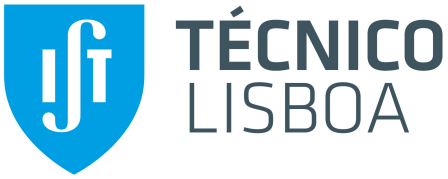

Instituto Superior Técnico

MEEC

Information Systems and Databases

Database Project, Part 3

Laboratory' shift: **b_SIBD77L03** — Lab 6 Monday (9:30 - 11:00)

Group 30:
Matilde Moreira, 84137—75hours—40%
João Cardoso, 84096—25hours—25%
Duarte Oliveira, 94192—50hours—35%

Laboratory Teacher:
Inês Filipe
Senior Lecturer:
Paulo Carreira

1^{st} *Semester*

2020-2021

1 Database Loading

The database that was created is presented in file *schemaPart3.sql* given by the professor. It was necessary to adapt the file populate.sql from the previous project to this new schema in order to avoid issues.

2 Integrity Constraints

It is necessary to write code using Stores Procedures and Triggers to implement the requested integrity constraints present on *RI.sql*. For the two first integrity constraints, the same logic is applied.Since we want the primary and secondary voltages, for every transformer, to match the corresponding voltages of the busbar. Thus, we use ALTER TABLE clauses to change the table previously created. To avoid using trigger, our idea was the same as presented in Part 2: Add a Unique constraint at Bus Bar Table to pair (Voltage,ID) and then create a new foreign key to Bus Bar at Transformer Table, where we also put the voltage of the transformer to check with the voltage of busbar. For the last integrity constraint, IC-5, we created a trigger and also a function. Our idea is each row, we insert or update, the trigger must check if the analyst is analysing incidents regarding elements of substation, which not supervises. So, in the function we raise an exception if the analyst is analysing an incident,which belongs to the substation he supervises it.

3 View

For this section we want to create a view to get the supervisors and the number of substations that each one of them supervisors but excluding the ones that do not supervise any substation. The code regarding this section is available in the beginning of the *queries.sql* due to the fact that this view is very useful to solve the queries asked in the next section.

4 Queries

In this section, we have to solve four queries. The first one has to return all the analysts that have analyzed every incident of element 'B-789'. The second query has to return all the supervisors that do not supervise substations located in the south of Rio Maior. We are given the coordinates, so that we can range the subquery inside of the "NOT IN" clause to filter the required coordinates of the substation. As it said, we easily discover the supervisors associated to these substations and then select them. For the third query, we want to know all the elements with the smallest amount of incidents. We select all the incidents grouping by id and then filter by the minimum of amount of incidents. Finally, the fourth query has to return the number of substations that each supervisor supervise even the supervisors which do not supervise any substations. This is the query that we use the view created in the previous section. This view is a table filled with the supervisors who has a substation associated to. Therefore, we can join the supervisors with no substation attributed using a UNION clause and the query is solved.

5 Application Development

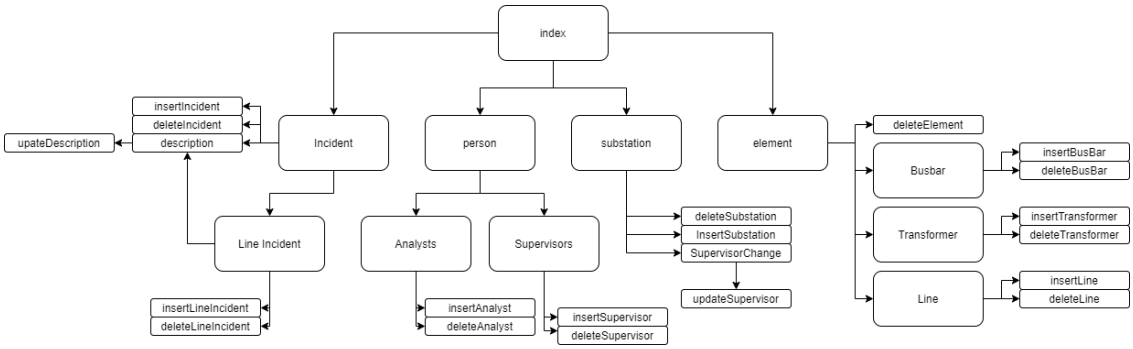

Figure 1: Hierarchy of WEB App

In the following section, it is possible to view the URL address, where the application can be tested, as well as the captured images that demonstrate the correct use of the WEB application. Still in the same section, it is possible to view some demonstrative examples and considerations about the available features are presented. The implementation of the application was done through the elaboration of a set of files written in CGI and HTML that offer functionalities that meet and beyond the proposals in the statement of the project. As suggested, simple solutions were used for data collection and presentation. To access the developed application, just access the following link:

https://web2.tecnico.ulisboa.pt/ist425437/

The use of the developed application starts at the entry page, *index.cgi*, whose screenshot is shown in Fig. 1.

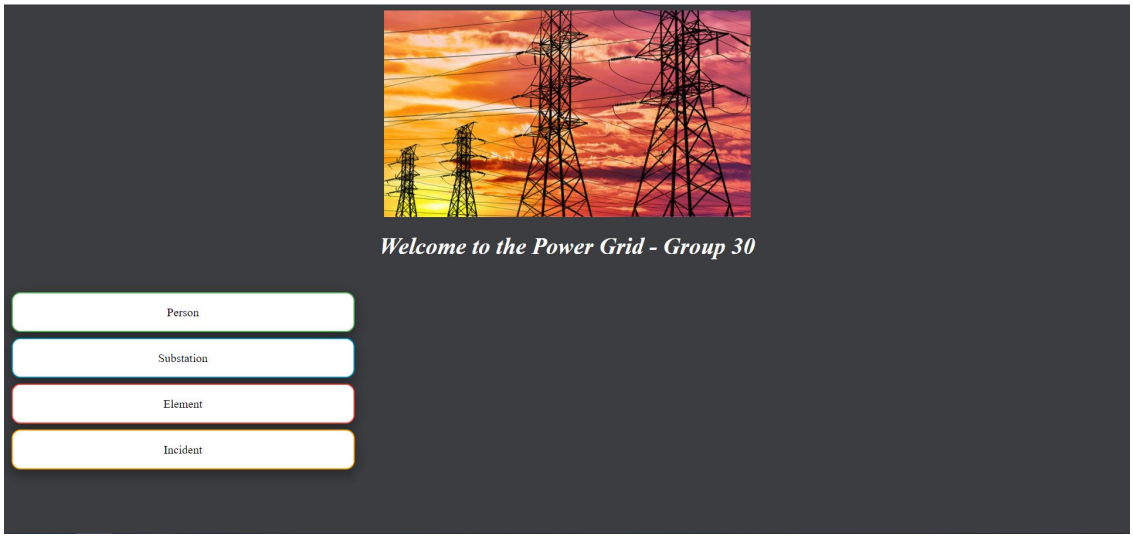

Figure 2: Screenshot of Index Page *index.cgi*

The main page has 4 buttons, each one redirects to another pages. The first assignment asked was to insert, list and remove bus bars, substation and also transformers. This is possible, clicking at Element button and then it redirects to *element.cgi*, Fig. 3.

It is possible to insert each one of the elements (busbar,transformer,line) just clicking at List Bus Bar, List Transformer or List Line, respectively and inserting the values, such as the id or even impedance, or voltages. We also implemented a CGI files to delete each one of the elements. This is possible just clicking Delete at one of the tables. To list one of the elements, we just make a query to select all the attributes associated to the respective element and then we print into a table. This table is the same table as we mentioned where it is possible to delete the respective elements.

The second assignment is related with the opportunity to change the supervisor assigned to a substation. And this is possible to obtain, clicking at Substation at main page, Fig. 4. Then, you click Change Supervisor to the substation, you want to change. Then it redirects to another page where you can select another person from supervisor table. After you select the supervisor, you press the button Change Supervisor and then it updates.

Finally, the last assignment asked was to register incidents for non-line elements and edit their description. We solved this assignment also for line incidents. After click at Incident at index.cgi, Fig. 5, it is possible to see all incidents, even the line incidents. In that page, it is possible to insert an incident and also to change the description. It is also possible to do that at line incident, just clicking at Line Incident.

6 Indexes

6.1 Return the number of transformers with a given primary voltage by locality:

The main objective of using indexes is to reduce the query answering time. In the first part we want to optimize the query that returns the number of transformers when is given a primary voltage grouping by locality. To facilitate our tests, let's assume that a primary voltage is 333.1V (pv = 333.1). Firstly, let's use the command "SET enable 'seq scan' = OFF;" to disable sequential scan when running a query. Then, we run: "EXPLAIN ANALYZE SELECT locality, COUNT(*) FROM transformer NATURAL JOIN substation WHERE pv = 333.1 GROUP BY locality;" . EXPLAIN ANALYZE is a Postgres command that accepts and executes the query and returns data regarding the query plan where we can see the approach that the planner took to execute the query and the planning and execution time as well. As we want to filter localities by the primary voltage of the transformer, the best choice here is to use an hash index. Analysing the given query, it would not make sense to use a b+tree index since we just want to filter by a line. An hash index will organise records into pages according to a hash function. This hash function will take as input the search key value (in this case, the primary voltage) and hashes it to know the container where the value is located. As we can see in the next two figures, Fig.6 and Fig.7, there are differences in the cost, planning and execution time of the same query with and without index. It is not so noticeable the effect of the index since the given data set is not so big as it should. Although, we can assist a decrease of the planning and execution time and the cost as well.

6.2 List all descriptions of line incidents that start with a given prefix within two points in time:

In the second part we want to optimize the query that lists all descriptions of line incidents starting and ending in two points in time. Let's assume that we want every description of line incident occurring

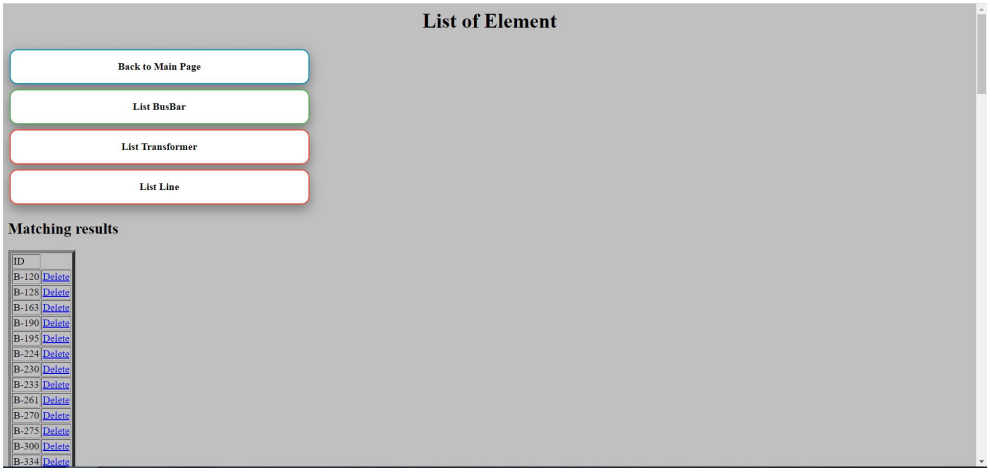

Figure 3: Screenshot of Element Page *element.cgi*

Figure 4: Screenshot of Substation Page *substation.cgi*

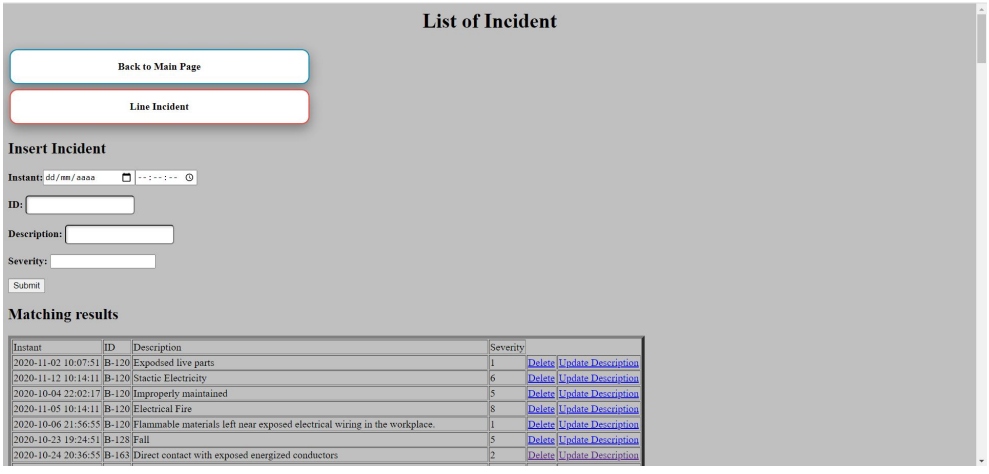

Figure 5: Screenshot of Incident Page *incident.cgi*

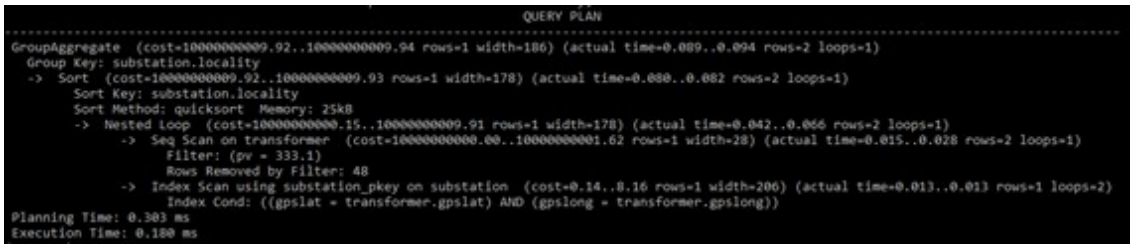

Figure 6: Query plan of the first query without using any index *index.cgi*

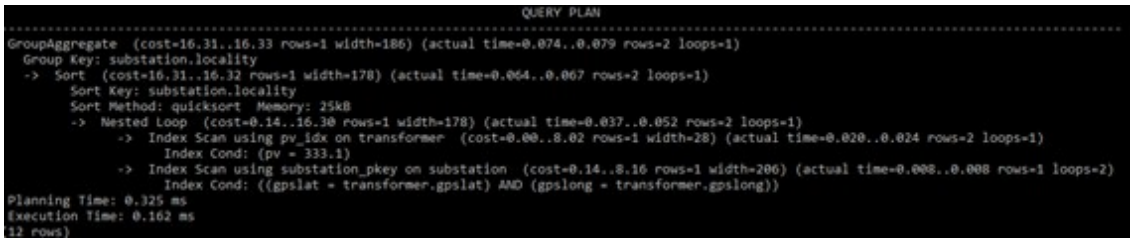

Figure 7: Query plan of the first query using **'CREATE INDEX pv_idx ON transformer USING HASH(pv);'**

in 2020. Again, let's use the command "SET enable seq scan = OFF;" to disable sequential scan when running a query. Then, we run: EXPLAIN ANALYZE SELECT id, description FROM incident WHERE instant BETWEEN '2020-01-01 00:00:00.000' AND '2020-12-31 00:00:00.000' AND description LIKE 'Fall and Burn';. In these case, the best choice is doing a composite index that contains the incident and the description. The B+Tree is a balanced tree that auto-adapts with inserts and deletes to keep balanced ensuring the same depth for all nodes. Index entries are ordered by the search key value kept in a hierarchical search structure. This type of index support range queries and composite indexes oppositely to the hash index, used in the previous question. As we can see in the following figure, we have a bitmap heap scan on incident and it filters less (236) rows than when scanning by the primary key of the incident, so we can conclude that this method is the most efficient to reduce the query answering time.

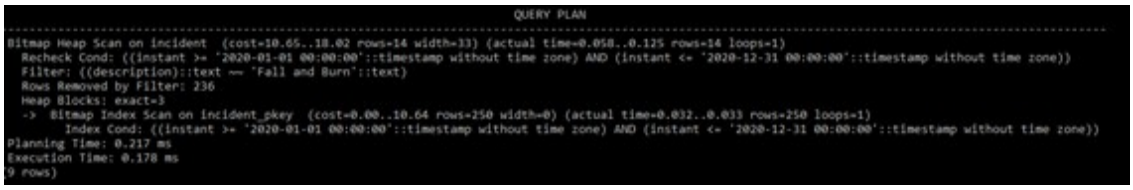

Figure 8: Query plan of the second query using **'CREATE INDEX description_of_incidents_idx ON incident USING BTREE(instant, description);'**

7 Multidimensional Model

In this section, the main objective is to create the star schema with the dimension and fact tables and then inserting population into them. The creation of star schema is written in the *star schema.sql* and the insert of information in the tables is present in the *etl.sql*.

The dimension tables are: the reporter table, the time table, the location table and the element table. Each one of these four tables has as primary key an id which will be foreign key in the fact table. The set of these id's will build the primary key of the fact table, the table of incident. We can notice from observing the *star schema.sql* created, that all the data type of each id is SERIAL. SERIAL is a data type of POSTGRESQL that is used to define auto increment number of column in a table, generating a serial sequence of integer numbers. We also observe that in the time dimension table the other variables are integer not null and for the type of the element in the element dimension table we use a CHAR NOT NULL variable to identify if the element is a busbar, transformer or line. Therefore, use use the substring of the if of the element which will be the first letter of it.

To insert into the reporter dimension table, we get the information from the analyst table because the reporter of an incident is an analyst. For the time dimension table, we create a procedure that uses as TIMESTAMP variable to create id's of time for every day of the year. The procedure extracts from the timestamp the day, the day of the week, the month, the trimester using a CASE to set the conditional system for the program to know in what range of months the trimesters are inserted and then the year. Now for the location dimension table, we get the coordinates and locality from the table of substation present in the *schemaPart3.sql* and the element dimension table will have the id of element table of the schema given by the Professor as well. Finally, to insert all values in the fact table, we assume that the operational table is a NATURAL JOIN of three tables: the analyses, the incident and the transformer. Then, we do LEFT OUTER JOIN's for each dimension table to make the values of these tables corresponding to the values of the operational table. It is necessary to highlight the fact that in the LEFT OUTER JOIN of the location dimension table we equal the coordinates of the transformer which is inside the operational table, as said earlier.

8 Data Analytics Queries

In this final section, we have to write an SQL query to know the total number of anomalies reported by severity, locality and day of the week, basing on the *star schema.sql* with the values inserted on the *etl.sql*, as previous seen. There are several options to solve this query because there are various OLAP operations in SQL. We use a set of multiple aggregations joining the queries with the UNION ALL operation. Firstly, we select the severity, then the day of the week and then the locality.