CS2102 Project Part 2

**Team 051**

|  |  |  |
| --- | --- | --- |
| Name | Student no. | Responsibilities |
| Liu Liu Zhengdao |  | - wrote the code for section 1  - wrote comments for some of the triggers in section 1 |
| Sean Tay | A0200112M | - wrote the routines and report comments for section 2 up to 2.2.2  - came up with test data to test section 2 |
| Jian Fanmin | A0199359E |
| Herman Thong Wen Han | A0200577J | - wrote the routine for section 2.2.3  - went through the code in section 1 and wrote the remaining comments for sections 1 and 2 |

# Triggers

## trigger\_one\_check:

Ensures that you have to make a delivery\_request before you add packages so it wouldn't make sense to check delivery requests when they're inserted since the check is done after a package is inserted, fresh delivery requests that the package's request id references should survive.

## trigger\_two\_check:

Checks the highest previous package id with the same request id then checks if the new addition has a consecutive package id. Also includes edge case where table is empty.

## trigger\_three\_and\_four\_check:

Trigger #3 is same as trigger #2. Trigger #4 selects the appropriate timestamp and prevents insertion of earlier timestamps.

## legs\_one\_two\_and\_three\_check:

Same as trigger #2.

## unsuccess\_delivery\_one\_check:

Before inserting on unsuccessful\_deliveries, raise an exception is new timestamp is earlier than leg's start time.

## unsuccess\_delivery\_two\_check:

Before inserting on unsuccessful\_deliveries, raise an exception if there have already been 3 unsuccessful deliveries.

## cancel\_one\_check:

Before inserting on, cancelled\_requests, raise an exception if cancel timestamp is earlier thtan request submission time.

## return\_leg\_one\_check:

Before inserting on return\_legs, raise an exception if the table is empty and return leg ID is 1, or if the table is not empty and the return leg ID is not consecutive.

## return\_leg\_two\_check:

Before inserting on return\_legs, raise an exception if: 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 or (iii) the return\_leg’s start\_time should be after the cancel\_time of the request (if any).

## return\_leg\_three\_check:

Before inserting on unsuccessful\_return\_deliveries, raise an exception if there have already been 3 unsuccessful return deliveries.

## unsuccess\_return\_one\_check:

Before inserting on unsuccessful\_return\_deliveries, raise an exception if return attempt timestamp is earlier than start of return leg.

# Routines

## submit\_request:

The procedure submit\_request takes input parameters like customer\_id, evaluator\_id, pickup\_addr, pickup\_postal, etc. as mentioned in the problem statement.

A local variable request\_id is declared to store the ID of the newly inserted delivery request.

The delivery\_requests table is inserted with the input values and the status is set to submitted. The RETURNING clause is used to get the id of the newly inserted row into request\_id.

A loop is used to insert each package's details into the packages table. The ith item of each input array is used to insert into the corresponding column in the table.

## resubmit\_request:

The procedure first creates a new delivery request with the same attributes as the original one, except for the evaluator\_id, status, and submission\_time. It then copies the packages from the original delivery request to the new one, with the reported measurements replaced by the ones provided as input and the actual measurements set to NULL.

## insert\_leg:

The procedure inserts a new row into the legs table with the given request\_id, handler\_id, start\_time, destination\_facility. The leg\_id is set to the maximum existing leg\_id for the request\_id plus one, which ensures that the leg\_id is unique. The end\_time is set to NULL by default.

## view\_trajectory:

The function returns a table that contains the trajectory of a delivery request, including the pickup address, delivery address, start time, and end time for each leg of the delivery. The function takes a single input parameter req\_id, which is the ID of the delivery request to be queried. It uses a CTE to combine information from the accepted\_requests, delivery\_requests, legs, cancelled\_or\_unsuccessful\_requests, return\_legs, and facilities tables. The resulting table is ordered by start time in ascending order.

## get\_top\_delivery\_persons:

The function joins the legs, delivery\_staff, and accepted\_requests tables, grouping by delivery person ID, and returns the top k results based on the count of their ID.

## get\_top\_connections(k INTEGER):

???? wtf???

# Reflections

During the project of writing SQL triggers and functions to enforce the given constraints, some difficulties were encountered such as handling complex constraints and identifying all the possible edge cases for testing.

Some constraints were straightforward to enforce, while others required more complex SQL queries, logic, and structures, like CTEs. It particular, writing the function view\_trajectory required handling many SQL queries at once.

In addition to handling complex constraints, another challenge was identifying all the possible edge cases for testing. It is important to have a comprehensive test set that covers all possible scenarios and edge cases to ensure that the SQL code works as expected and all constraints are properly enforced. One useful test case we often forgot is to add NULL values to see if the triggers can properly detect them.