TÉCNICO LISBOA

INSTITUTO SUPERIOR TÉCNICO

SIBD Project - Part 3

90071 – Francisco Nogueira (33% - 13h)
90144 – Miguel Figueira (33% - 13h)
90210 – Alexandre Fonseca (33% - 13h)

Grupo 14
Turno segunda-feira, 11h00
Docente: Francisco Regateiro

15 de dezembro de 2020

1 Database Loading

The database loading SQL commands are contained in the populate.sql script.

2 Integrity Constraints

The integrity constraints (IC1), (IC2) and (IC5) are implement as SQL commands in the RI.sql script. In order to implement these ICs only trigger functions were used.

3 View, SQL and Indexes

The file ”queries.sql” contains the View asked in the project sheet.

The file ”queries.sql” also contains all the queries asked in the project sheet. For query number 4, it was used the View created in exercise 3.

For exercise 6.1 a B+Tree index must be created for the attribute ”locality” from table ”substation”and a hash index must be created for the attribute ”pv” from table ”transformer”. The first index is necessary in order to help the group by operation, and the second one is also necessary since hash indexes are more efficient for queries which require equality tests like the one in 6.1.

In exercise 6.2 a B+Tree index must be created for the attribute ”instant” from table ”incident”, because B+Tree indexes are more efficient for range queries like the one in this exercise. The attribute ”instant” is the one indexed instead of ”description”, because the condition on description search (description LIKE ’%danificado%’) is very specific and only optimizes the querying time for a few cases, while in other it can have a worse performance.

Because we have very small tables, it is very difficult to see how these indexes perform. In a real world situation, it would make sense to use them for tables which exceed the available memory of our database by several orders of magnitude.

The SQL commands related to Part 6 are contained in the end of file ”queries.sql”.

4 Application Development

The application has a main page at the file ”index.html” (`http://web2.tecnico.ulisboa.pt/ist190210/index.html`) that displays buttons to redirect the user to the other pages where it is possible to issue requests to the database. These requests include:

1. list substations (”list_substations.cgi”), transformers (”list_transformers.cgi”) and bus bars

(”list_busbars.cgi”) as well as remove them (”delete_substation.cgi”, ”delete_transformer.cgi” and ”delete_busbar.cgi”) and add new ones (”insert_substation.cgi” and ”update_insert_substation”, ”insert_transformer.cgi” and ”update_insert_transformer.cgi”, ”insert_busbar.cgi” and ”update _insert_busbar.cgi”). There are two pages for the addition of new elements: the first one collects the data of the new element from the user and the second one updates the necessary tables with that data. The delete pages do not need the user to explicitly input the data. Instead, when pressing the button, a hidden form submits the necessary data for the DELETE query to be executed;

2. register new non-line incidents (”register_incident.cgi” and ”update_register_incident.cgi) and edit existing incidents’ descriptions (”edit_description.cgi”and ”update_edit_description.cgi). Again, there are two pages when adding/editing contents of the tables: one gets data from the user and the other updates the database with the new data. Removing these items does not make use of the ON DELETE CASCADE instructions. Instead a manual removal is performed on every table necessary to prevent inconsistencies between the contents of different tables;

3. for substations, it is also possible to assign a new supervisor through a hyperlink (”new_supervisor.cgi” and ”update_new_supervisor.cgi”) present in the listing page that redirects the user to a form where the new supervisor data should be inserted.

In addition to these functionalities that were asked, a few extra ones were implemented as a way to improve the user experience with the application. These include:

1. listing the existing non-line incidents to check if new incidents were added successfully, and also to confirm that changes to an incident’s description were also carried out as the user expected;

2. in the page ”edit_description.cgi” where the user can edit the description of an existing non-line incident, the form field is already filled with the current description as a way to ensure that only small changes (like correcting typos) can be carried out quickly and easily without the need to copy and paste the current description from somewhere else or to write the full description again;

3. only relevant information is shown to the user when listing the tables. For instance, when listing the transformers, knowing the primary and secondary voltages does not benefit the user since these values can also be found by checking the corresponding bus bar voltages. By doing this we reduce the amount of information on the screen and do not overwhelm the user with pointless or useless data.

The application is implemented in way that prevents SQL injection attacks by using Python

dictionaries to format the query strings. All the operations performed that result in modifications to the table contents are performed inside transactions as a way to prevent inconsistencies when some of the operations are performed successfully but then an error occurs (in the case where multiple queries are performed).

As we can see in figure 1, a simple index is shown to the user.

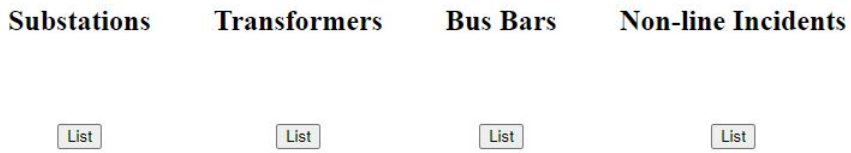

Fig. 1 – Index of the application

When the operation fails, the exception message is shown to the user. While in a real life situation this would not be advised from a security perspective, as it grants the user knowledge of the architecture of our application, it can be useful in an academic perspective. Either way, the user is always presented with option to go back to the listing page and from the listing page, it is also possible to go back to the index with the designated buttons. These functionalities can be observed in figures 2, 3 and 4.

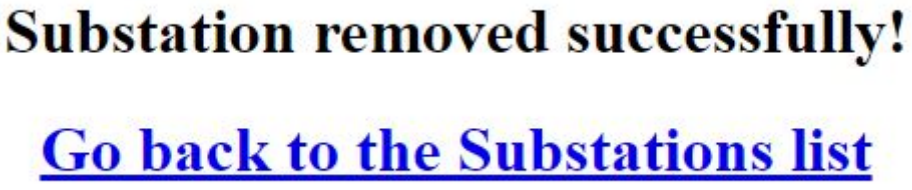

Fig. 2 – Message displayed when an operation is successful

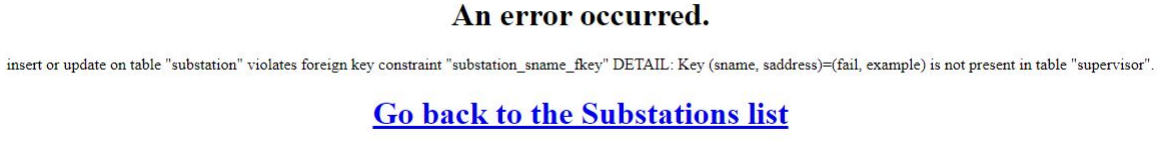

Fig. 3 – Message displayed when an operation fails

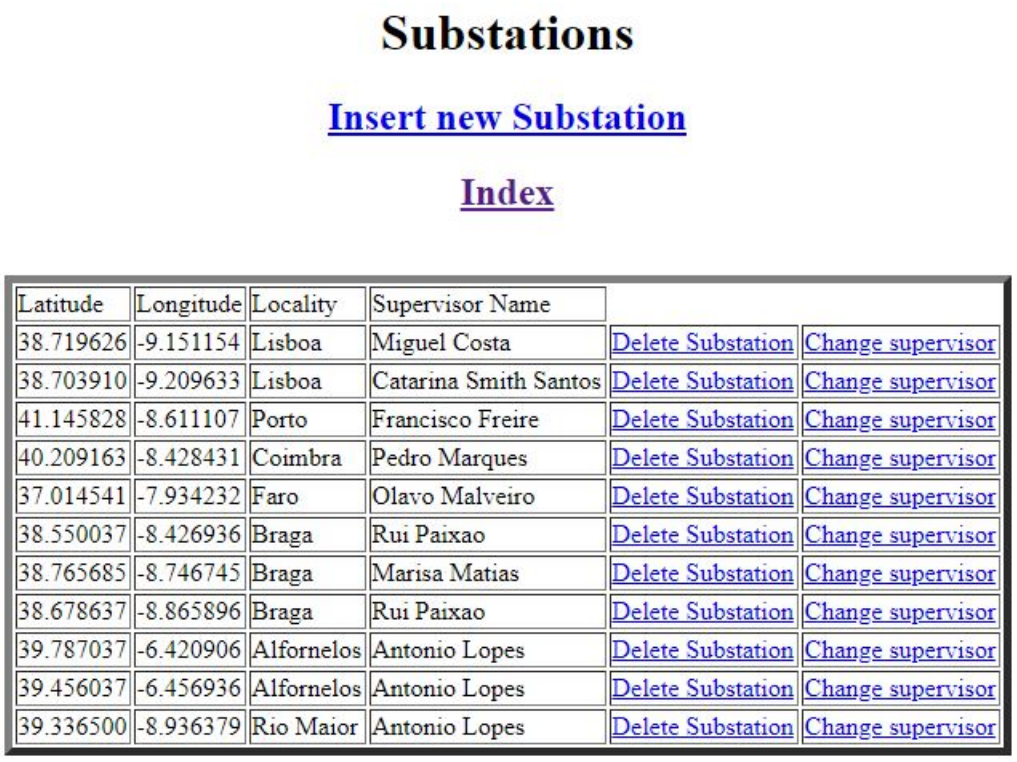

Fig. 4 – Example of a listing page with an ”index”button that redirects the user

5 Multidimensional Model

The star schema created in the database is contained in ”star_schema.sql” and it is composed by four dimension tables (”d_reporter”, ”d_time”, ”d_location”, ”d_element”) and one fact_table (”f_incident”).

The SQL commands used to load the necessary data into the star schema are in ”etl.sql”. Every row of information is added to the schema using functions. The data is loaded in a way so that the query from exercise 8 can be better understood.

In order to better define the location from ”d_location” that corresponds to each incident’s element it was assumed that our circuit was formed as busbar - transformer - busbar - line - busbar - transformer and so on. With this in mind a substation was considered to be composed by a transformer, the two bus bars it is connected to, and the line that is connected to the secondary bus bar of that same transformer.

6 Data Analytics Queries

The query that is asked for in exercise 8 can be written using different types of commands such as CUBE, ROLLUP and GROUPING SETS or simply using the union of GROUP BY clauses.

Using GROUPING SETS, the query returns a table with the number of incidents by each attribute separately.

In the case of CUBE and the union of GROUP BY clauses, it returns the number of incidents that verify each of all the possible combinations of the given attributes. The difference between CUBE and the union of GROUP BY clauses is that the first one returns the number of times an incident with no specified attributes happens, while the second does not. Besides that it is possible to do more operations using the union of GROUP BY clauses than using CUBE, for instance, if instead of wanting the possible combinations of all attributes, it's only needed some of those combinations.

ROLLUP returns the number of incidents that verify a different combination of attributes as shown in Figure 6, where there is a zoom in attributes from right to left.

File "olap.sql" is the one that contains this queries.

```sql
SELECT severity, locality, week_day, COUNT(*)
FROM f_incident i
INNER JOIN d_location l
ON i.id_location = l.id_location
INNER JOIN d_time t
ON i.id_time = t.id_time
GROUP BY ROLLUP (severity, locality, week_day);
```

Fig. 5 – SQL query that allows the analysis of the total number of anomalies reported by severity, locality and week_day using the ROLLUP command.

	severity	locality	week_day	count
1	<null>	<null>	<null>	7
2	Muito Grave	Lisboa	4	1
3	Grave	Faro	7	1
4	Grave	Faro	3	2
5	Ligeiro	Lisboa	3	1
6	Ligeiro	Coimbra	3	1
7	Grave	Lisboa	7	1
8	Ligeiro	Lisboa	<null>	1
9	Muito Grave	Lisboa	<null>	1
10	Grave	Lisboa	<null>	1
11	Grave	Faro	<null>	3
12	Ligeiro	Coimbra	<null>	1
13	Muito Grave	<null>	<null>	1
14	Ligeiro	<null>	<null>	2
15	Grave	<null>	<null>	4

Fig. 6 – Table returned by the query shown in Figure 5.