Project Team number: 66

Team members:

Maguire Ong
Kuei Yu Fei
Wang Lifu
Jiahui Liao

Responsibilities

Maguire Ong - Triggers and procedures
Kuei Yu Fei - Functions
Wang Lifu - Triggers and procedures
Jiahui Liao - Functions

Four of us helped each other check the logic flow of each trigger, procedure, and function.

Triggers

1. Name of trigger: check_add_user
   Trigger function: check_add_user
   We trigger when a user is being inserted by checking if the new user’s email is present in the backers or creators table. If it is present, then we return a new row, else we raise an exception. We use the deferment to defer the procedure of creating a user, followed by a backer or creator or both. This ensures that the created user is valid. (Valid here means that there isn’t any user that are neither backers or creators)

   The procedure can be used to avoid this trigger because the procedure is considered to be a single transaction when defer is being used. Therefore, upon creating a valid user using the procedure, this trigger will not be triggered.

2. Name of trigger: check_backs_min_amt
   Trigger function: check_backs_min_amt
   On an insertion to backs, we retrieve the minimum amount of rewards by filtering the rewards table using the reward name and id. We use the minimum amount to compare with the new backed amount, if the new pledged amount is greater than or equal to the minimum amount for the reward level, we accept it, otherwise we raise an exception.

3. Name of trigger: check_projects_reward_level
   Trigger function: check_projects_reward_level
   On an insertion of projects, we retrieve the count of each reward level. Then we check the total count of reward levels. If it is 0, we raise an exception, else we let it pass. Similar to Q1, the deferment function needs to be called as well. This ensures that the add_project procedure can work since we create a procedure, followed by the reward levels, which makes the overall trigger here plausible.

4. Name of trigger: check_refund_request
   Trigger function: check_projects_reward_level
   Upon insertion of refunds, we join the Projects and Backs table by the project id and filter the joined table with the new project id. Then we retrieve the day difference between the requested date and the deadline.
   If the day difference is within 90 days, we create a new refunds entry with accepted=True, else we create a new refunds entry with accepted=False.

5. Name of trigger: check_backs_reward_date
   Trigger function: check_backs_reward_date
   Upon insertion of backs with a certain rewards level, we join the rewards and projects table on project id and filter using the reward name and id. Then we retrieve the project deadline and creation date and ensure that the new backing date is before the deadline of the project and after it has been created. If it is not, we raise an exception.

6. Name of trigger: check_backs_refund_request
   Trigger function: check_backs_refund_request
   Upon an update of backs, we join the projects and backs table on project id and group by the project id to retrieve the total amount backed so far, project funding goal and the project deadline, including 2 conditions whereby the updated back id matches the project id and the total amount backed is greater than or equal to the funding goal.

Additionally, if there are no rows selected, we check that total_amount_backs or funding_goal IS NULL, which will throw an exception that no successful projects are found. Next, we checked that the old request date is NULL and the new request date is NOT NULL as indicated by the question. We also check for the requested deadline to be after the project deadline. If any condition fails, we raise an exception.

Procedures

1. Name of procedure: add_user
   With the provided attributes, we can create a user first.
   When kind is BACKER, we insert a row in Backers.
   when kind is CREATOR, we insert a row in Creators.
   when kind is BOTH, we insert a row in Backers and Creators.
   If kind is not a BACKER, CREATOR or BOTH, we raise an exception and the
   user is not added into any tables.

2. Name of procedure: add_project
   We initialize the n and idx variable to indicate the end of the array and start of the array respectively. We first create the project entry and insert into the projects table. Then we create a loop, then ends when the idx=n+1, otherwise, we retrieve the associated reward name and reward amount,
   then we can check the reward amount is invalid, if it is we raise an exception. Afterwhich, we insert the reward name and amount
   into the rewards table and increment the idx by 1 to continue with the loop.

3. Name of procedure: auto_reject
   We retrieve the backers email, and the day difference between the request date and project deadline. If the day difference is more than 90 days, we reject the request by inserting an entry into the Refunds table with the associated attributes, primarily with the date being today (given) and acceptable=False, indicating rejection.

Functions

1. Name of function: find_superbackers

   Set up a cursor that encapsulates the query, and then read the query result a few rows at a time.

   A successful project must meet two conditions. The first condition is that the total funding amount from backers is greater than or equal to the funding goal. The second condition is that the project deadline must have passed, which means less than or equal to today's date.

   To become a superbacker, all backers need to satisfy one or both conditions.

   Condition1

   Condition 1 is that the backer needs to have backed more than or equal to 5 successful projects and at least 3 successful project types in the last 30 days.

   Condition 2

   If the number of empty requests is greater than or equal to 1 and the date of the refund request is within 30 days, it prints false; otherwise, it prints true.

   If the number of approved refund requests is greater than or equal to 1, it prints false; otherwise, it prints true.

   If the number of rejected refund requests is greater than or equal to 1, it prints false; otherwise, it prints true.

   Overall, the total amount of funding must be more than or equal to $1500, refund requests need to be false, and both approved and rejected requests need to be false.

2. Name of function: find_top_success

A successful project must meet two conditions. The first condition is that the total funding amount from backers is greater than or equal to the funding goal. The second condition is that the project deadline must have passed, which means less than or equal to today's date.

If the ratio between the total amount of funding for the project and the funding goal is larger, the projects will be more successful.

If the ratios are the same, the later the deadline, the more successful the projects.

If the ratios and deadlines are the same, the smaller the id, the more successful the projects.

To check for success, the successful project id must be the same as the project id, the project deadline must be less than or equal to input today's date, and the project type must be equal to the input project type.

We retrieve the project id, project name, creator email, and total amount funded for successful projects.

3. Name of function: find_top_popular
   Set up a cursor that encapsulates the query, and then reads the query results a few rows at a time. Each query is grouped by project id and the dates that it received funding. Filters for the given project type, and checks that the project was created before today.
   Query results: project id, project name, backer email, number of days passed since project creation date up till backing date, total funding received on that particular backing date, and project goal.
   Query results are sorted by the number of days passed in descending order and project id in ascending order.

   A variable was declared to keep track of the total fundings a project receives up till the backing date that was fetched by cursor.
   Declared a counter to keep track of the number of popular projects that were already returned.

Created a loop that terminates either when there are no more results to be fetched, or the counter exceeds n.
Increased the total fundings variable by the amount of funding received on the particular backing date that was fetched by the cursor.
When total fundings of a project exceeds the project goal, return the number of days passed between the creation date and backing date, essentially giving us the number of days it took for the project to reach its funding goal, as well as the project id, project name, and backer email. Increment the counter by 1 if an output was returned.

<u>Difficulties faced</u>

1. Procedures and triggers can be affected by one another. As we started with procedures then moved onto triggers, we realized that creating users or projects using procedures will work only when defer is used. If they were created line by line, it doesn't work.
2. When there's multiple triggers, the sequence of activating the triggers could pose an issue when multiple triggers are created.
3. At times, it is difficult to debug what's going on especially when there's some form of loop since printing explicit statements is not allowed.
4. For function 1, due to many requirements for both condition 1 and condition 2, we cannot use a simple check to filter out the wrong data. The most difficult part is filtering out the invalid refund request, refund accept, and refund reject in condition 2.
5. It is hard to create different test cases to test whether the code is correct because there are so many different situations.
6. For function 2, because the syntax of sql and plpgsql is different, some of the methods cannot be used in sql. For example, in sql, 'p.ptype = ptype' shows that 'column reference 'ptype' is ambiguous'.
7. For function 3, it always outputs an empty row, and n does not limit the number of rows for output. However, the rest of the output has met expectations. The process of debugging is very difficult.

Lessons learned

1. As per point 1 under “Difficulties faced”, the issue here is that our code would not be marked individually. In fact, they will be marked by creating all routines at one go, therefore there could be possible knock-on effects. As a result, we will need to test them together.
2. As per point 2 under “Difficulties faced”, we need to be clear of the activation order when there’s multiple triggers while ensuring that the name of the trigger makes sense.
3. As per point 3 under “Difficulties faced”, we can make use of RAISE NOTICE as a workaround for print statements in PSQL.
4. As per point 6 under “Difficulties faced”, we can change it to ‘p.ptype = $3’.’$3’ represents input 3.
5. As per point 7 under “Difficulties faced”,Due to the wrong sequence of the statements, we can change the sequence for some statements.
6. We have learned how to create our own database to test each of triggers, functions, and procedures.