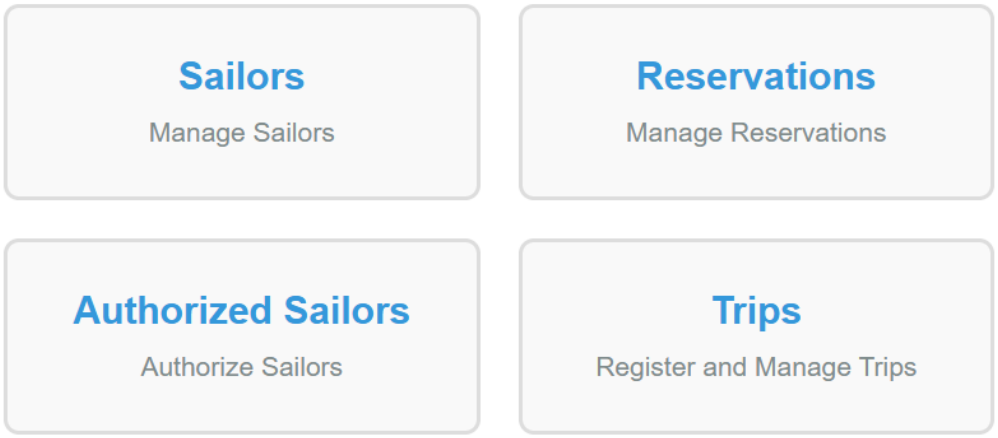
TÉCNICO LISBOA

Project DataBase – Part 2 – Group 7

Boating Management System



Student Name	IST-ID	Contribution (%)	Effort (Hours)
Francisco Silva	103563	33.3%	21
Bárbara Morais	113250	33.3%	21
João Andrade	100477	33.3%	21

Laboratory Teacher : Prof. Francisco Regateiro

Information Systems and Databases (SIBD)

Biomedical Engineering

January 2026

1. Introduction

This report presents the second part of the Database Project.

The project consists of implementing a Boating Management System with:

- Advanced SQL queries
- Integrity constraints using triggers and stored procedures
- A web application prototype using Python CGI

The system manages boats, sailors, reservations, authorizations, and trips, ensuring data integrity through constraints and providing a user-friendly interface for operations.

Online Application:

https://web.tecnico.ulisboa.pt/ist1113250/menu.cgi

2. SQL Queries

Query 1 - Country with Most Boats

Explanation: This query identifies the country that has the largest number of registered boats. It works by grouping all boats by country and counting them. The HAVING COUNT(*) >= ALL (...) condition ensures that only the country (or countries) whose boat count is greater than or equal to all other countries is returned. This effectively selects the country with the maximum number of boats.

Query 2: Sailors with At Least Two Certificates

Explanation: This query retrieves all sailors who possess two or more sailing certificates. It joins the sailor table with sailing_certificate on the sailor's email, groups the results by each sailor, and filters groups where the count of certificates is at least 2. This ensures that only sailors with multiple certifications are listed.

Query 3: Sailors Who Sailed to Every Location in Portugal

Explanation: This query uses a relational division pattern to find sailors who have visited every location in Portugal.

The inner NOT EXISTS checks for each location in Portugal whether the sailor has a trip ending there.

The outer NOT EXISTS ensures there are no locations in Portugal that the sailor has not visited. As a result, only sailors who have visited all Portuguese locations are returned.

Query 4 - Sailors with the Most Skipped Trips

Explanation : This query identifies the sailor (or sailors) who has completed the highest number of trips. It joins the sailor table with trip, groups by each sailor, and counts the trips. The HAVING COUNT(*) >= ALL (...) condition ensures only those with the maximum trip count are returned.

Query 5 - Sailors with the Longest Total Trip Duration per Reservation

Explanation : This query calculates the total sailing duration for each sailor per reservation.

SUM(t.arrival - t.takeoff) computes the total duration of trips for each Sailor-Reservation combination.

The HAVING clause with >= ALL (...) compares this sum against all other Sailor-Reservation sums, returning the pair(s) with the maximum total duration. This helps identify which sailors were most active during a specific reservation.

3. Integrity Constraints

IC-1 – Every sailor is either junior or senior

Constraint: Every sailor must belong to exactly one specialisation, junior or senior (mandatory and disjoint specialisation).

Implementation:

- **Function check_sailor_disjoint()**

Used by two BEFORE INSERT OR UPDATE triggers on tables junior and senior.

If an email being inserted into senior already exists in junior, the function raises an exception and the operation is rejected.

- **Function check_sailor_mandatory()**

Used by a CONSTRAINT TRIGGER AFTER INSERT on table sailor, declared as DEFERRABLE.

At commit time it checks whether the new sailor.email exists in junior or in senior. Otherwise it raises an exception and the transaction is rolled back.

This enforces that a sailor cannot appear in both specialisation tables (*disjoint*) and cannot exist in sailor without being classified as junior or senior (*mandatory participation*).

IC-2 – Non-overlapping trips for the same reservation

Constraint: For the same reservation, trip intervals must not overlap.

Implementation:

- **Function check_trip_overlap()**

Searches table trip for another row with the same reservation key that satisfies the overlap condition:
takeoff < NEW.arrival AND arrival > NEW.takeoff

It uses IS DISTINCT FROM on the primary key to avoid comparing the new row with itself during updates.

- **Trigger trig_check_trip_overlap**

A BEFORE INSERT OR UPDATE trigger on trip that calls check_trip_overlap().

If any overlapping trip is found for that reservation, the function raises an exception and the insertion or update is rejected.

This guarantees that, for each reservation, trips are temporally consistent and no two trips are active at the same time for the same boat and reservation interval.

4. View: trip_info

The trip_info view consolidates information about boat trips by combining data from multiple related tables. It provides a denormalized view that simplifies queries about trips by joining trip data with location details, country information (including ISO codes), and boat registration data.

Explanation:

- Origin coordinates in trip are joined to location (loc_from) and then to country (co_origin) to obtain origin location name and country (name and ISO code).
- Destination coordinates are joined to another location alias (loc_to) and to country (co_dest) for destination information.
- The boat is joined via (boat_country, cni) to table boat, and then to country (co_boat) to obtain the boat's registration country (name and ISO code).

This view provides a single, convenient relation with origin, destination, boat and start date, so application or reporting queries do not need to repeat all the joins.

5. Web Application

The web application was developed following a three-tier architecture:

1. Presentation Layer: HTML generated by Python CGI scripts

2. Application Layer: Python CGI scripts with business logic

3. Data Layer: PostgreSQL database

Access: http://web.tecnico.ulisboa.pt/ist1113250/menu.cgi

File Structure

web/

├── db.py # Database connection module

├── menu.cgi # Main menu/homepage

├── sailors.cgi # Sailor management

├── reservations.cgi # Reservation management

├── authorized.cgi # Authorization management

└── trips.cgi # Trip management

db.py

Central module for database connections. Stores credentials and provides

`get_connection()` function used by all CGI scripts.

```python
def get_connection():
  dsn = f"host={HOST} port={PORT} user={IST_ID}
      password={PASSWORD} dbname={DB_NAME}"
  return psycopg2.connect(dsn)
```

menu.cgi

Entry point showing links to all modules.

sailors.cgi

Features:

- List all sailors (junior/senior)

- Create new sailor
- Delete sailor
- Uses LEFT JOINs to determine sailor type

reservations.cgi

Features:

- List all reservations with boat info
- Create reservation (inserts into date_interval first)
- Delete reservation (cascades to authorised table)
- Dropdown selects for boats and senior sailors

authorized.cgi

Features:

- List authorizations (sailor + reservation)
- Authorize sailor for reservation
- Deauthorize sailor
- JavaScript to populate hidden fields from dropdown

trips.cgi

Features:

- List all trips
- Register new trip
- Delete trip
- Location dropdowns for origin/destination

Security Measures

1. SQL Injection Prevention and XSS attacks

All queries use parameterized queries with psycopg2:

```python
cur.execute("INSERT INTO sailor(email, firstname, surname)
        VALUES (%s, %s, %s)", (email, firstname, surname))
```

Never concatenate user input directly into SQL strings.

All user-generated content is escaped using html.escape() before being displayed in HTML pages, preventing Cross-Site Scripting attacks.

2. Transaction Management

All operations use transactions:

```python
conn.autocommit = False
try:
  # operations
  conn.commit()
except:
  conn.rollback()
```

Ensures atomicity either all operations succeed or none do.

3. Error Handling

Try-catch blocks handle exceptions gracefully, showing user-friendly messages.