Carnegie Mellon University
Department of Computer Science
15-415/615 - Database Applications
C. Faloutsos, A. Pavlo , Spring 2015

Homework 2 (by Elomar Souza) - Solutions

Due: **hard and e-copy**, at **1:30pm, Feb. 10, 2015**

VERY IMPORTANT:

1. OPTIONAL: by Thu 2/5, **verify** you logged in on postgres, by responding to the test question on the *Blackboard* (0 points).
2. **Electronic copy** of a `tar`-file with your queries, via *Blackboard*, before the due date (Feb. 10, 2015, 1:30pm). See page 2 ('*What to deliver*'), for details.
3. **Hard copy** of your SQL queries, **and** your output, in **class** at 1:30pm, on Tuesday, Feb. 10, 2015.

Reminders

- *Plagiarism*: Homework is to be done **individually**.
- *Typeset* all of your answers. Handwritten work will get zero points.
- *Late homeworks*: Standard policy: email (a) to all TAs, (b) with the subject line exactly `15-415 Homework Submission (HW 2)`, and (c) the count of slip-days you are using.

For your information:

- Graded out of **100** points
- **4** questions total
- Expected effort: ≈ 4-10h (1-2h to set up postgres; 1-2h per question).

Revision : 2015/02/25 23:53

Question	Points	Score
Oscar Nominations: Part I	25	
Oscar Nominations: Part II	25	
Steelers Season	25	
Gerrymandering	25	
Total:	100	

Preliminaries

Postgres set-up

Before starting the homework, follow the instructions for setting up PostgreSQL and importing the data we'll be working with, available at `http://www.cs.cmu.edu/~christos/courses/dbms.S15/hws/HW2/postgresql-setup.html`.

What to deliver: Check-list

The three items, as mentioned on the front page:

1. *OPTIONAL* (0 points):
   - What: the answer to the **test** question
   - When: Thu Feb. 5
   - Where: on *Blackboard*

   Our goal is to remind you to log-in on postgres, to avoid last-minute issues.
2. **MANDATORY - hard copy**:
   - What: hard copy of your **SQL queries**, plus their **output**.
   - When: Feb. 10, 2015, 1:30pm
   - Where: in class

   As before, please separate your answers for each question, providing on all answers the usual information (`course#`, `Homework#`, `Question#`, `Andrew ID`, `name`).
3. **MANDATORY - tar-file**:
   - What: A `tar` file (`<your-andrew-id>.tar`) with all your code - no need to include the answers.
   - When: Feb. 10, 2015, 1:30pm
   - Where: on *Blackboard*, under 'Assignments'/'Homework #2'

   Please use the template provided at `www.cs.cmu.edu/~christos/courses/dbms.S15/hws/HW2/hw2.tar` - just insert your SQL query to each place-holder `.sql` file.
   The goal is that, when the grader types `make`, he/she should see all your answers to all the questions.

FAQs

- *Q: May we use* `views`*?*
- A: Yes - you may use any feature of SQL that is supported by postgres on the andrew linux machines.
- *Q: What if a question is unclear?*
- A: Please post on blackboard; or write down your assumptions, and solve *your* interpretation of the query. We will accept all reasonable interpretations.

Question 1: Oscar Nominations: Part I . . . . . [25 points]

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

Graded by: Elomar

For this question you will look into Oscar nominations throughout history in four categories: "Leading Actor", "Leading Actress", "Supporting Actor", and "Supporting Actress". Figure 1 gives the schema of the tables you will use. For example, the tuple in the `nominations` table

(1142, 1939, 3, 602, "Gone with the Wind", "Scarlett O'Hara", t)

means that the #1142 nomination record was in the year 1939, for person #602 (=Vivien Leigh), for category #3 (=Leading actress), for the role of "Scarlett O'Hara", in the movie "Gone with the Wind" - and did indeed win.

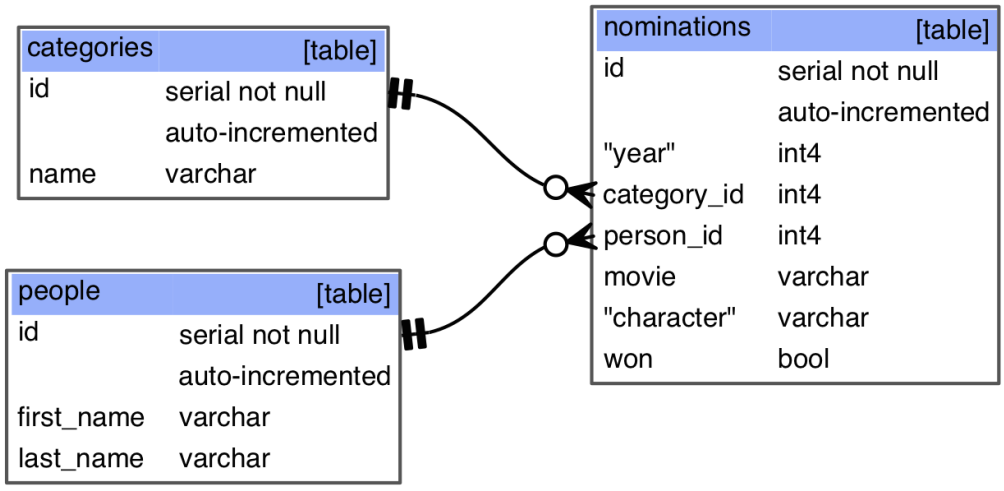

Figure 1: Tables for Oscar Nominations

(a) **[1 point]** (Warm-up): List all 4 categories (`id` and `name`), sorted on `name`.

Solution:

```
select * from categories order by name;
```

id	*name*
1	Leading Actor
3	Leading Actress
2	Supporting Actor
4	Supporting Actress

(b) **[6 points]** List the unique titles (`movie`), of all movies nominated in 1990, sorted alphabetically.

Solution:

```
select distinct(movie) from nominations where year = 1990
order by movie;
```

movie
Awakenings
Cyrano de Bergerac
Dances With Wolves
Dick Tracy
Ghost
Good Fellas
Longtime Companion
Misery
Mr. & Mrs. Bridge
Postcards from the Edge
Pretty Woman
Reversal of Fortune
The Field
The Godfather, Part III
The Grifters
Wild at Heart

(16 rows)

(c) **[6 points]** List the most common first name for a person, and the count of people that have this first name. (E.g., return ("Mary", 23), if there are 23 actresses named "Mary", and no other first name is that popular). If there is a tie, give only the alphabetically first one. (*Hint:* Check the `limit` keyword.)

Solution:

```
select first_name, count(*) as popularity
from people
group by first_name
order by popularity desc, first_name
limit 1;
```

first_name	*count*
Robert	17

(1 row)

(d) **[6 points]** *Super-movie(s)*: List the `movie` name , and `year`, for the movie that attracted the highest count of nominations in the history of Oscars. If more than one such movie exists, list them all (`movie`, `year`), sorted on movie name.

Solution:

```
select movie, year
from nominations
group by movie, year
having count(*) >= all (
  select count(*) from nominations group by movie
)
order by movie;
```

movie	*year*
All about Eve	1950
Bonnie and Clyde	1967
From Here to Eternity	1953
Mrs. Miniver	1942
Network	1976
On the Waterfront	1954
Peyton Place	1957
The Godfather Part II	1974
Tom Jones	1963

(9 rows)

(e) **[6 points]** List the unique last names of actors nominated for the Leading Actor category between 2000 and 2002, ordered alphabetically.

Solution:

```
select distinct(last_name) from people
join nominations on nominations.person_id = people.id
join categories on categories.id = nominations.category_id
        where categories.name = 'Leading_Actor'
        and nominations.year between 2000 and 2002
order by last_name;
```

last_name
Bardem
Brody
Cage
Caine
Crowe
Day-Lewis
Hanks
Harris
Nicholson
Penn
Rush
Smith
~~Washington~~
Wilkinson

(14 rows)

Grading info:

- -1 for including extra columns
- -1 for missing required columns
- -1 for missing order clause
- -1 for typo that prevented query from running
- -2 for including duplicates
- (c) -3 for returning most nominated first name instead of most common
- (e) -1 for hardcoding category_id
- (e) -2 for using wrong category

Question 2: Oscar Nominations: Part II . . . . . [25 points]

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

Graded by: Jiayu

For this question we will continue looking into Oscar nominations, using the same tables, of Figure 1.

(a) **[5 points]** *Super actors/actresses*: List the first and last names of the 5 people that *won* the most awards, and how many awards each one won. Sort in reverse award count (highest winner, first); break ties alphabetically by first name; then by last name. If more than 5 people qualify, give the first 5 only, using the `limit` SQL keyword, and favoring the alphabetically-smaller names (same rules as for tie-breaking, above).

Solution:

```
select first_name, last_name, count(*) as wins
from nominations join people on person_id = people.id
where won='t'
group by people.id
order by wins desc, first_name, last_name
limit 5;
```

first_name	*last_name*	*wins*
Katharine	Hepburn	4
Ingrid	Bergman	3
Jack	Nicholson	3
Walter	Brennan	3
Anthony	Quinn	2

(5 rows)

- -1 if you did not include the "count" column
- -2 if you did not use "order"
- -2 if you did not apply "limit" at last
- -5 if you did not limit to only the people who had 'won' oscar (logically wrong)

(b) **[10 points]** *The "Susan Lucci" query*:[1] List the first name, last name, and number of nominations of the person who had the most nominations but never won an award. In case of a tie, use `limit`, to return the alphabetically-first person (smallest first name; on tie on that, return the one with the smallest last name).

[1] FYI, Susan Lucci is an actress that was nominated 19 times for Lead Actress in the Emmy Awards, before winning for the first time in 1999. Comic shows made fun of the situation, making her famous.

Solution:

```
-- get a list of ids of all Oscar winners

-- drop the view, if it already exists
drop view if exists winner_ids;

create view winner_ids as (
    select person_id
    from nominations
    where won='t');

select first_name, last_name, count(*) as nom_count
from people as p join nominations as n on p.id=n.person_id
where p.id not in
    (select person_id from winner_ids)
group by p.id
order by nom_count desc, first_name, last_name
limit 1;
```

first_name	*last_name*	*count*
Peter	O'Toole	8

(1 row)

- -1 if you did not include the "count" column
- -2 if you did not use "order"
- -10 if you tried to select from people who have ever not won, instead of excluding people who have ever won - there is a big difference! (logically wrong)

(c) **[10 points]** *Remakes:* A movie is a remake if it has the same title as an older movie. (And, almost always, it has the same plot.) List all cases where both the original movie and its remake were nominated for an award. Specifically, list the name of the movie, the year the original came out, and the year the remake came out, ordered alphabetically. For example, the first (and maybe, the only) remake tuple, is shown in Table 1.

movie	*yearOriginal*	*yearRemake*
A Star Is Born	1937	1954
...	...	...

Table 1: The first tuple of the "remakes" query.

Solution:

```
select distinct  n1.movie, n1.year, n2.year
from nominations as n1, nominations as n2
where n1.movie = n2.movie
    and n1.year < n2.year
order by n1.movie;
```

movie	*yearOriginal*	*yearRemake*
A Star Is Born	1937	1954
Cyrano de Bergerac	1950	1990
Goodbye, Mr. Chips	1939	1969
Henry V	1946	1989
Moulin Rouge	1952	2001
The Hurricane	1937	1999
The Letter	1928	1940
True Grit	1969	2010

(8 rows)

- -1 if you did not use "distinct"
- -2 if you did not use "order"

Question 3: Steelers Season.....[25 points]

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

Graded by: Vinay

For this question you will look into all the games played by the Pittsburgh Steelers in the 2014/2015 season. Here is the schema we will use, shown in Figure 2. The `id` is an integer (1 to n, for each of the n games that the Steelers played).

games	[table]
id	serial not null
	auto-incremented
oponent	varchar
points_scored	int4
points_suffered	int4

Figure 2: Table for Steelers games

List the cumulative balance after each game, ordered by `id`. The balance of a game is the difference between points scored minus points suffered. The *cumulative* balance for game i, is the sum of balances for all games up to and including game i. For example, for the data we provided in the postgres-dump, Figure 3 shows the cumulative balance of Steelers, for their first two games.

id	*opponent*	*cumulativeBalance*
1	Cleveland Browns	3
2	Baltimore Ravens	-17

Figure 3: First lines of correct response.

Hint: Try a θ-join - actually, a θ-self-join.

Solution:

```
select g1.id, g1.opponent, sum(g2.points_scored - g2.points_suffered)
    as cumulativeBalance
from games as g1, games as g2
where g1.id >= g2.id
group by g1.id
order by g1.id;
```

id	*opponent*	*cumulativeBalance*
1	Cleveland Browns	3
2	Baltimore Ravens	-17
3	Carolina Panthers	1
4	Tampa Bay Buccaneers	-2
5	Jacksonville Jaguars	6
6	Cleveland Browns	-15
7	Houston Texans	-8
8	Indianapolis Colts	9
9	Baltimore Ravens	29
10	New York Jets	22
11	Tennessee Titans	25
12	New Orleans Saints	22
13	Cincinnati Bengals	43
14	Atlanta Falcons	50
15	Kansas City Chiefs	58
16	Cincinnati Bengals	68
17	Baltimore Ravens	55

(17 rows)

Question 4: Gerrymandering ..... [25 points]

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

Graded by: Vinay

Preliminaries: Gerrymandering is, according to Wikipedia, a *practice that attempts to establish a political advantage for a particular party or group by manipulating district boundaries to create partisan advantaged districts.*

For this question, we will simplify the problem, and assume that the n cities of this country are on a line, in positions 1, 2, ... n. (The real, 2-d-rectangle version of the problem can also be answered with SQL, but we will not do it here).

In this 1-d setting, you are given a numbered list of cities, with the votes-balance for your favorite party (party 'X') in each city. See Figure 4 for an example with n=5 cities.

id	*name*	*votes_balance*
1	Charlotte	-5
2	Raleigh	10
3	Greensboro	-3
4	Durham	20
5	Winston-Salem	-30

Figure 4: Example table

Your task: Find the range (`start_id`, `end_id`) of cities, to give party 'X' the best voting advantage, that is, the largest sum of votes-balance. Specifically, list the `id` and `name` of the starting and ending city, and the votes-balance for that range.

For example, for the instance of Figure 4, your code should return the table of Figure 5, where *range_votes_balance* = 10-3+20.

`start_id`	`start_name`	`end_id`	`end_name`	`range_votes_balance`
2	Raleigh	4	Durham	27

Figure 5: Response to example table

Break ties in favor of the smallest `start_id` (and, further on, of the smallest `end_id`).

Hint: This is a generalization of the cumulative sum of question #3. Thus, here we need several θ-self-joins.

Solution:

```
drop view if exists RVB_view;

create view RVB_view as (
    select  s.id            as start_id,
            s.name          as start_name,
            e.id                  as end_id,
            e.name                as end_name,
           sum(c.votes_balance) as rvb
    from cities as s, cities as e, cities as c
    where s.id <= c.id and c.id<=e.id
    group by s.id, e.id
);

select  r.start_id, r.start_name, r.end_id, r.end_name,
        r.rvb as range_votes_balance
from RVB_view as r
where r.rvb >= all (
    select max(rvb) from RVB_view)
order by r.start_id, r.end_id;
```

start_id	*start_name*	*end_id*	*end_name*	*range_votes_balance*
4	Durham	19	Wilson	25305

(1 row)