Grant Proposal Embedded SQL Database

CMPT 354 Final Project

Jacob Dauz – 301317900
Gleb Pirogov – 301298139
Chen-Yu (Allen) Wei – 301331695

A. Initial Setup

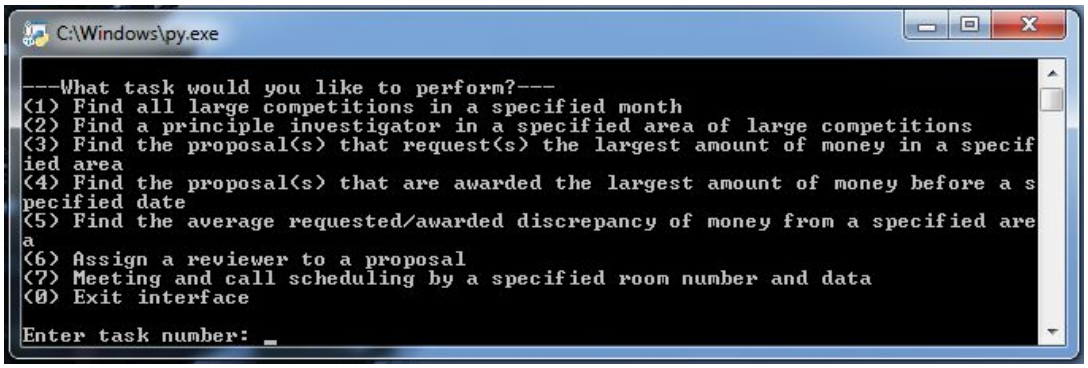
CREATE queries for Postgres are in DauzWeiPirogov-Schema.txt and INSERT queries are in DauzWeiPirogov-Insert.txt. Before continuing, open both files in a text-editor and simply copy-paste the contents into Postgres (the command line should have `postgres=#` at the bottom).

Before running the program with 'python3 DauzWeiPirogov.py', make sure that DauzWeiPirogov-Program.py has the correct port and username-password combination. By default, the values (at the top of the file) are username = 'postgres', password = 'password', host = '127.0.0.1', port = '5432', and database = 'postgres'. These should be edited if they are not the same as those of the computer running the program.

B. Task 2 - Embedded SQL Interface

B1. Explanation

The application window will look like this (see screenshot). The user is able to choose 1 of 7 queries to run, and will be prompted for inputs, such as dates, proposal areas, or room numbers. The results will be outputted below, and the user can either close the program or run another query.



B2. Assumptions

For Q6; conflicts of interest are not assumed to always apply both-ways. Separate entries must exist (e.g. researcher 2 → 3 AND researcher 3 → 2) for 2-way conflicts.

B3. Instructions

B3-1. Query 1

After typing "1" and pressing enter, the user should enter a month. Entering "asdasdasd" or "Febru" will result in an error message, and a prompt to re-enter. Entering "April" or "april" gives the results: `2, 'CS Call 2'` and `4, 'Eng Call 2'`. To confirm that this is the correct output, the user may run the "`select * from call;`" query in Postgres, where they will see that there are 3 non-cancelled proposals in April; CS Call 2, Eng Call 2, and Bio Call 2. Running "`select * from collaborator;`" shows us that only proposals with id 4 and 9 have more than 10 researchers. Running "`select * from proposal;`", we can see that proposal ids 4 and 9 correspond to call ids 2 and 4, and there are no proposals for over $20,000 with callid 6 (Bio Call 2's id). Therefore, the output is correct.

B3-2. Query 2

Inputting "2" for task, "Engineering" for area, and "2" for researcher id, outputs `4, 'Eng Call 2'.` Running "`select * from proposal;`" tells us that proposal with id 9 (previously identified as having 10+ participants) indeed has principle investigator with id 2. Doing the same search but for principle investigator 5 returns nothing (as proposal 10 was for Eng Call 2, but did not request over $20,000 or have 10+ participants. Therefore the output is correct.

B3-3. Query 3

Inputting "3" for task and "Biology" for area outputs `13.` Running "`select * from proposal;`" tells us that proposal with id 13 requested the most money ($40,000). Proposal #13 has call id 5. "`select * from call;`" shows us call #5 is for Biology. Therefore the output is correct.

B3-4. Query 4

Inputting "4" for task and "2020-05-01" for date outputs `6.` Inputting "2020-03-20" for date outputs `11.` Running "`select * from proposal;`", we see that all proposals were submitted prior to 2020-05-01. The largest amount awarded out of all proposals ($25,000) was to proposal #6. So the first output is correct. Before the date 2020-03-20, we can see that the largest amount awarded ($20,000) was to proposal #11. Therefore both outputs are correct.

B3-5. Query 5

Inputting "5" for task and "Engineering" for area outputs `8750.00.` Running "`select * from proposal;`", and remembering that callid 3 and 4 are the only Engineering calls, we see that there were two succesfully awarded proposals. The first received 25000.00 after requesting 22500.00, and the second 15000.00 after requesting 30000.00. We can calculate:

$$\frac{|22500-25000| + |30000-15000|}{2} = 8750$$

Which matches our output; the output is correct.

B3-6. Query 6

Inputting "6" for task and "6" for proposal id outputs a series of researcher ids and names. The id's are: `3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15`. The ids that are missing are 1, 2, and 10. Running "`select * from review;`", we can see that reviewer #10 has 3 unsubmitted reviews, and so cannot take on another one yet. Reviewer #1 is already reviewing proposal #6. "`select * from conflict;`", shows us that reviewer #1 has a conflict of interest with #2, explaining why #2 isn't in the output either. Therefore this first output is correct.

Note that reviewer #3 has a conflict with #4 (and vice versa). Inputting "3" for researcher id and "2020-05-01" for the review submission deadline, we would expect that neither reviewer #3 nor

#4 show up in the initial output anymore. Inputting "q" (to indicate we're done adding reviewers), and then "0", "6", and "6" again, the outputted id's are now: `5, 6, 7, 8, 9, 11, 12, 13, 14, 15`. Just as we expected. We can confirm this by again running "`select * from review;`", which now has a new entry: `13, 6, 3, 2020-05-01, f.` Both the output and input queries are therefore correct.

B3-7. Query 7

Inputting "7" for task, "1" for roomid, and the date "2020-04-05" outputs a conflict error. Room 1 is already booked on 2020-04-05. "2020-04-03" will also output a conflict error. Entering a valid date such as "2020-04-19" moves the user onto the next step. Running "`select * from meeting;`", on Postgres, we can see that the first two dates attempted already have scheduled meetings. Therefore the conflict errors were correct.

The last step involves checking the validity of 3 inputted calls, confirming if calls and reviewers are free to be scheduled on the date we just entered. Inputting calls ids "2", "7" and "5" outputs `Call 2 is possible`, `Call 7 is impossible`, and `Call 5 is possible`. Running "`select * from meeting;`" shows us that a new meeting was created at meeting room 1 on '2020-04-19' with the three calls specified, confirming that the output is correct.

Assumptions: User has to input a valid room which has already been inserted in the 'meeting' table. This prevents users from creating as many meeting rooms as they want through task 7. For testing, only rooms 1 and 2 are valid from the task 1 DDL insert statements.

C. Task 3 - Assertions (on-paper-only)

C1. Explanation

Using the below assertions, we're able to control the data inputted by the users according to the information given. The following assertions are able to create the desired databases and tables we want:

1. Group the natural join of review r and conflict c by the r.proposal and check if c.researcher2 = any of the other researchers that worked on this proposal

2. Natural join on review r and collaborator coll if r.proposal = collab.proposalid and check that the row doesn't exists r.reviewerid = collab.researcherid (review is also a collaborator of the proposal)
3. Natural join meetingreviewer with meeting m where m.id = mr.meetingid. And check that it doesn't exist a review who attends meetings two consecutive days

4. Natural join meeting , meetingcall, meetingreviewer on m.id = mc.meetingid and m.id = mr.meetingid. Group by the date and reviewerid will give us rows of the same reviewerid and date. Then we just check if the total rows >= 1 (it means that the same reviewer have reviewed one or more than one proposal on the same day)

C2. Assumptions

1. (m.date - 1) means yesterday
2. For problem 1, assume that if table conflict has researcher=x and researcher2 =y then it also has researcher1=y and researcher2=x

C3. Answers

C3-1. Assertion 1

```
CREATE ASSERTION conflictinterest
CHECK (NOT EXIST (select * from review r , conflict c where r.reviewerid =
c.researcher1 group by r.proposal having c.researcher2 = any(select
r2.reviewerid from review r2 where r2.proposal = r.proposal));
```

C3-2. Assertion 2

```
CREATE ASSERTION checkreviewers
CHECK (NOT EXIST (SELECT *
                  FROM review r, collaborator coll
                  WHERE r.proposal = collab.proposalid
                        AND r.reivewerid = collab.researcherid)) ;
```

C3-3. Assertion 3

```
CREATE ASSERTION checkreviewers2
CHECK (NOT EXIST (select * from meetingreviewer mr, meeting m where m.id =
mr.meetingid and exist (select * from meetingreviewer mr2, meeting m2
where mr2.reviewerid = mr.reviewerid and m2.id = mr2.meetingid and m2.date
= m.date-1));
```

C3-4. Assertion 4

```
CREATE ASSERTION checkattendees
CHECK ((select count(*) from meeting m,meetingcall mc,meetingreviewer mr
where m.id = mc.meetingid and m.id = mr.meetingid group by
mr.reviewerid,m.date) >= 1);
```