CMPT354 Final Project

Nicolas Ong – 301312417 – 2020-04-20

Task 1: Schema Extension

Here I will list the modifications I made to the given schema, and explain the reasoning behind these decisions.

Attribute Added	In Relation	Reason For Addition	Used In...
reqdamount	proposal	Description mentioned each application has a requested amount.	Q1, Q2, Q3, and Q5
submitted	proposal	Q4 from Task 2 requires knowing the date proposals are submitted.	Q4
awarded	proposal	Description mentioned the application status and amount of money awarded are all for a specific date.	

Relation Added	Reason For Addition	Attributes	Used In...
coreviewer	Description mentioned that “Each reviewer has records about ... who their co-reviewers were for each grant.”	id, reviewid, researcherid	Q6, Q7, A1, A2, A4
meetings	Description mentioned Grant Selection Committees’ meetings.	id, roomid, date	Q7, A3
discussedcalls	Description mentioned that calls are discussed at these meetings.	meetingid, callid	Q7, A4
rsvps	Description mentioned that reviewers participate in these meetings.	meetingid, researcherid	Q7, A3, A4

Task 2: SQL Queries and Embedded SQL Interface

Here I will explain how my program and my queries work. The queries themselves can be found at the top of my program “Ong-Program.py”, or at the bottom of this document.

My Program

I wrote my program, “Ong-Program.py” in Python3 using the pscopg2 library. When using it, you first have to enter credentials to connect to the server. Then you enter a menu where you can select an operation by typing its number. When one is selected, the program runs that operation until it completes, or it hits an error. Both result in a return to the menu.

Setting Up The Environment
To run my program, follow these steps.

1. Clear your PostgreSQL database on CSIL so it has no relations.
2. Run all commands in "Ong-Schema.txt".
3. Run all commands in "Ong-Insert.txt", or insert your own data.
4. Run "`pip install psycopg2-binary --user`" to install psycopg2. The binary means that this library shouldn't be dependent on any external libraries like the standard one is. If psycopg2 is already installed, that's fine.
5. Run "`python3 Ong-Program.py`".

Note that there is no makefile because of how Python works.

Query 1 and Query 2
I combined Query 1 and 2 because they are very similar. Query 2 is essentially Query 1, but results are further refined to user-specified areas and principle investigators. If the user leaves these blank in the program, the Query will assume the user is interested in all areas/principle investigators, and will effectively be Query 1.

This query returns competitions open at a given month. I assume a call with status "closed" is not open at the given month. This is because the date when the call was closed, or when it might open again, are not known. If the call's status is open, I also assume that the call was open on the given date if that date is before its deadline. This is because I don't know when the call was opened, or each time its status was temporarily changed to "closed".

This query gets all the large calls that match the user's criteria by doing the following:

1. Get a table of all proposals with their amount of collaborators.
2. Combine that table with the calls table.
3. Choose rows where the proposal is large, the call is open at the given month, the call is in one of the given areas, and the proposal has one of the given principle investigators.

In addition to leaving areas and principle investigators blank, the user can also leave the month blank. This will tell the query that the user is interested in all months.

Query 3
This query gets the proposals that requested the largest amount of money in a given area by doing the following:

1. Create a table of each proposal with the area of the call it was submitted in.
2. Create a table of the maximum requested amount in each area.
3. Combine those tables on the area and requested amount attributes.
4. Select the rows in the given area.

As a bonus, the user can leave the area blank to get proposals from all areas.

Query 4
This query gets the proposals that were awarded the largest amount of money of the proposals submitted before the given date by doing the following:

1. Create a table of each proposal submitted before the given date.
2. Create a table of the largest award for proposals before the given date.
3. Joins the table on the award amount.

The user can leave the date blank to get all proposals considered.

Query 5
This query gets the average requested/awarded discrepancy of the given area by doing the following:

1. Join call and proposal tables.
2. Select rows with the given area.
3. Group by area.
4. Calculate the average requested/awarded discrepancy for each group.
5. Select the area and calculated average for the given area.

The user can leave the area blank to get the averages for all areas.

Query 6
This query is broken down into 2 queries.

- After the user tells the program the proposal he would like reviewed, the program offers a list of reviewers who are not in conflict with the given proposal. This is query 6A.
- After the user enters a list of reviewers to assign the proposal to, the program runs a query to put this information in the database. This is query 6B.

Query 6A gets a list of reviewers who are not in conflict with the proposal being reviewed by doing the following:

1. Get all researchers in conflict with people on the given proposal.
   a) Join conflict, collaborator, and proposal tables.
   b) Select rows relevant to the given proposal.
2. Get all people on the given proposal.
3. Get all people reviewing >=3 proposals.
   a) Combine all co-reviewers with primary reviewers.
   b) Group by researcher ID.
   c) Count the amount of rows each researcher shows up in.
4. Get the union of the above tables.
5. Get all researchers not in that table.

Note that for both query 6A and query 1&2 I assume that the primary investigator is included in the table of collaborators.

Query 6B is a simple insert query that inserts the given researcher IDs with the given proposal ID into the review table. Since no date is given, I assume that the deadline is in 2 weeks.

Query 7
This query is broken down into 6 queries.

- After the user tells the program the room and date he wants to hold the meeting in, the program checks if the room is available at that date. This is query 7A.
- After the user tells the program the calls he'd like to discuss at the meeting, the program checks if any of the reviewers are not available. This is query 7B.
- If all the reviewers are available, the program runs 4 queries to put everything into the database.
    - Inserts a meeting on the given date in the given room. Query 7Ci.
    - Fetches the meeting ID of the meeting. Query 7Cii.
    - Inserts calls to be discussed in that meeting. Query 7Ciii.
    - RSVPs all relevant reviewers to the meeting. Query 7Civ.

Query 7A checks if the room is available at that date by selecting rows in the meetings table where the given room is booked on the given day. If there are no such rows, the room is free.

Query 7B finds all the reviewers that are unavailable by doing the following:

1. Get a table of all proposals under the given calls.
2. Get a table of all reviewers, co-reviewers and the proposals they're reviewing by adding the coreviewers table to the review table.
3. Combine the above tables to create a table of all the researchers reviewing the proposals under the given calls.
4. Combine that table with the meetings table to find all meetings those reviewers are in.
5. Select all the rows where the meeting is on the same day as the given date.

If query 7B returns no rows, all reviewers are available. Otherwise, there are some that are unavailable.

Queries 7Ci, 7Cii, and 7Ciii are very simple.

Query 7Civ RSVPs all relevant reviewers by finding them in the same way 7B does, and inserting them into the rsvps table.

Task 3: Assertions

The requirements tell us to write assertions, which are standard SQL, but are not implemented in any of the currently available DBMSs. In this section, I will describe how the assertions I wrote work. The actual assertions can be found at the bottom of this document.

Assertion 1
This assertion checks that reviewers do not violate the conflict of interest found in the conflict table. It does this by doing the following:

1. Create a table (reviewerID, researcherID) of reviewers and the researchers in the proposals they're reviewing by doing the following:

a) Get table of researchers and their proposals.
b) Get table of reviewers and their proposals.
c) Combine them on proposal ID.
2. Create a table (researcherID, researcherID) from the conflict table of researchers with conflicts of interest.
3. Choose rows from that table that are in the first table.
4. If no rows exist, then the assertion has not been violated. If rows exist, the assertion has been violated, and the update is rejected.

Assertion 2
This assertion checks that no reviewers are also collaborators on a proposal in the same grant call as the proposal they're reviewing. It does this by doing the following:

1. Create a table (callID, researcherID) with the ID of each researcher that is a collaborator or principle investigator on a proposal in the call.
2. Create a similar table, but for researchers who are reviewing or co-reviewing a proposal in the call.
3. Join them on call ID, and only select the rows where the person reviewing a proposal in a call is also a collaborator in a proposal in that call.
4. If no rows exist, then the assertion has not been violated. If rows exist, then the assertion has been violated, and the update is rejected.

Assertion 3
This assertion checks that no reviewers are assigned to attend more than 1 meeting in 2 consecutive days by doing the following:

1. Create a table (meetingid, meetingid) where the dates are consecutive.
2. Join the rsvps table on itself on researcher IDs.
3. Select the rows with meeting ids that are in the first table.
4. If no rows exist, then the assertion has not been violated. If rows exist, then the assertion has been violated, and the update is rejected.

Assertion 4
This assertion checks that all meeting attendees have reviewed proposals for at least one call being discussed in that meeting. It does this by doing the following:

1. Create a table (researcherid, meetingid, callid) of the meetings each researcher is attending, and each call they can review for that meeting.
2. Create a table (researcherid, callid) of the calls each researcher is reviewing.
3. Left join the two tables above on researcherid and callid, so if the reviewer is not reviewing the call in table 1, the two joined attributes will be NULL.
4. Group this table according to researcherid and meetingid.
5. Count the amount of joined call IDs that are not NULL in each group. This is the amount of calls the researcher is reviewing in each meeting.
6. If no groups exist where this count is 0, then the assertion has not been violated. If such groups exist, then the assertion has been violated, and the update is rejected.

Appendix: Raw SQL

Query 1 and Query 2:

```
SELECT E.id, E.title, E.deadline, E.area, F.reqdamount, F.pi,
F.num_participants
FROM call E JOIN (--the following query creates (callid, reqdamount,
pi, num_participants)
     SELECT callid, reqdamount, pi, num_participants
     FROM proposal C JOIN ( --the following query creates
(proposal_id, num_collaborators)
          SELECT A.id, COUNT(researcherid) num_participants
          FROM proposal A JOIN collaborator B ON A.id=B.proposalid
          GROUP BY A.id
     ) D ON C.id = D.id
) F ON E.id = F.callid
WHERE (F.reqdamount > 20000 OR F.num_participants > 10) AND
((E.area=ANY (%(areas)s) OR %(areas)s IS NULL) AND
(F.pi=ANY (%(pis)s) OR %(pis)s IS NULL) AND
(status='open' AND (%(month)s::date < deadline OR %(month)s IS
NULL)));
```

Query 3:

```
SELECT F.area, E.id, E.reqdamount
FROM ( --gets table with each proposal's reqdamount and area.
     SELECT C.area, D.id, D.reqdamount, D.callid, D.pi, D.status,
D.amount, D.submitted, D.awarded
     FROM call C JOIN proposal D ON C.id=D.callid
) E JOIN ( --gets table with max reqdamount in each area.
     SELECT A.area, MAX(reqdamount)
     FROM call A JOIN proposal B ON A.id=B.callid
     GROUP BY A.area
) F ON E.reqdamount=F.max AND E.area=F.area
WHERE F.area=%(area)s OR %(area)s IS NULL;
```

Query 4:

```
SELECT id, amount, submitted
FROM (--gets all proposals b4 user given date
     SELECT *
     FROM proposal
     WHERE submitted < %(date)s::date OR %(date)s IS NULL
) A JOIN ( --gets max amount awarded from proposals b4 given user
date
     SELECT MAX(amount)
     FROM proposal
     WHERE submitted < %(date)s::date OR %(date)s IS NULL
) B ON A.amount = B.max;
```

Query 5:

```
SELECT A.area, AVG(ABS(reqdamount-amount))
FROM call A JOIN proposal B ON A.id=B.callid
WHERE A.area=%(area)s OR %(area)s IS NULL
GROUP BY A.area;
```

Query 6A:

```
SELECT *
FROM researcher
WHERE id NOT IN (--table of researchers that are in conflict with
given proposal
     --1. people that conflict with people in the proposal:
     SELECT C.researcher2 AS id
     FROM
          proposal A
          JOIN collaborator B ON A.id=B.proposalid
          JOIN ( --union of conflicts.
               SELECT *
               FROM conflict
               UNION
               SELECT id, researcher2 AS researcher1, researcher1 AS
researcher2, reason, expiry
               FROM conflict
          ) C ON B.researcherid=C.researcher1
     WHERE A.id = %(proposal)s
     UNION
     --2. people that are in the proposal
     SELECT D.researcherid AS id
     FROM collaborator D
     WHERE D.proposalid=%(proposal)s
     UNION
     --3. people reviewing >=3 proposals
     SELECT C.id
     FROM (--list of reviewer,proposal
          --all co-reviewers,proposal
          SELECT B.researcherid AS id, A.proposalid AS proposalid
          FROM review A JOIN coreviewer B ON A.id=B.reviewid
          WHERE A.submitted=FALSE
          UNION
          --all primary reviewers, proposal
          SELECT reviewerid AS id, proposalid
          FROM review
          WHERE submitted=FALSE
     ) C
     GROUP BY C.id
     HAVING COUNT(C.id) >= 3
);
```

Query 6B (generated by the program):

```
INSERT INTO review VALUES
(DEFAULT, %(rid0)s, %(proposal)s, now() + interval '2 week', FALSE),
(DEFAULT, %(rid1)s, %(proposal)s, now() + interval '2 week', FALSE),
(DEFAULT, %(rid2)s, %(proposal)s, now() + interval '2 week', FALSE);
```

Query 7A:

```
SELECT A.id, A.roomid, A.date
FROM meetings A
WHERE A.roomid=%(room)s AND A.date=%(date)s::date;
```

Query 7B:

```
SELECT F.researcherid, G.id, G.roomid, G.date
FROM (--table of all meetings the researchers handling proposals
under the given calls are in.
     SELECT D.researcherid, E.meetingid
     FROM (--finds all people reviewing the given calls.
          SELECT DISTINCT C.researcherid
          FROM (--list of all reviewers and their proposals
               --all reviewers reviewing proposals under given calls:
               SELECT B.researcherid AS researcherid, A.proposalid AS
proposalid
               FROM review A JOIN coreviewer B ON A.id=B.reviewid
               UNION
               --all primary reviewers, proposal
               SELECT reviewerid AS researcherid, proposalid
               FROM review
          ) C WHERE C.proposalid IN (--all the proposals under the
given calls
               SELECT B.id
               FROM (
                    SELECT *
                    FROM call
                    WHERE id = ANY (%(calls)s)
               ) A JOIN proposal B ON A.id = B.callid
          )
     ) D JOIN rsvps E ON D.researcherid=E.researcherid
) F JOIN meetings G ON F.meetingid=g.id
WHERE G.date=%(date)s;
```

Query 7Ci:

```
INSERT INTO meetings VALUES
(DEFAULT, %(room)s, %(date)s::date);
```

Query 7Cii:

```
SELECT id
FROM meetings
WHERE roomid=%(room)s AND date=%(date)s::date;
```

Query7Ciii (generated by the program):

```
INSERT INTO discussedcalls VALUES
(%(meeting)s, %(call0)s),
(%(meeting)s, %(call1)s),
(%(meeting)s, %(call2)s);
```

Query7Civ:

```
INSERT INTO rsvps
--all the researchers  reviewing proposals relavent to the calls
being discussed in the new meeting:
SELECT %(meeting)s AS meetingid, D.researcherid
FROM (
     SELECT DISTINCT C.researcherid
     FROM (--list of all reviewers and their proposals
          --all reviewers reviewing proposals under given calls:
          SELECT B.researcherid AS researcherid, A.proposalid AS
proposalid
          FROM review A JOIN coreviewer B ON A.id=B.reviewid
          UNION
          --all primary reviewers, proposal
          SELECT reviewerid AS researcherid, proposalid
          FROM review
     ) C WHERE C.proposalid IN (--all the proposals under the given
calls
          SELECT B.id
          FROM (
               SELECT *
               FROM call
               WHERE id = ANY (%(calls)s)
          ) A JOIN proposal B ON A.id = B.callid
     )
) D;
```

Assertion 1:

```
CREATE ASSERTION reviewer_conflicts
CHECK (
     (SELECT COUNT(*)
     FROM (
          SELECT researcher1, researcher2
          FROM conflict
          UNION
          SELECT researcher2 AS researcher1, researcher1 AS
researcher2
          FROM conflict
     ) G
     WHERE (researcher1, researcher2) IN (--gets (researcher,
reviewer of that researcher).
          SELECT E.researcherid, F.researcherid
          FROM (--the following gets ALL researchers on ALL
proposals. (prop_id, resea_id)
               SELECT A.id AS proposalid, B.researcherid
               FROM proposal A JOIN collaborator B ON
A.id=B.proposalid
          ) E JOIN (--the following gets ALL reviewers on ALL
proposals. (prop_id, rea_id)
               SELECT C.proposalid, D.researcherid
               FROM review C JOIN coreviewer D ON C.id=D.reviewid
               UNION
               SELECT proposalid, reviewerid AS researcherid
               FROM review
          ) F ON E.proposalid=F.proposalid
     )) = 0
);
```

Assertion 2:

```
CREATE ASSERTION reviewer_on_proposal_in_call
CHECK (
     (SELECT COUNT(*)
     FROM (--(callid, researcher of collaborator/pi of a proposal in
that call)
          SELECT A.id AS callid, C.researcherid
          FROM call A
          JOIN proposal B ON A.id=B.callid
          JOIN collaborator C ON B.id=C.proposalid
     ) E JOIN (--(callid, researcher reviewing a proposal in that
call)
          --1. main reviewers
          SELECT A.id AS callid, C.reviewerid AS researcherid
          FROM call A
          JOIN proposal B ON A.id=B.callid
          JOIN review C ON B.id=C.proposalid
          UNION
          --2. coreviewers
          SELECT A.id AS callid, D.researcherid
          FROM call A
          JOIN proposal B ON A.id=B.callid
          JOIN review C ON B.id=C.proposalid
          JOIN coreviewer D ON C.id=D.reviewid
     ) F on E.callid=F.callid
     WHERE E.researcherid=F.researcherid) = 0
);
```

Assertion 3:

```
CREATE ASSERTION reviewers_in_2_consecutive_meetings
CHECK (
     (SELECT COUNT(*)
     FROM rsvps C, rsvps D ON C.researcherid=D.researcherid
     WHERE (C.meetingid, D.meetingid) IN (--table where meetings are
consecutive
          SELECT A.id AS day1id, B.id AS day2id
          FROM meetings A JOIN meetings B ON A.date + integer
'1'=B.date;
     )) = 0
);
```

Assertion 4:

```
CREATE ASSERTION meeting_attendee_review_proposal
CHECK (
     (SELECT COUNT(*)
     FROM (--(researcherid, meetingid, number of calls in the meeting
he reviewed)
          SELECT E.researcherid, E.meetingid, COUNT(F.callid)
          FROM (--(researcherid, a meeting he is in, callid he should
be reviewing)
               SELECT A.researcherid, A.meetingid, B.callid
               FROM rsvps A JOIN discussedcalls B ON
A.meetingid=B.meetingID
          ) E LEFT JOIN (--(researcherid, callid he is reviewing)
               --1. main reviewers
               SELECT C.reviewerid AS researcherid, A.id AS callid
               FROM call A
               JOIN proposal B ON A.id=B.callid
               JOIN review C ON B.id=C.proposalid
               UNION
               --2. coreviewers
               SELECT D.researcherid, A.id AS callid
               FROM call A
               JOIN proposal B ON A.id=B.callid
               JOIN review C ON B.id=C.proposalid
               JOIN coreviewer D ON C.id=D.reviewid
          ) F ON E.researcherid=F.researcherid AND E.callid=F.callid
          GROUP BY E.researcherid, E.meetingid
          HAVING COUNT(F.callid) = 0
     ) G) = 0
);
```