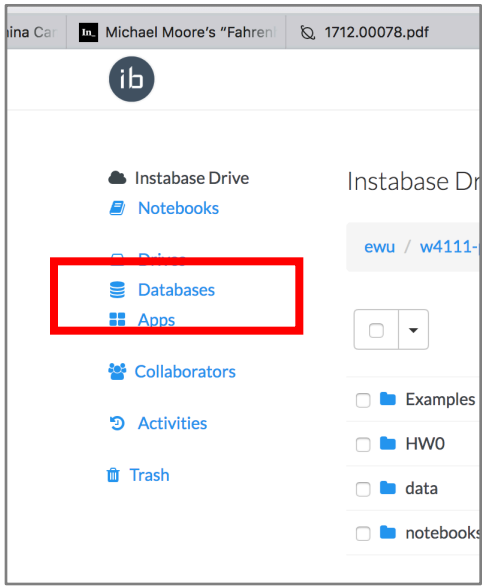
sid	bid	day
1	102	9/12
2	102	9/13
2	103	9/14

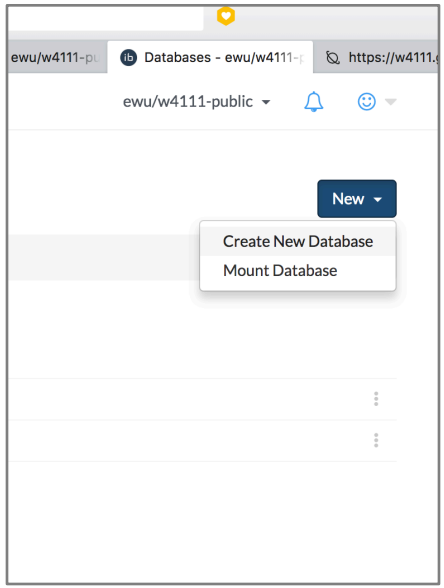


Is Reserves table correct?
Day should be part of key

Follow along at home!

https://www.instabase.com/ewu/w4111-public/fs/Instabase%20Drive/Examples/sql.ipynb





Pick postgres

```
ib.connect_db('ib://ewu/w4111-public/databases/w4111')
```

instabase user

Repo

DBName

<30 year old sailors

```
SELECT *
FROM Sailors
WHERE age < 30
```

sid	name	rating	age
1	Eugene	7	22
3	Ken	8	27

```
SELECT name, age
FROM Sailors
WHERE age < 30
```

name	age
Eugene	22
Ken	27

<30 year old sailors

```
SELECT *
FROM Sailors
WHERE age < 30
```

$\sigma_{age<30}$ (Sailors)

```
SELECT name, age
FROM Sailors
WHERE age < 30
```

$\pi_{name, age}$ ($\sigma_{age<30}$ (Sailors))

Multiple Relations

```
SELECT S.name
FROM   Sailors AS S, Reserves AS R
WHERE  S.sid = R.sid AND R.bid = 102
```

$$\pi_{name}(\sigma_{bid=2}(Sailors \bowtie_{sid} Reserves))$$

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
2	103	9/14

Structure of a SQL Query

DISTINCT
Optional, answer should not have duplicates
Default: duplicates not removed (multiset)

target-list
List of expressions over attrs of tables in relation-list

```
SELECT   [DISTINCT] target-list
FROM     relation-list
WHERE    qualification
```

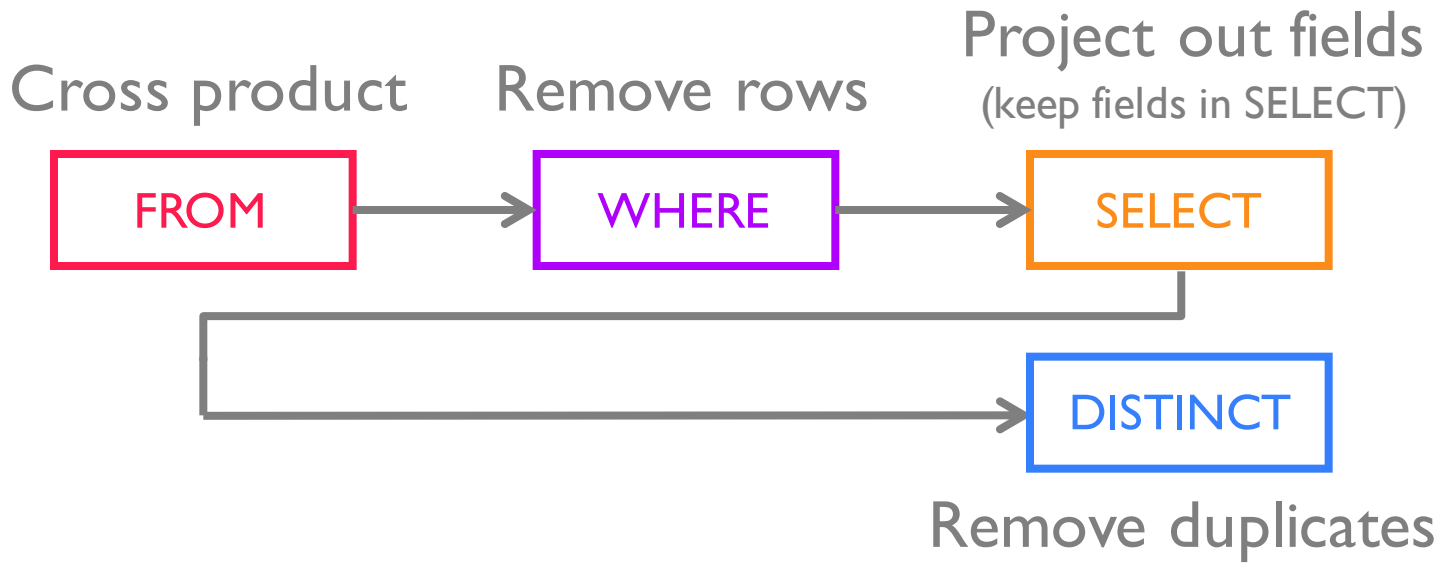
relation-list
List of relation names
Can define range-variable "AS X"

qualification
Boolean expressions

- Combined w/ AND, OR, NOT
- attr *op* const
- $attr_1$ *op* $attr_2$
- op is =, <, >, !=, etc

Conceptual Query Evaluation

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



Not how actually executed! Above is likely very slow

DISTINCT (vol.1)

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
2	103	9/14

```
SELECT  bid
FROM    Reserves
```

bid
102
102
103

```
SELECT  DISTINCT bid
FROM    Reserves
```

bid
102
103

Sailors that reserved 1+ boats

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

Would DISTINCT change anything in this query?

What if SELECT clause was SELECT S.name?

Range Variables

Disambiguate relations

same table used multiple times (self join)

```
SELECT  sid
FROM    Sailors, Sailors
WHERE   age > age
```

```
SELECT  S1.sid
FROM    Sailors AS S1, Sailors AS S2
WHERE   S1.age > S2.age
```

Range Variables

Disambiguate relations

same table used multiple times (self join)

```
SELECT  sid
FROM    Sailors, Sailors
WHERE   age > age
```

```
SELECT  S1.name, S1.age, S2.name, S2.age
FROM    Sailors AS S1, Sailors AS S2
WHERE   S1.age > S2.age
```

Expressions (Math)

```
SELECT   S.age, S.age – 5 AS age2, 2*S.age AS age3
FROM     Sailors AS S
WHERE    S.name = ‘eugene’
```

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

Expressions (Strings)

```
SELECT   S.name
FROM     Sailors AS S
WHERE    S.name LIKE 'e_%'
```

'_' any one character (. in regex)

'%' 0 or more characters of any kind (.* in regex)

Most DBMSes have rich string manipulation support e.g., regex

PostgreSQL documentation

http://www.postgresql.org/docs/9.1/static/functions-string.html

Expressions (Date/Time)

```
SELECT   R.sid
FROM     Reserves AS R
WHERE    now() - R.date < interval ‘1 day’
```

TIMESTAMP, DATE, TIME types

now() returns timestamp at start of transaction

DBMSes provide rich time manipulation support

exact support may vary by vender

Postgresql Documentation

http://www.postgresql.org/docs/9.1/static/functions-datetime.html

Expressions

Constant	2
Col reference	Sailors.name
Arithmetic	Sailors.sid * 10
Unary operators	NOT, EXISTS
Binary operators	AND, OR, IN
Function calls	abs(), sqrt(), ...
Casting	1.7::int, '10-12-2015'::date

sid of Sailors that reserved red or blue boat

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

OR

```
SELECT   R.sid
FROM     Boats B, Reserves R
WHERE    B.bid = R.bid AND B.color = ‘red’
UNION ALL
SELECT   R.sid
FROM     Boats B, Reserves R
WHERE    B.bid = R.bid AND B.color = ‘blue’
```

sid of Sailors that reserved red or blue boat

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

OR

```
SELECT   R.sid
FROM     Boats B, Reserves R
WHERE    B.bid = R.bid AND B.color = ‘red’
UNION
SELECT   R.sid
FROM     Boats B, Reserves R
WHERE    B.bid = R.bid AND B.color = ‘blue’
```

sid of Sailors that reserved red and blue boat

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

```
SELECT   R.sid
FROM     Boats B, Reserves R
WHERE    B.bid = R.bid AND B.color = ‘red’
INTERSECT ALL
SELECT   R.sid
FROM     Boats B, Reserves R
WHERE    B.bid = R.bid AND B.color = ‘blue’
```

sid of Sailors that reserved red and blue boat

Can use self-join instead

```
SELECT   R.sid
FROM     Boats B1, Reserves R1
WHERE
         B1.bid = R1.bid AND

         B1.color = ‘red’
```

sid of Sailors that reserved red and blue boat

Can use self-join instead

```
SELECT   R.sid
FROM     Boats B1, Reserves R1, Boats B2,Reserves R2
WHERE
         B1.bid = R1.bid AND

         B1.color = ‘red’
```

sid of Sailors that reserved red and blue boat

Can use self-join instead

```
SELECT   R.sid
FROM     Boats B1, Reserves R1, Boats B2,Reserves R2
WHERE
         B1.bid = R1.bid AND
         B2.bid = R2.bid AND
         B1.color = ‘red’ AND B2.color = ‘blue’
```

sid of Sailors that reserved red and blue boat

Can use self-join instead

```
SELECT   R.sid
FROM     Boats B1, Reserves R1, Boats B2,Reserves R2
WHERE    R1.sid = R2.sid AND
         B1.bid = R1.bid AND
         B2.bid = R2.bid AND
         B1.color = ‘red’ AND B2.color = ‘blue’
```

sids of sailors that haven't reserved a boat

```
SELECT   S.sid
FROM     Sailors S

EXCEPT

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

Can we write EXCEPT using more basic functionality?

EXCEPT ALL actually takes duplicates into account (multi-set cardinality) will then matter.

SET Comparison Operators

Binary: Relation *op* Relation

op: UNION, INTERSECT, EXCEPT

Binary: Tuple *op* Relation

op: IN, NOT IN

Unary: OP Relation

op: EXISTS, NOT EXISTS, UNIQUE, NOT UNIQUE

Turning Scalar operators into Set operators:

op ANY, *op* ALL

op $\in$ { <, >, =, ≤,≥,≠,…}

Many of these rely on Nested Query Support

Nested Queries

```
SELECT   S.sid
FROM     Sailors S
WHERE    S.sid IN (SELECT    R.sid
                   FROM      Reserves R
                   WHERE     R.bid = 101)
```

a “Subquery”

Nested Queries

```
SELECT  S.sid
FROM    Sailors S
WHERE
```


```
for S in Sailors
    if boolean_function(S):
        yield S.sid
```

Nested Queries

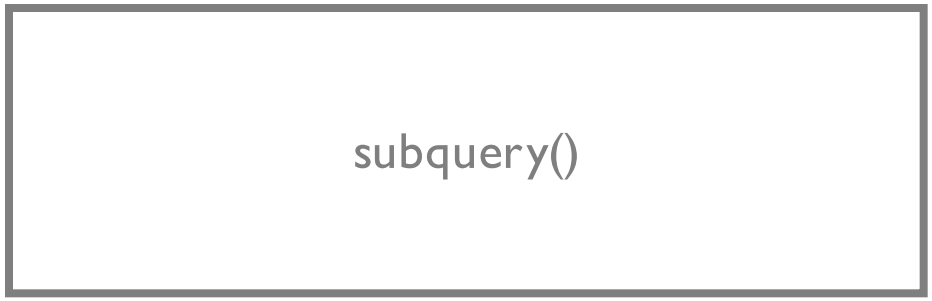
```
SELECT   S.sid
FROM     Sailors S
WHERE    S.sid IN
```

subquery()

```
for S in Sailors
    if S.sid in subquery():
        yield S.sid
```

Nested Queries

```
SELECT   S.sid
FROM     Sailors S
WHERE    S.sid IN
```




```
squery = subquery()
for S in Sailors
    if S.sid in squery:
        yield S.sid
```

Nested Queries

```
SELECT  S.sid
FROM    Sailors S
WHERE   S.sid IN (SELECT   R.sid
                  FROM     Reserves R
                  WHERE    R.bid = 101)
```

Many clauses can contain SQL queries

WHERE, FROM, HAVING, SELECT

Conceptual model:

```
for each Sailors tuple
   run the subquery and evaluate qualification
```

Nested Correlated Queries

```
SELECT   S.sid
FROM     Sailors S
WHERE    EXISTS (SELECT   *
                 FROM     Reserves R
                 WHERE    R.bid = 101 AND
                          S.sid = R.sid)
```

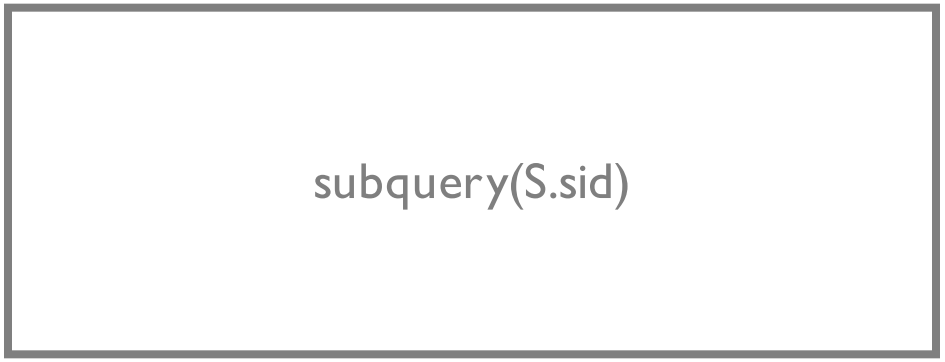
Outer table referenced in nested query

Conceptual model:

```
for each Sailors tuple
   run the subquery and evaluate qualification
```

Nested Correlated Queries

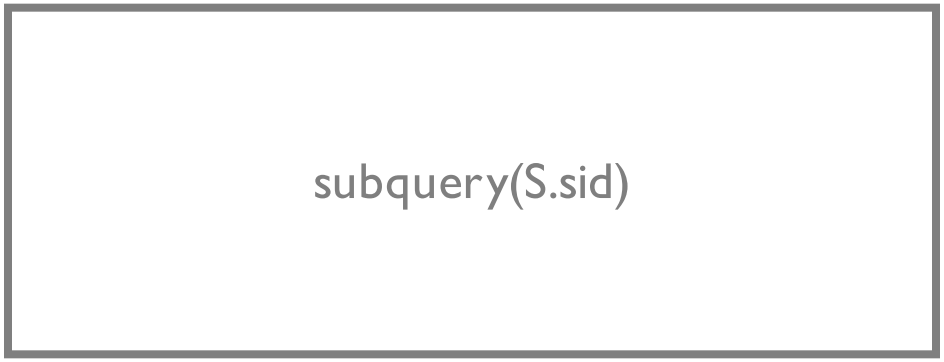
```
SELECT   S.sid
FROM     Sailors S
WHERE    EXISTS
```




```
squery = subquery(S.sid)
for S in Sailors
    if S.sid in squery:
        yield S.sid
```

Nested Correlated Queries

```
SELECT   S.sid
FROM     Sailors S
WHERE    EXISTS
```




```
for S in Sailors
    if squery(S.sid):
        yield S.sid
```

Nested Correlated Queries

```
SELECT  S.sid
FROM    Sailors S
WHERE   UNIQUE (SELECT  *
                FROM    Reserves R
                WHERE   R.bid = 101 AND
                        S.sid = R.sid)
```

UNIQUE checks that there are no duplicates

What does this do?

Nested Correlated Queries

```
SELECT  S.sid
FROM    Sailors S
WHERE   UNIQUE (SELECT  R.sid
                FROM    Reserves R
                WHERE   R.bid = 101 AND
                        S.sid = R.sid)
```

UNIQUE checks that there are no duplicates

What does this do?

Sailors whose rating is greater than any sailor named “Bobby”

```
SELECT S1.name
FROM   Sailors S1
WHERE  S1.rating > ANY (SELECT   S2.rating
                        FROM     Sailors S2
                        WHERE    S2.name = ‘Bobby’)
```

What about this?

```
SELECT S1.name
FROM   Sailors S1
WHERE  S1.rating > ALL (SELECT   S2.rating
                        FROM     Sailors S2
                        WHERE    S2.name = 'Bobby')
```

Sailors whose rating is greater than
ALL sailors named "Bobby"

Rewrite INTERSECT using IN

```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating > 2

INTERSECT

SELECT  R.sid
FROM    Reserves R
```

```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating > 2 AND
        S.sid IN (
            SELECT  R.sid
            FROM    Reserves R
        )
```

Similar trick for EXCEPT → NOT IN

What if want *names* instead of sids?

Rewrite INTERSECT using IN

```
SELECT S2.name
FROM Sailors S2, (
    SELECT S.sid
    FROM   Sailors S
    WHERE  S.rating > 2

    INTERSECT

    SELECT R.sid
    FROM   Reserves R
    ) as tmp
WHERE tmp.sid = S2.sid
```

```
SELECT  S.name
FROM    Sailors S
WHERE   S.rating > 2 AND
        S.sid IN (
            SELECT R.sid
            FROM   Reserves R
        )
```

Translation harder for INSTERSECT ALL

Sailors that reserved all boats (Division)

Hint: double negation

S reserved all boats == no boat that S didn't reserve

```
SELECT    S.name
FROM      Sailors S
WHERE     NOT EXISTS (

    (SELECT   B.bid FROM   Boats B)

    EXCEPT

    (SELECT   R.bid
     FROM Reserves R
     WHERE R.sid = S.sid)
)
```

Sailors that reserved all boats (Division)

Hint: double negation

S reserved all boats == ∄ boat B ∄ reservation of B by S

```
SELECT  S.name
FROM    Sailors S
WHERE   NOT EXISTS (
```

Sailors S such that

there's no boat B without

a reservation by S

Sailors that reserved all boats (Division)

Hint: double negation

S reserved all boats == ∄ boat B ∄ reservation of B by S

```
SELECT  S.name
FROM    Sailors S
WHERE   NOT EXISTS (SELECT B.bid
                    FROM   Boats B
                    WHERE  NOT EXISTS (
```

Sailors S such that

there's no boat B without

a reservation by S

Sailors that reserved all boats (Division)

Hint: double negation

S reserved all boats == ∄ boat B ∄ reservation of B by S

```
SELECT  S.name
FROM    Sailors S
WHERE   NOT EXISTS (SELECT B.bid
                    FROM   Boats B
                    WHERE  NOT EXISTS (SELECT R.bid
                                       FROM Reserves R
                                       WHERE  R.sid = S.sid
                                          AND R.bid = B.bid ))
```

Sailors S such that

there's no boat B without

a reservation by S

NULL

Field values sometimes unknown or inapplicable

SQL provides a special value *null* for such situations.

The presence of null complicates many issues e.g.,

Is age = null true or false?

Is null = null true or false?

Is null = 8 OR 1 = 1 true or false?

Special syntax "IS NULL" and "IS NOT NULL"

3 Valued Logic (true, false, unknown)

How does WHERE remove rows?

if qualification doesn't evaluate to true

New operators (in particular, outer joins) possible/needed.

NULL

(null > 0) = null

(null + 1) = null

(null = 0) = null

(null AND true) = null

null is null = true

Some truth tables

AND	T	F
T	T	F
F	F	F

OR	T	F
T	T	T
F	T	F

NULL

(null > 0) = null

(null + 1) = null

(null = 0) = null

(null AND true) = null

null is null = true

Some truth tables

AND	T	F	NULL
T	T	F	NULL
F	F	F	F

OR	T	F	NULL
T	T	T	T
F	T	F	NULL

NULL

(null > 0) = null

(null + 1) = null

(null = 0) = null

(null AND true) = null

null is null = true

Some truth tables

AND	T	F	NULL
T	T	F	NULL
F	F	F	F
NULL	NULL	F	NULL

OR	T	F	NULL
T	T	T	T
F	T	F	NULL
NULL	T	NULL	NULL

JOINS

```
SELECT [DISTINCT] target_list
FROM table_name
    [INNER | {LEFT |RIGHT | FULL } {OUTER}] JOIN table_name
    ON qualification_list
WHERE ...
```

INNER is default

Difference in how to deal with NULL values

PostgreSQL documentation:

http://www.postgresql.org/docs/9.4/static/tutorial-join.html

Inner/Natural Join

```
SELECT  s.sid, s.name, r.bid
FROM    Sailors S, Reserves r
WHERE   s.sid = r.sid

SELECT s.sid, s.name, r.bid
FROM    Sailors s INNER JOIN Reserves r
ON      s.sid = r.sid

SELECT  s.sid, s.name, r.bid
FROM    Sailors s NATURAL JOIN Reserves r
```

All Equivalent for example tables

Natural Join means equi-join for each pair of attrs with same name

Sailor names and their reserved boat ids

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

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Result

sid	name	bid
1	Eugene	102
2	Luis	102

Sailor names and their reserved boat ids

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

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Result

sid	name	bid
1	Eugene	102
2	Luis	102

Sailor names and their reserved boat ids

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

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Result

sid	name	bid
1	Eugene	102
2	Luis	102

Notice: No result for Ken!

Left Outer Join (or No Results for Ken)

Returns all matched rows *and all unmatched rows from table on left of join clause*

(*at least one* result row for each row in left table)

```
SELECT  s.sid, s.name, r.bid
FROM    Sailors s LEFT OUTER JOIN Reserves r
ON      s.sid = r.sid
```

All sailors & bid for boat in their reservations

Bid set to NULL if no reservation

Left Outer Join

```
SELECT  s.sid, s.name, r.bid
FROM    Sailors s LEFT OUTER JOIN Reserves r
ON      s.sid = r.sid
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Result

sid	name	bid
1	Eugene	102
2	Luis	102
3	Ken	NULL

Can Left Outer Join be expressed with Cross-Product?

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day

```
Sailors x Reserves
```

```
Sailors s LEFT OUTER JOIN Reserves r
ON  s.sid = r.sid
```

Result

sid	name	bid

Result

sid	name	bid
1	Eugene	NULL
2	Luis	NULL
3	Ken	NULL

Can Left Outer Join be expressed with Cross-Product?

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day

$$\text{Sailors} \bowtie \text{Reserves}$$

$$\cup$$

$$(\text{Sailors} - (\text{Sailors} \bowtie \text{Reserves})) \times \{(\text{null}, \ldots)\}$$

How to compute this with a query?

Joins as For Loops

```
for s in Sailors:

    for r in Reserves:
        if s.sid = r.sid:
            yield s, r
```

Inner Join

Joins as For Loops

```
for s in Sailors:
    bmatched = False
    for r in Reserves:
        if s.sid = r.sid:
            yield s, r
            bmatched = True

    if not bmatched:
        yield s, null
```

Left Outer Join

Right Outer Join

Same as LEFT OUTER JOIN, but guarantees result for rows in table on right side of JOIN

```
SELECT  s.sid, s.name, r.bid
FROM    Reserves r RIGHT OUTER JOIN Sailors S
ON      s.sid = r.sid
```

FULL OUTER JOIN

Returns all matched *or* unmatched rows from both sides of JOIN

```
SELECT  s.sid, s.name, r.bid
FROM    Sailors s FULL OUTER JOIN Reserves r
ON      s.sid = r.sid
```

FULL OUTER JOIN

```
SELECT  s.sid, s.name, r.sid, r.bid
FROM    Sailors s Full OUTER JOIN Reserves r
ON      s.sid = r.sid
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
4	109	9/20

Result

sid	name	sid	bid
1	Eugene	1	102
2	Luis	2	102
3	Ken	NULL	NULL
NULL	NULL	4	109

Functions as Joins

What is f(x) = x * 2?

What is f(8)?

x
8

x	f(x)
1	1*2 = 2
2	2*2 = 4
3	3*2 = 6
...	...

How big is this relation?

Serious people can count: Aggregation

```
SELECT  COUNT(*)
FROM    Sailors S

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

SELECT COUNT(DISTINCT S.name)
FROM    Sailors S
WHERE   S.name LIKE 'D%'

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

COUNT([DISTINCT] A)

SUM([DISTINCT] A)

AVG([DISTINCT] A)

MAX/MIN(A)

STDDEV(A)

CORR(A,B)

PostgreSQL documentation

http://www.postgresql.org/docs/9.4/static/functions-aggregate.html

Name and age of oldest sailor(s)

```
SELECT  S.name, MAX(S.age)
FROM    Sailors S
```

```
SELECT  S.name, S.age
FROM    Sailors S
WHERE   S.age >= ALL (SELECT  S2.age
                      FROM    Sailors S2)
```

```
SELECT  S.name, S.age
FROM    Sailors S
WHERE   S.age = (SELECT    MAX(S2.age)
                 FROM      Sailors S2)
```

```
SELECT  S.name, S.age
FROM    Sailors S
ORDER BY   S.age DESC
LIMIT 1
```

← When does this not work?

GROUP BY

```
SELECT min(s.age)
FROM   Sailors s
```

Minimum age among all sailors

What if want minimum age *per rating level*?

We don't even know how many rating levels exist!

If we did, could write (awkward):

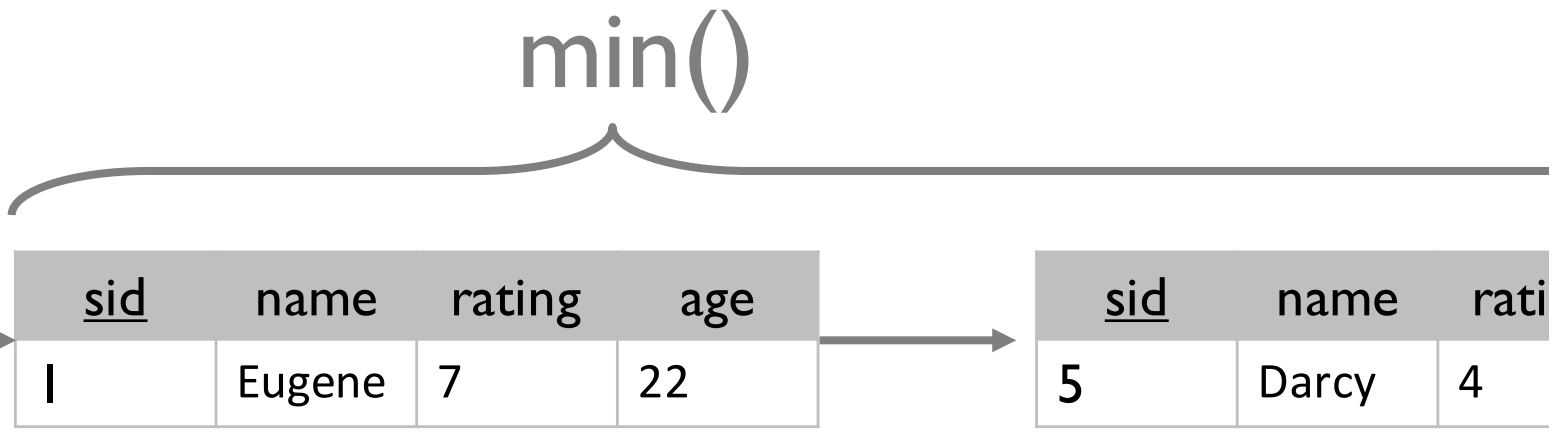
```
for rating in [0…10]
    SELECT min(s.age)
    FROM   Sailors s
    WHERE  s.rating = <rating>
```

What is a list in relational terms?

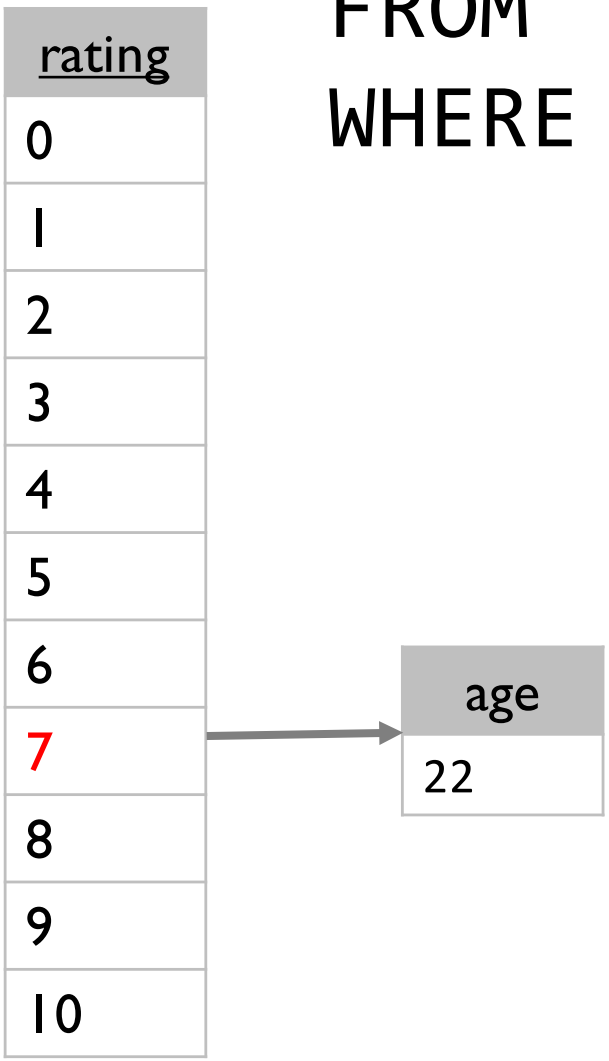
```
for rating in [0...10]
   SELECT min(s.age)
   FROM  Sailors s
   WHERE s.rating = <rating>
```

```
for rating in ratings
   SELECT min(s.age)
   FROM   Sailors s
   WHERE  s.rating = <rating>
```

rating
0
1
2
3
4
5
6
7
8
9
10



```
for rating in ratings
   SELECT min(s.age)
   FROM   Sailors s
   WHERE  s.rating = <rating>
```



GROUP BY

```
SELECT count(*)
FROM   Reserves R
```

Total number of reservations

What if want reservations per boat?

May not even know all our boats (depends on data)!

If we did, could write (awkward):

```
for boat in [100…131]
    SELECT count(*)
    FROM   Reserves R
    WHERE  R.bid = <boat>
```

GROUP BY

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

grouping-list is a list of expressions that defines groups

set of tuples w/ same value for all attributes in grouping-list

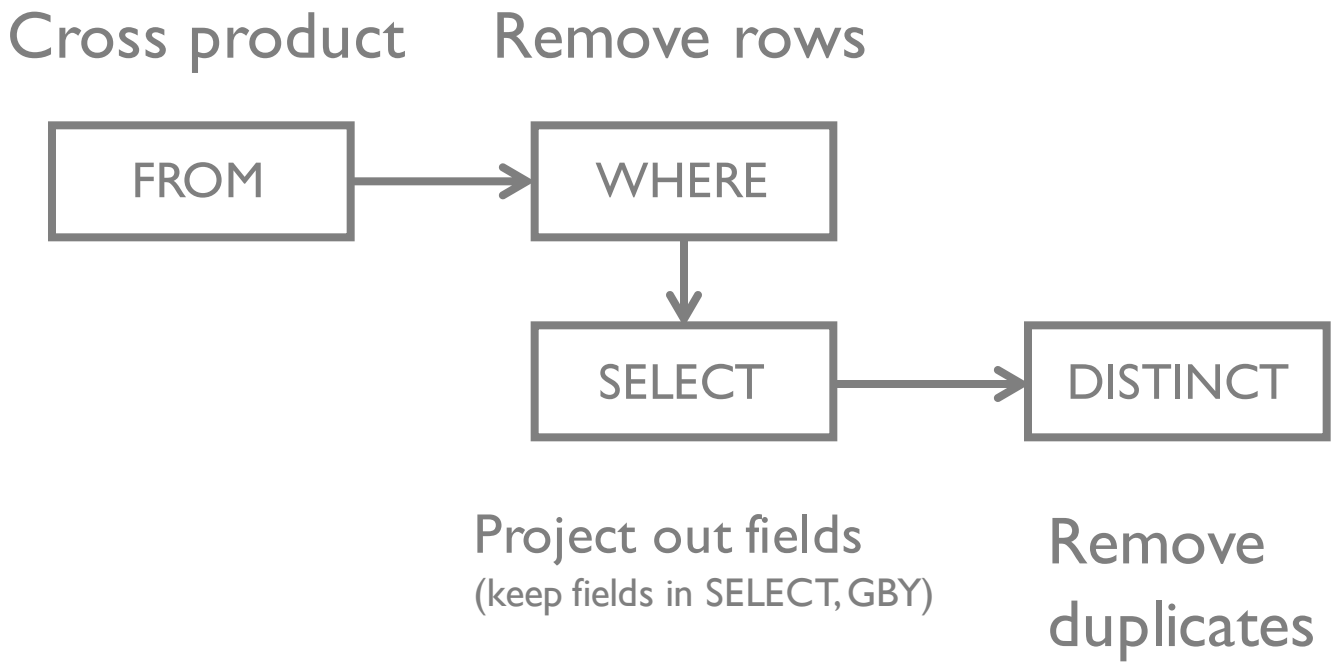
target-list contains

attribute-names $\subseteq$ *grouping-list*

aggregation expressions

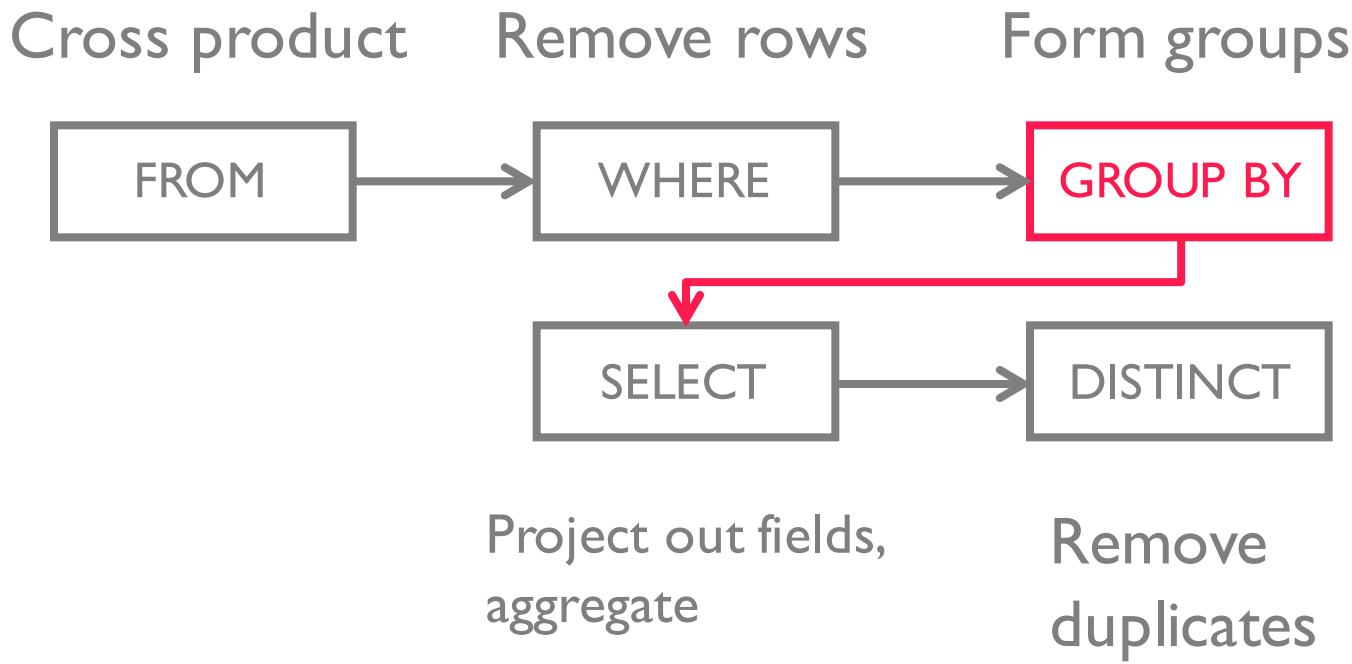
Conceptual Query Evaluation

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



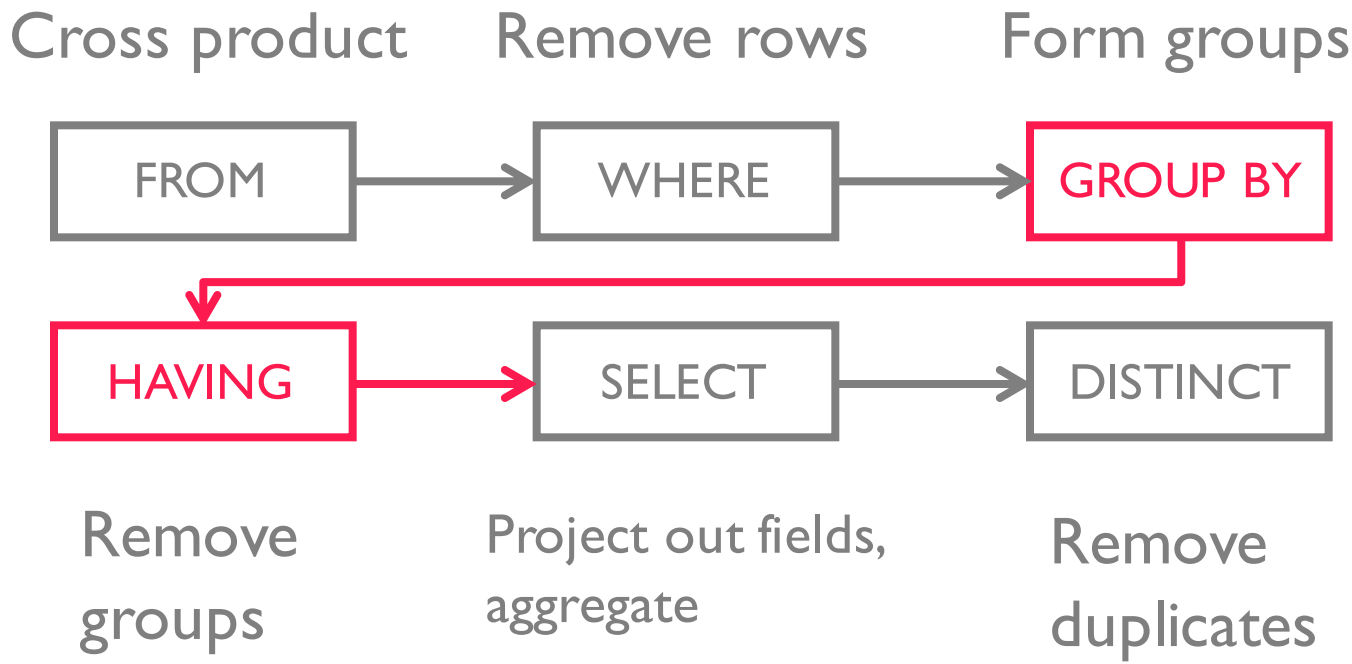
Conceptual Query Evaluation

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

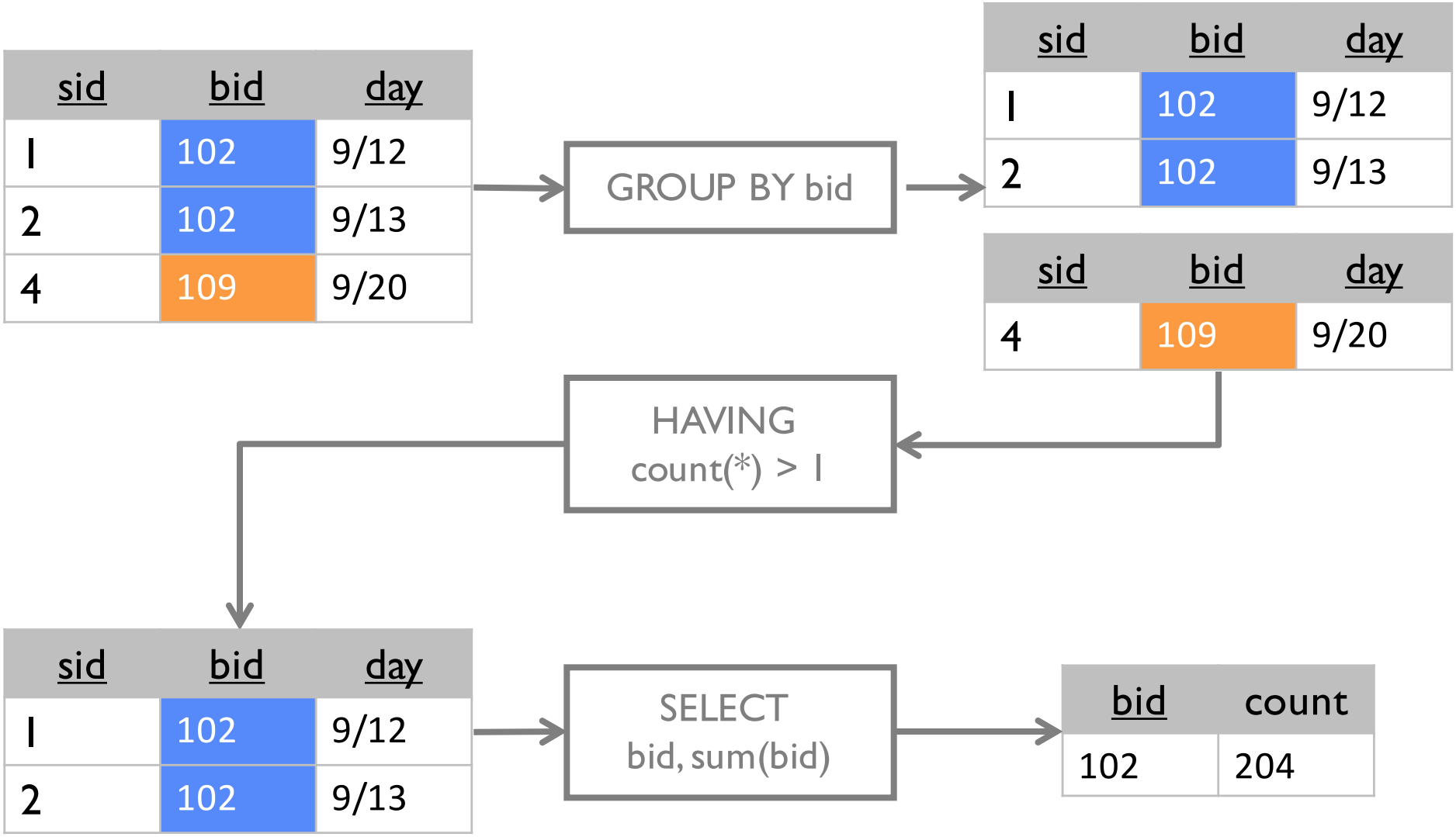


Conceptual Query Evaluation

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



Conceptual Evaluation



GROUP BY

```
SELECT    rating, min(age)
FROM      Sailors
GROUP BY  rating
```

Minimum age for each rating

```
SELECT    min(age)
FROM      Reserves R, Sailors S
WHERE     S.sid = R.sid
GROUP BY  bid
HAVING    count(*) > 2
```

Minimum sailor age

for each boat that has >2 reservations

HAVING

group-qualification used to remove groups

similar to WHERE clause

Expressions must have *one value per group*.

An aggregation function or in *grouping-list*

```
SELECT   bid, count(*)
FROM     Reserves R
GROUP BY bid
HAVING   color = 'red'
```

AVG age of sailors reserving red boats, by rating

```
SELECT
FROM      Sailors S, Boats B, Reserves R
WHERE     S.sid = R.sid AND
          R.bid = B.bid AND
          B.color = ‘red’
```

AVG age of sailors reserving red boats, by rating

```
SELECT    S.rating, avg(S.age) AS age
FROM      Sailors S, Boats B, Reserves R
WHERE     S.sid = R.sid AND
          R.bid = B.bid AND
          B.color = ‘red’
GROUP BY  S.rating
```

What if move `B.color=‘red’` to HAVING clause?

Error

Ratings where the avg age is min over all ratings

```
SELECT S.rating
FROM    Sailors S
WHERE   S.age = (
            SELECT MIN(AVG(S2.age))
            FROM Sailors S2
        )
```

```
SELECT S.rating
FROM    (SELECT S.rating, AVG(S.age) as avgage
           FROM Sailors S
        GROUP BY S.rating) AS tmp
WHERE   tmp.avgage = (
    SELECT MIN(tmp2.avgage) FROM (
        SELECT  S.rating, AVG(S.age) as avgage
          FROM  Sailors S
      GROUP BY  S.rating
    ) AS tmp2
)
```

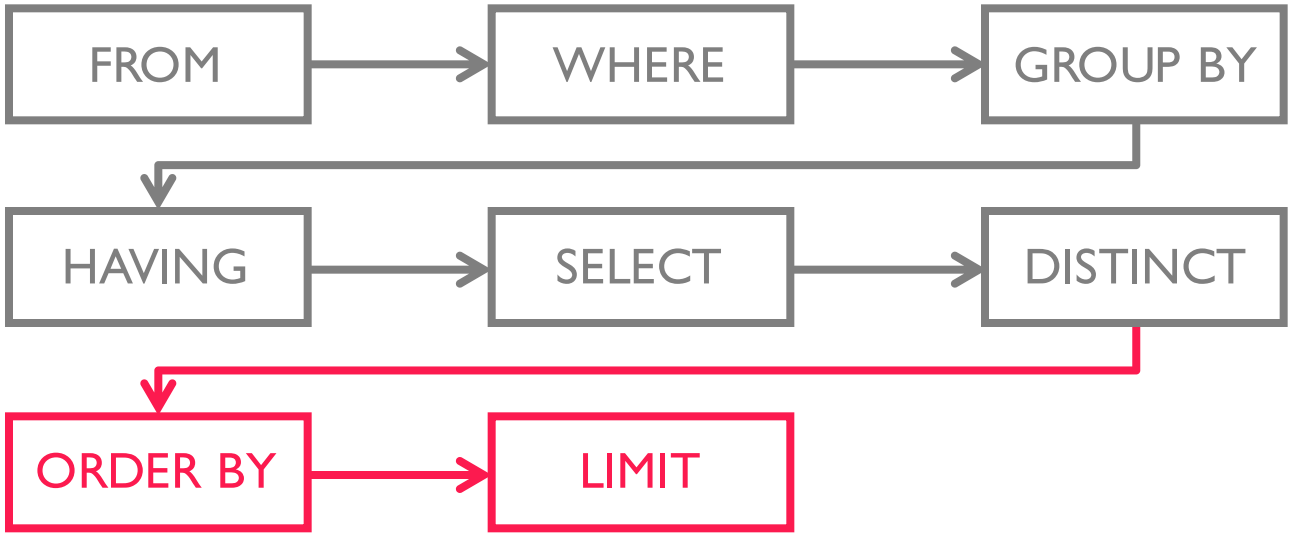
Ratings where the avg age is min over all ratings

```
SELECT S.rating
FROM    Sailors S
WHERE   S.age = (
            SELECT MIN(AVG(S2.age))
            FROM Sailors S2
        )
```

```
SELECT S.rating
FROM    (SELECT S.rating, AVG(S.age) as avgage
           FROM Sailors S
       GROUP BY S.rating) AS tmp
WHERE   tmp.avgage <= ALL (
    SELECT tmp2.avgage FROM (
        SELECT  S.rating, AVG(S.age) as avgage
          FROM  Sailors S
      GROUP BY  S.rating
    ) AS tmp2
)
```

ORDER BY, LIMIT

```
SELECT    [DISTINCT] target-list
FROM      relation-list
WHERE     qualification
GROUP BY  grouping-list
HAVING    group-qualification
ORDER BY  order-list
LIMIT     limit-expr [OFFSET offset-expr]
```



ORDER BY

```
SELECT    S.name
FROM      Sailors S
ORDER BY  (S.rating/2)::int ASC,
          S.age DESC
```

List of *order-list* expressions dictates ordering precedence

- Sorted in ascending by age/rating ratio
- If ties, sorted high to low rating

ORDER BY

```
SELECT    S.name, (S.rating/2)::int, S.age
FROM      Sailors S
ORDER BY  (S.rating/2)::int ASC,
          S.age DESC
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

name	int4	age
Luis	1	39
Ken	4	27
Eugene	4	22

ORDER BY

```
SELECT    S.name, (S.rating/2)::int, S.age
FROM      Sailors S
ORDER BY  (S.rating/2)::int ASC,
          S.age ASC
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

name	int4	age
Luis	1	39
Eugene	4	22
Ken	4	27

LIMIT

```
SELECT    S.name, (S.rating/2)::int, S.age
FROM      Sailors S
ORDER BY  (S.rating/2)::int ASC,
          S.age DESC
LIMIT     2
```

Only the first 2 results

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

name	int4	age
Luis	1	39
Ken	4	27

LIMIT

```
SELECT    S.name, (S.rating/2)::int, S.age
FROM      Sailors S
ORDER BY  (S.rating/2)::int ASC,
          S.age DESC
LIMIT     2 OFFSET 1
```

Only the first 2 results

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

name	int4	age
Ken	4	27
Eugene	4	22

LIMIT

```
SELECT    S.name, (S.rating/2)::int, S.age
FROM      Sailors S
ORDER BY  (S.rating/2)::int ASC,
          S.age DESC
LIMIT     (SELECT count(S2.*) / 2
           FROM Sailors AS S2)
```

Can have expressions instead of constants

Result

name	int4	age
Luis	1	39

Integrity Constraints

Conditions that every legal instance must satisfy

- Inserts/Deletes/Updates that violate ICs rejected
- Helps ensure app semantics or prevent inconsistencies

We've discussed

- domain/type constraints, primary/foreign key
- general constraints ⟵

Beyond Keys: Table Constraints

Runs when table is not empty

```
CREATE TABLE Sailors(
    sid int,
    …
    PRIMARY KEY (sid),
    CHECK (rating >= 1 AND rating <= 10)

CREATE TABLE Reserves(
    sid int,
    bid int,
    day date,
    PRIMARY KEY (bid, day),
    CONSTRAINT no_red_reservations
    CHECK (‘red’ NOT IN (SELECT B.color
                         FROM Boats B
                         WHERE B.bid = bid))
```

Nested subqueries

Named constraints

Multi-Relation Constraints

of sailors + # of boats should be less than 100

```
CREATE TABLE Sailors (
    sid int,
    bid int,
    day date,
    PRIMARY KEY (bid, day),
    CHECK (
        (SELECT COUNT(S.sid) FROM Sailors S)
        +
        (SELECT COUNT(B.bid) FROM Boats B)
        < 100
    )
```

What if Sailors is empty?

Only runs if Sailors has rows (ignores Boats)

ASSERTIONS: Multi-Relation Constraints

```
CREATE ASSERTION small_club
CHECK (
    (SELECT COUNT(*) FROM Sailors S)
    +
    (SELECT COUNT(*) FROM Boats B)
    < 100
)
```

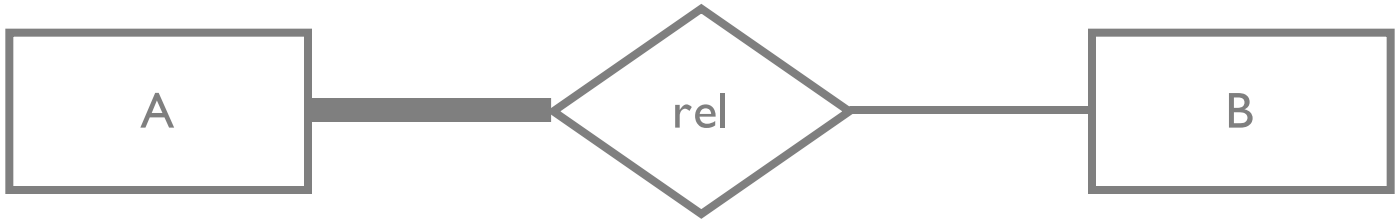
ASSERTIONs are not associated with any table

Total Participation

So many things we can't express or don't work!

Assertions

Nested queries in CHECK constraints



Advanced Stuff

User defined functions

Triggers

WITH

Views

User Defined Functions (UDFs)

Custom functions that can be called in database

Many languages: SQL, python, C, perl, etc

```
CREATE FUNCTION function_name(p1 type, p2 type, …)
RETURNS type
```

User Defined Functions (UDFs)

Custom functions that can be called in database

Many languages: SQL, python, C, perl, etc

```
CREATE FUNCTION function_name(p1 type, p2 type, …)
RETURNS type
AS $$

-- logic

$$ LANGUAGE language_name;
```

User Defined Functions (UDFs)

Custom functions that can be called in database

Many languages: SQL, python, C, perl, etc

```
CREATE FUNCTION function_name(p1 type, p2 type, …)
RETURNS type
AS $$

-- logic

$$ LANGUAGE language_name;
```

SQL, PL/SQL, Python, ...

A simple UDF (lang = SQL)

```
CREATE FUNCTION mult1(v int) RETURNS int
AS $$
SELECT v * 100;
$$ LANGUAGE SQL;
```

Schema!

Last statement is returned

```
CREATE FUNCTION function_name(p1 type, p2 type, …)
RETURNS type
AS $$

-- logic

$$ LANGUAGE language_name;
```

A simple UDF (lang = SQL)

```
CREATE FUNCTION mult1(v int) RETURNS int
AS $$
SELECT v * 100;
$$ LANGUAGE SQL;

SELECT mult1(S.age)
FROM   sailors AS S
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
220
390
270

http://www.postgresql.org/docs/9.1/static/xfunc-sql.html

A simple UDF (lang = SQL)

```
CREATE FUNCTION mult1(v int) RETURNS int
AS $$
SELECT $1 * 100;
$$ LANGUAGE SQL;

SELECT mult1(S.age)
FROM   sailors AS S
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
220
390
270

http://www.postgresql.org/docs/9.1/static/xfunc-sql.html

Process a Record (lang = SQL)

```
CREATE FUNCTION mult2(x sailors) RETURNS float
AS $$
SELECT (x.sid + x.age) / x.rating;
$$ LANGUAGE SQL;

SELECT mult2(S.*) AS v
FROM   sailors AS S
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

v
3.285
20.5
3.75

http://www.postgresql.org/docs/9.1/static/xfunc-sql.html

Process a Record (lang = SQL)

```
CREATE FUNCTION mult2(sailors) RETURNS int
AS $$
SELECT ($1.sid + $1.age) / $1.rating;
$$ LANGUAGE SQL;

SELECT mult2(S.*)
FROM   sailors AS S
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Result

int4
3.285
20.5
3.75

http://www.postgresql.org/docs/9.1/static/xfunc-sql.html

Procedural Language/SQL(lang = plsql)

```
CREATE FUNCTION proc(v int) RETURNS int
AS $$
DECLARE
    -- define variables
BEGIN
    -- PL/SQL code
END;
$$ LANGUAGE plpgsql;
```

http://www.postgresql.org/docs/9.4/static/plpgsql.html

Procedural Language/SQL(lang = plsql)

```
CREATE FUNCTION proc(v int) RETURNS int
AS $$
DECLARE
    -- define variables.  VAR TYPE [= value]
    qty int = 10;
BEGIN
    qty = qty * v;
    IF (SELECT COUNT(*) FROM foo) > 10 THEN
        INSERT INTO blah VALUES(qty);
    END IF;
    RETURN qty + 2;
END;
$$ LANGUAGE plpgsql;
```

http://www.postgresql.org/docs/9.4/static/plpgsql.html

Procedural Code (lang = plpython2u)

```
CREATE FUNCTION proc(v int) RETURNS int
AS $$
import random
return random.randint(0, 100) * v
$$ LANGUAGE plpython2u;
```

Very powerful – can do anything so must be careful

run in a python interpreter with no security protection

`plpy` module provides database access

```
plpy.execute("select 1")
```

http://www.postgresql.org/docs/9.4/static/plpython.html

Procedural Code (lang = plpython2u)

```
CREATE FUNCTION proc(word text) RETURNS text
AS $$
import requests
resp = requests.get('http://google.com/search?q=%s' % v)
return resp.content.decode('unicode-escape')
$$ LANGUAGE plpython2u;
```

Very powerful – can do anything so must be careful

- run in a python interpreter with no security protection

`plpy` module provides database access

- `plpy.execute("select 1")`

http://www.postgresql.org/docs/9.4/static/plpython.html

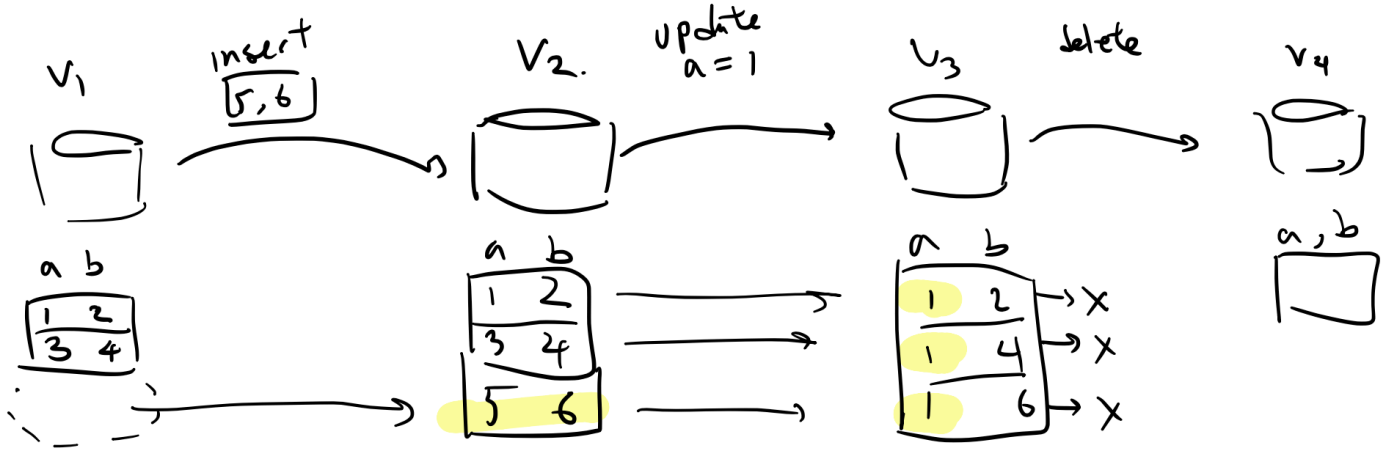
Administrivia

Midterm 1: Havemeyer 309, 3/7 8:40-9:55am

HW2 due 3/5

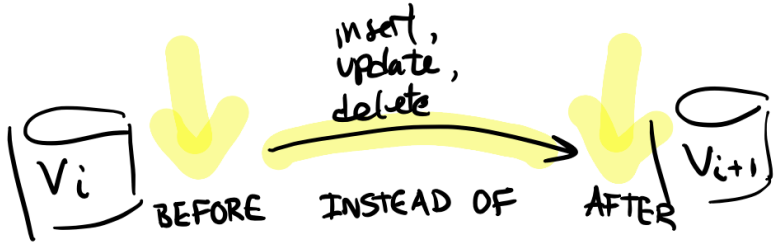
Triggers (background)

- Recall that a database instance is the database schema + the specific records
- Changing a DB instance essentially creates a new DB instance because the records are different. Let's call each instance a "version" of the DB
- Let's say we made 3 separate changes

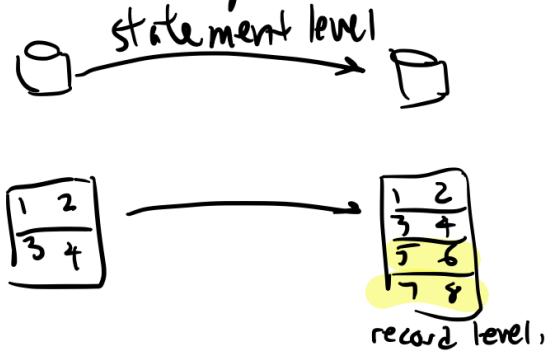


Triggers (background)

- When/where can we add trigger logic?



- At what granularity?



Does a SELECT query create a new version?

No

Triggers (logical)

def: procedure that runs automatically if specified changes in DBMS happen

```
CREATE TRIGGER name
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Triggers (logical)

def: procedure that runs automatically if specified changes in DBMS happen

```
CREATE TRIGGER name
    [BEFORE | AFTER | INSTEAD OF] event_list
    ON table
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Triggers (logical)

def: procedure that runs automatically if specified changes in DBMS happen

```
CREATE TRIGGER name
    [BEFORE | AFTER | INSTEAD OF] event_list
    ON table

    WHEN trigger_qualifications
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Triggers (logical)

def: procedure that runs automatically if specified changes in DBMS happen

```
CREATE TRIGGER name
    [BEFORE | AFTER | INSTEAD OF] event_list
    ON table
    [FOR EACH ROW]
    WHEN trigger_qualifications
    procedure
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Copy new young sailors into special table *(logical)*

```
CREATE TRIGGER youngSailorUpdate
    AFTER INSERT ON SAILORS
REFERENCING NEW TABLE NewInserts
FOR EACH STATEMENT
    INSERT
        INTO YoungSailors(sid, name, age, rating)
        SELECT sid, name, age, rating
        FROM NewInserts N
        WHERE N.age <= 18
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Copy new young sailors into special table
(logical)

```
CREATE TRIGGER youngSailorUpdate
    AFTER INSERT ON SAILORS
FOR EACH ROW
    WHEN NEW.age <= 18
    INSERT
        INTO YoungSailors (sid, name, age, rating)
        VALUES (NEW.sid, NEW.name, NEW.age, NEW.rating)
```

Event activates the trigger

Condition tests if triggers should run

Action what to do

Triggers *(logical)*

Can be complicated to reason about

- Triggers may (e.g., insert) cause other triggers to run
- If > 1 trigger match an action, which is run first?
- ¯_(ツ)_/¯

```
CREATE TRIGGER recursiveTrigger
    AFTER INSERT ON SAILORS
FOR EACH ROW
    INSERT INTO Sailors(sid, name, age, rating)
        SELECT sid, name, age, rating
        FROM Sailors S
```

Triggers vs Constraints

Constraint

- Statement about state of database
- Upheld by the database for *any* modifications
- Doesn't modify the database state
- *Safe*

Triggers

- Operational: X should happen when Y
- Specific to statements
- *Very* flexible

Triggers (postgres)

```
CREATE TRIGGER name
    [BEFORE | AFTER] event_list ON table
    FOR EACH (ROW | STATEMENT)
    WHEN trigger_qualifications
    EXECUTE PROCEDURE user_defined_function();
```

PostgreSQL only runs *trigger* UDFs

http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html
http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html

Trigger Example

```
CREATE FUNCTION copyrecord() RETURNS trigger
AS $$
BEGIN
    INSERT INTO blah VALUES(NEW.a);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Signature: no args, return type is trigger

Returns NULL or same record structure as modified row

Special variables: OLD, NEW

```
CREATE TRIGGER t_copyinserts BEFORE INSERT ON a
    FOR EACH ROW
        EXECUTE PROCEDURE copyrecord();
```

http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html
http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html

Total boats and sailors < 100

```
CREATE FUNCTION checktotal() RETURNS trigger
AS $$
BEGIN
    IF ((SELECT COUNT(*) FROM sailors) +
        (SELECT COUNT(*) FROM boats) < 100) THEN
        RETURN NEW
    ELSE
        RETURN null;
    END IF;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER t_checktotal BEFORE INSERT ON sailors
    FOR EACH ROW
        EXECUTE PROCEDURE checktotal();
```

http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html
http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html

You can get into trouble…

```
CREATE FUNCTION addme_bad() RETURNS trigger
AS $$
BEGIN
    INSERT INTO a VALUES (NEW.*);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER t_addme_bad BEFORE INSERT ON a
    FOR EACH ROW
        EXECUTE PROCEDURE addme_bad();
```

http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html
http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html

You can get into trouble…

```
CREATE FUNCTION addme_stillwrong() RETURNS trigger
AS $$
BEGIN
    IF (SELECT COUNT(*) FROM a) < 100 THEN
        INSERT INTO a VALUES (NEW.a + 1);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER t_addme_stillwrong BEFORE INSERT ON a
    FOR EACH ROW
        EXECUTE PROCEDURE addme_stillwrong();
```

http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html
http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html

You can get into trouble…

```
CREATE FUNCTION addme_works() RETURNS trigger
AS $$
BEGIN
    IF (SELECT COUNT(*) FROM a) < 100 THEN
        INSERT INTO a VALUES (NEW.a + 1);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER t_addme_works AFTER INSERT ON a
    FOR EACH ROW
        EXECUTE PROCEDURE addme_works();
```

http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html
http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html

WITH

```
WITH RedBoats(bid, count) AS
    (SELECT   B.bid, count(*)
     FROM     Boats B, Reserves R
     WHERE    R.bid = B.bid AND B.color = ‘red’
    GROUP BY  B.bid)
SELECT    name, count
FROM      Boats AS B,  RedBoats AS RB
WHERE     B.bid = RB.bid AND count < 2
```

Names of unpopular boats

WITH

```
WITH RedBoats(bid, count) AS
   (SELECT   B.bid, count(*)
    FROM     Boats B, Reserves R
    WHERE    R.bid = B.bid AND B.color = ‘red’
   GROUP BY  B.bid)
SELECT    name, count
FROM      Boats AS B,  RedBoats AS RB
WHERE     B.bid = RB.bid AND count < 2
```

```
WITH tablename(attr1, …) AS (select_query)
   [,tablename(attr1, …) AS (select_query)]
main_select_query
```

Recursive WITH

```
WITH RECURSIVE t(n) AS (
   VALUES (1)
   UNION [ALL]
   SELECT n+1 FROM t
)
SELECT sum(n) FROM t;
```

Is there a problem with this query?

Recursive WITH

```
WITH RECURSIVE t(n) AS (
   VALUES (1)
   UNION [ALL]
   SELECT n+1 FROM t WHERE n < 10
)
SELECT sum(n) FROM t;
```

Fibonacci Series up to 50

```
WITH RECURSIVE fib(n,m) AS (
   VALUES (0,1)
   UNION
   ???

)
SELECT distinct n
   FROM fib
 WHERE n < 50;
```

Fibonacci Series up to 50

```
WITH RECURSIVE fib(n,m) AS (
   VALUES (0,1)
   UNION
   SELECT m,n+m FROM fib

)
SELECT distinct n
   FROM fib
 WHERE n < 50;
```

Fibonacci Series up to 50

```
WITH RECURSIVE fib(n,m) AS (
   VALUES (0,1)
   UNION
   SELECT m,n+m FROM fib
   WHERE n < 50
)
SELECT distinct n
   FROM fib
 WHERE n < 50;
```

Views

```
CREATE VIEW view_name
AS select_statement
```

"tables" defined as query results rather than inserted base data

- Makes development simpler
- Used for security

Not *materialized*

References to *view_name* replaced with *select_statement*

Similar to WITH, lasts longer than one query

Names of popular boats

```
CREATE VIEW boat_counts
AS  SELECT      bid, count(*)
    FROM        Reserves R
    GROUP BY    bid
    HAVING      count(*) > 10
```

Used like a normal table

```
SELECT  bname
FROM    boat_counts bc, Boats B
WHERE   bc.bid = B.bid
```

Names of popular boats

```
SELECT  bname
FROM
    (SELECT bid, count(*)
     FROM Reserves R
     GROUP BY bid
     HAVING count(*) > 10) bc,
    Boats B
WHERE   bc.bid = B.bid
```

Rewritten expanded query

CREATE TABLE

```
CREATE TABLE <table_name> AS
    <SELECT STATEMENT>
```

Guess the schema:

```
CREATE TABLE used_boats1 AS
    SELECT r.bid
    FROM   Sailors s,
           Reservations r
    WHERE  s.sid = r.sid
```

```
used_boats1(bid int)
```

```
CREATE TABLE used_boats2 AS
    SELECT r.bid as foo
    FROM   Sailors s,
           Reservations r
    WHERE  s.sid = r.sid
```

```
used_boats2(foo int)
```

How is this different than views?

- What if we insert a new record into Reservations?

Summary

SQL is pretty complex

- Superset of Relational Algebra SQL99 turing complete!
- Human readable

More than one way to skin a horse

- Many alternatives to write a query
- Optimizer (theoretically) finds most efficient plan

additional slides

Some Tricky Queries

Lets write some tricky queries

- social graph analysis
- statistics

Social Network

```
-- A directed friend graph. Store each link once
CREATE TABLE Friends(
    fromID int,
    toID int,
    since date,
    PRIMARY KEY (fromID, toID),
    FOREIGN KEY (fromID) REFERENCES Users,
    FOREIGN KEY (toID) REFERENCES Users,
    CHECK (fromID < toID));

-- Return edges in both directions
CREATE VIEW BothFriends AS
    SELECT * FROM Friends
    UNION
    SELECT F.toID, F.fromID, F.since
    FROM Friends F;
```

friends of friends of friends do I have?

```
SELECT    count(distinct F3.toID)
FROM      BothFriends F1,
          BothFriends F2,
          BothFriends F3
WHERE     F1.toID = F2.fromID AND
          F2.toID = F3.fromID AND
          F1.fromID = <myid>;
```

friends of friends for each user?

```
SELECT    F1.fromID,  count(distinct F3.toID)
FROM      BothFriends F1,
          BothFriends F2,
          BothFriends F3
WHERE     F1.toID = F2.fromID AND
          F2.toID = F3.fromID
GROUP BY  F1.fromID;
```

Median

Given n values in sorted order, value at idx n/2

if n is even, can take lower of middle 2

Robust statics compared to avg

- if want avg to equal 0, what fraction of values need to be corrupted?
- if want median to be 0, what fraction?

Breakdown point of a statistic

crucial if there are outliers

helps with over-fitting

Median

Given n values in sorted order, value at idx n/2

```
SELECT     T.c
FROM       T
ORDER BY   T.c
LIMIT      1
OFFSET     (SELECT COUNT(*)/2
            FROM T AS T2)
```

Administrivia

TWO Exam locations!!
501 NWC if the last UNI digit is 0 - 5
Pupin 329 if the last UNI digit is 6 – 9

Professor Wu's OH: Weds 3:30-4:30PM

Guest lecture next Tuesday (10/23)
Dr. Thibault Sellam. Application APIs

Lecture cancelled next Thursday (10/25)

Median

Given n values in sorted order, value at idx n/2

```
SELECT  c AS median
FROM    T
WHERE
   (SELECT COUNT(*) FROM T AS T1
    WHERE T1.c < T.c)
   =
   (SELECT COUNT(*) FROM T AS T2
    WHERE T2.c > T.c);
```

Faster Median

```
SELECT  x.c as median
FROM    T x, T y
GROUP BY x.c
HAVING
   SUM((y.c <= x.c)::int) >= (COUNT(*)+1)/2
   AND
   SUM((y.c >= x.c)::int) >= (COUNT(*)/2)+1
```

Window Functions (not on exam)

How to run queries over ordered data

- Partition over a sequence of rows
- Each row can be in multiple partitions

```
aggregation OVER (
    [PARTITION BY attrs]
    [ORDER BY attrs]
)
```

Window Functions (not on exam)

1,1,2,3,4,4,5,6

```
SELECT count(*) OVER (PARTITION BY c)
FROM T

for row in T
    partition = (SELECT * FROM T
                 WHERE T.c =  row.c)
    count = len(partition)
    # add count to output row
```

Window Functions (not on exam)

1,1,2,3,4,4,5,6

```
SELECT row_number() OVER (ORDER BY c)
FROM T

for row in T
    partition = (SELECT * FROM T ORDER BY C)
    row_num = idx of row in partition
    # add rank to output row
```

Window Functions (not on exam)

1,1,2,3,4,4,5,6

```
SELECT row_number() OVER (PARTITION BY C ORDER BY c)
FROM T

for row in T
    partition = (SELECT * FROM T ORDER BY C)
    row_num = idx of row in partition
    # add rank to output row
```

Window Functions (Median)

How to run queries over ordered data

- O(n logn)
- Works with even # of items

```
CREATE VIEW twocounters AS
(SELECT   c,
          ROW_NUMBER() OVER (ORDER BY c ASC) AS RowAsc,
          ROW_NUMBER() OVER (ORDER BY c DESC) AS RowDesc
 FROM T);


SELECT AVG(c)
FROM twocounters
WHERE RowAsc IN (RowDesc, RowDesc - 1, RowDesc + 1);
```