Administrative Notes

HW2 out!

Conceptual Evaluation

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

GROUP BY bid

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

sid	bid	day
4	109	9/20

HAVING
count(*) > 1

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

SELECT
bid, sum(bid)

bid	count
102	204

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(*) > 1
```

Minimum sailor age
for each boat
with more than 1 reservation

HAVING

group-qualification used to remove groups

similar to WHERE clause

Expressions must have *one value per group*. Either

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
)
```

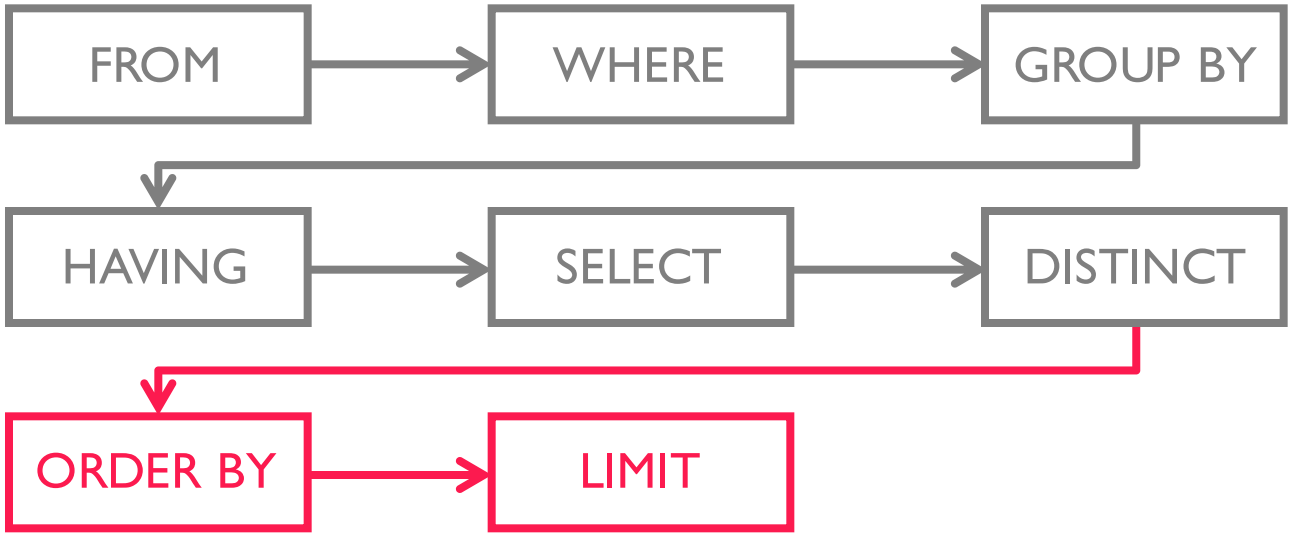
Ratings where the avg age is min over all ratings

```
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
)
```

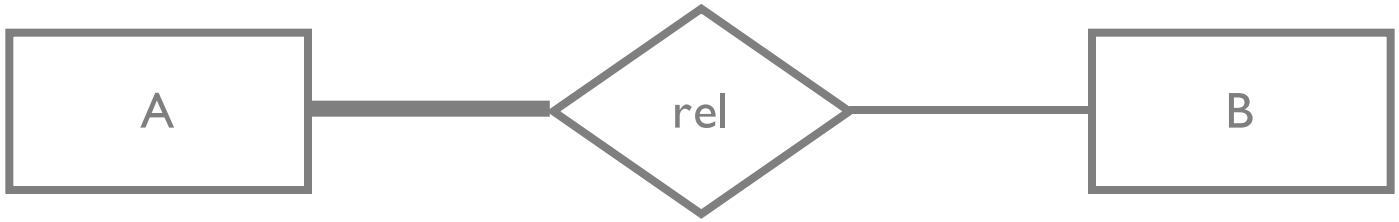
ASSERTIONs are not associated with any table

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 int
AS $$
SELECT (x.sid + x.age) / x.rating;
$$ 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

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

Aside: Control Flow in SQL

```
if (a > b):
  return a
return b
```

```
SELECT  (a > b)*a +
        (a <= b) * b
```

Aside: Control Flow in SQL

```
if (a > b):
  if (a > c):
    return a
  return c
return b
```

```
SELECT  (a > b)*a +
        (a <= b) * b
```

Aside: Control Flow in SQL

```
if (a > b):
  if (a > c):
    return a
  return c
return b
```

```
SELECT  (a > b)*(a > c)*a +
        (a > b)*(a <= c)*c +
        (a <= b) * b
```

Procedural Language/SQL(lang = plsql)

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

Boilerplate

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

Procedural Language/SQL(lang = plsql)

```
CREATE FUNCTION proc(v int) RETURNS int
AS $$
DECLARE
    -- define variables.  VAR TYPE [= value]
    qty int = 10;
BEGIN
    qty = qty * v;
    INSERT INTO blah VALUES(qty);
    RETURN qty + 2;
END;
$$ 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