CS2102 Project (Part 2)

The objective of this part of the team project is for you to apply what you have learned in class to implement triggers and routines for a schema designed for the application mentioned in Part 1. In relation to this, we provide a reference ER diagram in “ER.pdf”, as well as a relation schema in “schema.sql”. Note that the schema has implemented some constraints using foreign keys and checks. You are to implement additional triggers and routines, **using PostgreSQL 14**, according to the requirements detailed in the following.

Note: Please **do not** make any changes to the schema in “schema.sql”. This is because we will evaluate each team’s implementation using automated testing, for which we need each team’s schema to be exactly the same. (For details, you may refer to the FAQ in Section 4.)

1. Constraints to be Enforced using Triggers

Please implement appropriate triggers to enforce the following 14 constraints. For simplicity, you only need to consider INSERT triggers (i.e., triggers that are activated by insertions), and do NOT need to consider DELETE or UPDATE. Whenever an insertion violates a constraint, you should prevent the insertion from taking effect (e.g., by raising an exception in an AFTER trigger), but should not modify the tuple to be inserted. You can assume that you are given a database in a correct state. For each constraint, you are allowed to use multiple triggers to enforce it.

Delivery_requests related:

(1) Each delivery request has at least one package.

Package related:

(2) For each delivery request, the IDs of the packages should be consecutive integers starting from 1.

Unsuccessful_pickups related:

(3) For each delivery request, the IDs of the unsuccessful pickups should be consecutive integers starting from 1.

(4) The timestamp of the first unsuccessful pickup should be after the submission_time of the corresponding delivery request. In addition, each unsuccessful pickup’s timestamp should be after the previous unsuccessful pickup’s timestamp (if any).

Legs related:

(1) For each delivery request, the IDs of the legs should be consecutive integers starting from 1.

(2) For each delivery request, the start time of the first leg should be after the submission_time of the delivery request and the timestamp of the last unsuccessful pickup (if any).

(3) For each delivery request, a new leg cannot be inserted if its start_time is before the end_time of the previous leg, or if the end_time of the previous leg is NULL.

Unsuccessful_deliveries:

(4) The timestamp of each unsuccessful_delivery should be after the start_time of the corresponding leg.

(5) For each delivery request, there can be at most three unsuccessful_deliveries.

Cancelled_requests related:

(6) The cancel_time of a cancelled request should be after the submission_time of the corresponding delivery request.

Return_legs related:

(7) For each delivery request, the first return_leg's ID should 1, the second return_leg's ID should be 2, and so on.

(8) For a delivery request, the first return_leg cannot be inserted if (i) there is no existing leg for the delivery request or (ii) the last existing leg's end_time is after the start_time of the return_leg. In addition, the return_leg's start_time should be after the cancel_time of the request (if any).

(9) For each delivery request, there can be at most three unsuccessful_return_deliveries.

Unsuccessful_return_deliveries related:

(10) The timestamp of each unsuccessful_return_delivery should be after the start_time of the corresponding return_leg.

2. Routines

Routines must be implemented with the same name specified in this document, following the same order of input parameters and returning the exact output parameter type (if any). You may change the names of the input parameters in your implemented routines, but not the data type of the input parameters. If a call to a routine should fail because of database constraint violations, the routine should not return any values. You may raise an exception explicitly, or simply allow it to fail silently. It is important to adhere to these instructions since automated testing will be used to evaluate your implementation. Therefore, the input and output formats of routines have to be fixed.

For all routines, you may assume that the input values are all valid, i.e., you do not need to do validity checks for them in your implementation.

In addition, for arrays, you should follow PostgreSQL's default behaviour of starting the index subscript from 1. You can also use any functions available in PostgreSQL.

2.1 Procedures

The following routines in this section do not have return values, and should be implemented as PostgreSQL procedures (https://www.postgresql.org/docs/14/sql-createprocedure.html).

(1) submit_request(customer_id INTEGER, evaluator_id INTEGER, pickup_addr TEXT, pickup_postal TEXT, recipient_name TEXT, recipient_addr TEXT, recipient_postal TEXT, submission_time TIMESTAMP, package_num INTEGER, reported_height INTEGER[], reported_width INTEGER[], reported_depth INTEGER[], reported_weight INTEGER[], content TEXT[], estimated_value NUMERIC[])

- Submit a delivery request from a customer
- package_num refers to the number of packages pertinent to the delivery request
- The status of the delivery request should be set to "submitted"
- The pickup_date, num_days_needed, and price of the delivery request should be set to NULL
- NUMERIC[] refers to an array of NUMERIC; TEXT[] refers to an array of TEXT

- reported_height[i] refers to the reported_height of the i-th package of the delivery request; similarly for reported_width[i], reported_depth[i], reported_weight[i], content[i], and estimated_value[i]
- the actual_height, actual_width, actual_depth, actual_weight of each package should be NULL

(2) resubmit_request(request_id INTEGER, evaluator_id INTEGER, submission_time TIMESTAMP, reported_height INTEGER[], reported_width INTEGER[], reported_depth INTEGER[], reported_weight INTEGER[])

- Resubmit an existing delivery request (whose id equals request_id in the input parameter) as a new request when the actual measurements of the packages are found to be significantly different from the reported measurements during a delivery pickup.
- The attributes of the new request are the same as those of the original one, except that:
    - the id of the new request is new
    - the evaluator_id of the new request is set to the evaluator_id in the input parameters
    - the status of the new request is set to “submitted”
    - the submission_time of the new request is set to the submission_time in the input parameters of the procedure.
    - the pickup_date, num_days_needed, and price of the new request are set to NULL
- The packages of the new request are the same as the packages in the original request, except that:
    - the request_id of each package is the id of the new request
    - the reported_height of the i-th package equals reported_height[i] in the input parameters; similarly for reported_width[i], reported_depth[i], and reported_weight[i]
    - the actual_height, actual_width, actual_depth, and actual_weight of each package are set to NULL

(3) insert_leg(request_id INTEGER, handler_id INTEGER, start_time TIMESTAMP, destination_facility INTEGER)

- Insert a leg for the delivery request whose ID equals request_id in the input parameters
- The function should set the leg_id of the leg properly
- In the input parameters, handler_id refers to handler_id of the leg; similarly for start_time and destination_facility
- The end_time of the leg should be set to NULL

2.2 Functions

The following routines in this section have return values, and should be implemented as PostgreSQL functions (https://www.postgresql.org/docs/14/sql-createfunction.html).

(1) view_trajectory(request_id INTEGER)

- Output: TABLE (source_addr TEXT, destination_addr TEXT, start_time TIMESTAMP, end_time TIMESTAMP)
- Retrieves the legs and return_legs (if any) of a delivery request, and outputs the addresses of the starting point and ending point of each leg and return_leg, as well as their start_time and end_time, in ascending order of start_time
- request_id refers to the ID of the delivery request
- Please refer to Project (Part 1) for the definitions of the first and last legs of a delivery process, as well as the definitions of the first and last legs of a return delivery process.
- For simplicity, we assume that for the given delivery request, either the packages have been successfully delivered to the recipient, or the packages have been successfully returned to the customer.
- We omit the case when there are three unsuccessful deliveries in the **return** delivery process.

(2) get_top_delivery_persons (k INTEGER)

- Output: TABLE (employee_id INTEGER)
- Retrieves the IDs of the top k delivery persons that have handled the largest numbers of "trips", in descending order of the numbers of trips
  - A trip is defined as a leg, or return delivery leg, or an unsuccessful pickup
  - If two delivery persons have the same number of trips, then the one with the smaller id should rank before the other

(3) get_top_connections(k INTEGER)

- Output: TABLE(source_facility_id INTEGER, destination_facility_id INTEGER)
- Finds the top k "connections" that have the largest numbers of total occurrences in the legs and return_legs tables
  - A connection is defined as a pair of source_facility and destination_facility. (We do not consider NULL source_facility or NULL destiatnion_facility.)
  - Each time a leg or a return leg starts from a source_facility S and ends at a destination_facility T, we say that the connection (S, T) "occurs" once
  - In the output, you should rank the connections in descending order of the numbers of occurrences; if two connections have the same number of total occurrences, then they should be sorted in ascending order of (source_facility_id, destination_facility_id)

3. Deadline and Deliverables

Each team is to upload a zip file named **teamNNN.zip**, where NNN is the 3-digits team number, to the Canvas assignment named "**Triggers and Routines**". You should add leading zeroes to your team number (e.g., team005.zip). Only one file is to be submitted per group. If there are multiple submissions for a group, only the latest submission will be chosen. If the latest submission is uploaded after the deadline, the penalty applies.

The submission **teamNNN.zip** should contain the following two files:

- report.pdf: project report in pdf format
- proc.sql: the triggers and routines of your implementation

The project report (up to a maximum of 20 pages in pdf format with at least 10-point font size) should include the following contents:

- Names and student numbers of all team members and project team number (on the first page).
- A listing of the Project (Part 2) responsibilities of each team member.
  - Team members who did not contribute a fair share of the project work will not receive the same marks awarded to the team.
- For each trigger:
  - Provide the name of the trigger.
  - Explain the basic idea of the trigger implementation.
- For each routine:
  - Explain the basic idea of the routine implementation.
- A summary of any difficulties encountered and lessons learned from the project.

For late submissions, 2 marks (out of 10) will be deducted for submissions up to one day's late; submissions late for more than one day will receive zero marks and will not be graded.

4. FAQ

(1) "Why are we not allowed to make any changes to the schema provided or the routine names and parameters given?"

- This is because we will evaluate each submission using automatic testing. That is, we will prepare some test data following the schema provided to you, and then combine it with your proc.sql and run some queries. We will examine the database state after each query to check whether it is correct; if it is, then you get some marks associated with the query. To facilitate such automatic testing, it is important that every team's implementation is based on exactly the same schema, and uses the exact routine interface that we provided.

(2) "Will the test data be provided?"

- Unfortunately, no. But you may generate your own test data to check your implementation.