Instituto Superior Técnico

Master Degree in Electrical and Computer Engineering

Information Systems and Databases

SIBD Project - Part III

Group 03

Authors:	IST Number:	Effort in Hours:
André Silva	90015	25 Hours
João Pires	103799	25 Hours
Nuno Martins	103783	25 Hours

Total Effort in Hours: 75 Hours

Laboratory Teacher: Associate Professor Bruno Martins

Laboratory Shift: PB06

30th January, 2022

The third assignment consists of the development of SQL queries, integrity constraints and the creation of a web application prototype and OLAP queries.

1 Database Loading

We created our database using the supplied schema-part3.sql file making a few adjustments over some variable names. After running the file the result is all tables created without any record. Then, the database was populated with the records required to ensure that all SQL queries from section 3 have a non-empty result. To populate the database we created a file 'populate.sql' which after running adds records to previously created tables.

2 Integrity Constraints

In the second task we wrote the code to implement the integrity constrains using stored procedures and triggers. The ICs are implemented in the file 'ICs.sql'. The integrity constraints file must run before the populate.sql file to make sure that there are no violations in the populating the tables. Regarding the integrity constrains implementation:

IC-1: Two reservations for the same boat can not have their corresponding date intervals intersecting.
We created a trigger that selects into a variable all tries to find cases where the new start and end dates for the same set (cni, boat country, sailor id, sailor country) intersect. If one case is found then an exception is raised and the new reservation is not created.

IC-2: Any location ust be specialized in one of three - disjoint - entities: marina, port or wharf.
To implement this IC we created four triggers: one is used to make sure that the specialization is mandatory and it only allows to insert a location if the coordinates (latitude,longitude) match the coordinates of one marina, wharf or port. The other 3 triggers are used to guarantee that entities are disjoint. For example, if a wharf is inserted, the trigger verifies if those coordinate values exist in port or marina tables.

IC-3: A country where a boat is registered must correspond - at least - to one location.
This IC was implemented through a trigger that when inserting or updating a boat raise an exception if the boat country does not match with any location country.

3 SQL

The implemented SQL queries to answer the proposed questions are in the file 'queries.sql'.

4 View

To have a view summarizing the most important information regarding boat trips, we created three views to get different information and combined them into one that returns the desired information about each trip. The SQL statements to create the desired view are in the file 'view.sql'.

5 Application Development

We developed a web application using python cgi scripts and html pages with some functionalities. The web application can be accessed trough the URL: http://web2.tecnico.ulisboa.pt/ist1103783/main.cgi.

There are some interpretations that could be made and that are not fully expressed in the handout. Thus, we will explain what each functionality does and which pages are used. All functionalities can be chosen from the main.cgi page which is used as a menu/central page. We used a scheme of two cgi pages to all input/output operations, using one to collect information trough forms and the other to execute the changes in database and retrieve success or error messages. We also added some tiny details like show the available schedule when creating a reservation or the avaible countries to register an owner/sailor/boat.

Register and remove owners: owner_register_info.cgi, owner_register.cgi, owner_remove_info.cgi, owner_remove.cgi. When an owner is registered, if he is a sailor the data is inserted only in the owner table and if he is not the data is inserted both in person and owner tables. To remove an owner we remove all trips and reservations related with his boat and the boat, only removing the owner after that.

Register and remove boats: boat_ register_ info.cgi, boat_ register.cgi, boat_ remove_ info.cgi, boat_ remove.cgi. When register a boat the values collected in the form are inserted into the table boat. We also placed a checkbox to confirm if the boat has a fixed vhf inserting the data also in the table boat_vhf in those cases. To remove a boat, we remove all related trips and reservation first and, after that, we remove the boat from boat table (and boat_vhf if applicable).

Register, remove and list sailors: sailor_register_info.cgi, sailor_register.cgi, sailor_remove_info.cgi, sailor_remove.cgi, sailor_list.cgi. To register an sailor, if he is a owner the data is inserted only in the sailor table and if he is not the data is inserted both in person and sailor tables. To remove a sailor, if he is not owner we remove all trips and reservations related with him and remove from both sailor and person tables; if the sailor is also an owner, we removed his trips and reservations but remove only from sailors table. We also added a list of registered sailors and available countries in the database to make it easier to register or remove someone. Finally, to list the sailors, the main.cgi calls the sailor_list.cgi file which executes a simple query to list all sailors.

Create and remove reservations: reservation_register_info.cgi, reservation_register.cgi, reservation_remove_info.cgi, reservation_remove.cgi, sailor_list.cgi. When a reservation is created the values are inserted into reservation table. To remove a reservation, we first removed the trips associated with that reservation and the reservation itself after that.

To guarantee the atomicity of related operations in the database we always use auto-commit = False and made the transactions manually executing SQL commands "START TRANSACTION;" and "COMMIT";.

In order to prevent attacks by SQL injection and prize security, our SQL queries executed in cgi files were written with safe named parameters and only allow some errors to be shown (like trying to remove a non-existing person) and making all the other error not showing up with details.

All files are available in the web folder.

6 Data Analytic Queries

Using the view we created before we wrote two SQL queries (file 'analytics.sql') to analyze the total number of trips according to different groups depending on the start date and location of origin. To solve the question 6.1 we used the union of group by clauses and, for the 6.2, a cube instruction.

Answering this question allowed us to understand better the importance of cubes and higher dimensions to analyze the information we have in our database. With a single SQL instruction we have all the information organized!

7 Indexes

We included in the files the 'indexes.sql' file with the SQL queries we used to create our indexes and make several different tests.

7.1

In the first query, since our table boat is not ordered by iso_code, a table scan is costly, and we need an index to optimize the query. A composite B+Tree index over the attributes (boat_iso_code,year) on the table boat would be best since B+Tree indexes optimize both range and point queries (while Hash tables don't support range queries). The condition boat_iso_code = country.iso_code is more selective in our table than the range condition wich decreases the I/O needed.This also means that our index has a faster planning and execution time than the composite index (year,boat_iso_code).

Using the 'EXPLAIN ANALYZE' SQL query we obtained the following results:

```
QUERY PLAN
Nested Loop  (cost=0.14..9.30 rows=1 width=78) (actual time=0.039..0.067 rows=3 loops=1)
  Join Filter: (boat.boat_iso_code = country.iso_code)
  Rows Removed by Join Filter: 5
  ->  Index Scan using country_country_name_key on country  (cost=0.14..8.16 rows=1 width=12) (actual time=0.022..0.025 rows=1 loops=1)
        Index Cond: ((country_name)::text = 'Portugal'::text)
  ->  Seq Scan on boat  (cost=0.00..1.10 rows=3 width=90) (actual time=0.010..0.020 rows=8 loops=1)
        Filter: (year >= 2000)
Planning Time: 0.267 ms
Execution Time: 0.108 ms
```

Figura 1: *Query plan using composite index on (boat_iso_code,year)*

7.2

About the second query, we didn't know how to optimize the LIKE condition, but the range condition can be optimized with a B+Tree index over the attributes (start_date) on the table trip.