INSTITUTO SUPERIOR TÉCNICO

Database Project, Part 3

Turno: SIBD-2PB08

Grupo: 21

número	nome	contribuição (%)	horas gastas
93190	Tiago Mendes	34	40
94037	Keval Visnú Ramniclal	33	20
94196	Gonçalo Mesquita	33	30

1 Populate

To populate the database, it is needed to run the *populate.sql* script. Some of the actions of data insertion are contained within a transaction method, with deferred triggers . This is due to table dependencies, for example, when inserting a new sailor in the Sailor table, it is also needed to insert that record in the Junior or Senior tables.

2 Integrity Constraints

Integrity constraints are responsible for maintaining a coherent data. The following integrity constraints were used:

- (IC-1) Every Sailor is either Senior or Junior.
- (IC-2) The take-off and arrival dates of trips for the same reservation must not overlap (i.e., one trip cannot take off before the arrival of another).

2.1 IC-1

To verify the first integrity constraint, it was created a function called **check_sailor()** and a trigger **check_sailor_tg**. The function check_sailor() checks for any record which might break the integrity constraint by being in the junior and senior tables simultaneously or by not being in none of them. The trigger uses this function after an insertion is made to the sailor table.

2.2 IC-2

To guarantee that the take-off and arrival of trips for the same reservation do not overlap, a function called **check_dates()** was created and is called before insertions on the trip table. The function check_dates() will verify if the take_off and arrival dates of the new trip record to be inserted are between any take_off and arrival dates of the rest of the trip records with the same reservation.

3 Web Application

A Web Application was implemented to simulate a client and database interaction. This web application was developed in python and served as an interface to many tasks such as creation, deletion and listing of entities. These entities could be sailors, trips or reservations. The Web app is accessible at `https://web2.tecnico.ulisboa.pt/ist193190/home.cgi`. In the first web page,`home.cgi`, it is displayed several options as seen in figure 1.

Figura 1: home.cgi main page

After choosing one, the client is redirected to the corresponding web page and there is able to choose actions based on the entity type. For the **Sailor** and **Trip** entities, it is possible to List, Create and Remove elements. On the other hand, for the **Reservation** entity, it is also possible to authorize or remove authorization of sailors from a reservation.

To list all elements of a given table, a query is made to the corresponding database in `list.cgi`, following image 2. Which is done by selecting all columns and entries of a table, as seen in 3.

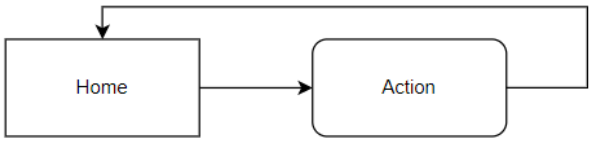

Figura 2: Flow of Communications between the home page and the web page for the List action.

To create or delete table entries, the flow of communication is as in image 4. It is required to fill a form in order to correctly choose the entry to delete.

When inserting a new sailor, the same entry is needed to be inserted in the sub tables junior or senior. This is done as seen in figure 5. By using the command `START TRANSACTION;` it is possible to combine both insert

```
# Making query
sql = 'SELECT * FROM {};'.format(mode)
```

Figura 3: Query to list all entries in said table.

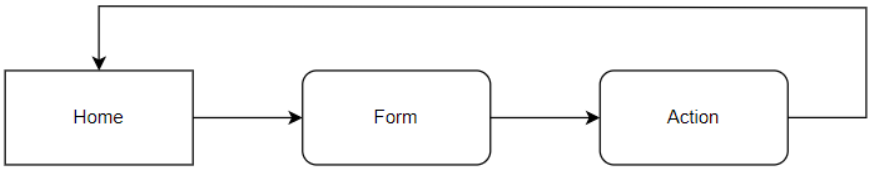

Figura 4: Flow of Communications between the home server and the web page for the Create and Delete actions.

commands in one, making them an atomic operation. Hence guaranteeing the atomicity of related operations.

```
# Creating connection
connection = psycopg2.connect(login.credentials)
cursor = connection.cursor()

verify_sailor = "SELECT * FROM sailor WHERE email = %(email)s"

cursor.execute(verify_sailor, {'email': email})
result = cursor.fetchall()

if len(result) == 0:
    cursor.execute("START TRANSACTION;")
    cursor.execute("SET CONSTRAINTS ALL DEFERRED;")

    insert_sailor = "INSERT INTO sailor VALUES(%(firstname)s, %(surname)s, %(email)s)"
    cursor.execute(insert_sailor, {'firstname': firstname,'surname' : surname, 'email': email})

    insert_sailor = "INSERT INTO {} VALUES(%(email)s)".format(rank)
    cursor.execute(insert_sailor, {'email': email})

    cursor.execute("COMMIT;")
    connection.commit()
    print('Create New Sailor: SUCCESS')
```

Figura 5: Inserting Sailor Operations.

At last, to delete entries from tables with dependencies, it was implemented a series of triggers. These triggers would remove records in the correct

order from all related tables.

4 Queries

The code for the queries needed for the SQL exercise can be found in the *queries.sql* script. There's also a file called *analytics.sql* which contains the script for the queries that allow to analyze the total number of trips according to different groups. The file *output.txt* contains the results of both queries exercises.

5 Indexes

The purpose of the indexes is to enhance the search of records without going through the whole table.

5.1 Query 7.1

For the first query, corresponding to exercise 7.1 of the assignment, there is a range condition `"WHERE year >= <some year>"` and an equality condition `"boat_class = <some class>"`. For this type of query, it is advised to use composite B+Tree indexes on both attributes, year and boat_class, since this type of index is efficient in both range and point queries. Hash indexes can be very effective in equality conditions since records can be found at distance 1. However, they are very ineffective in range conditions. Therefore, the possibility of creating a hash index for the boat_class attribute fails since its effectiveness depends on the number of distinct boat classes and the number of records mapped to the same bucket. When using this hash function, the lower number of records per bucket the better, since it would be faster to make a sequential search at the end.

5.2 Query 7.2

For the query in exercise 7.2 we have an aggregation function, "COUNT(*)", which can be effectively grouped by an B+Tree index on the boat_country attribute. Since the query is also selecting this attribute it would be an index-only-scan since the data needed is already available in the index file.