Database design

Unit 3: Database design & development

Students: Quique López, Sergio Madaleno & Federico Sanjuan
Teacher: Gustavo Aranda
Unit: Unit 3 Database design & development
Level: 4

Index

Database

Firstly, we design all the database's tables and how they would be related to each other. We wanted to represent a generic company's employee database, where every employee is accounted for. This database could be used as a Human Resources' inventory.

For the *Employee* table we added an ID, which will be considered as the **primary key** (PK from now on), a series of strings where the personal information of the employee will be stored as the name, surname and the address, a float number with the annual salary of the employee and a series of integers with several **foreign keys** (FK from now on) with information of the employee's city of residence, nationality and the company where they work.

For the *Company* table we added an ID, which will be the **PK**, a string with the name of the company and an integer as a **FK** to the *Country* table, to represent where the company is based.

For the *Company* table we added an ID, which will be the **PK**, a string with the name of the company and an integer as a **FK** to the *Country* table, to represent where the company is based. Basically, the *Country* table needed the same parameters as the *Company* table.

For the Country table we added an ID, which will be the **PK**, and a string with the name of the country.

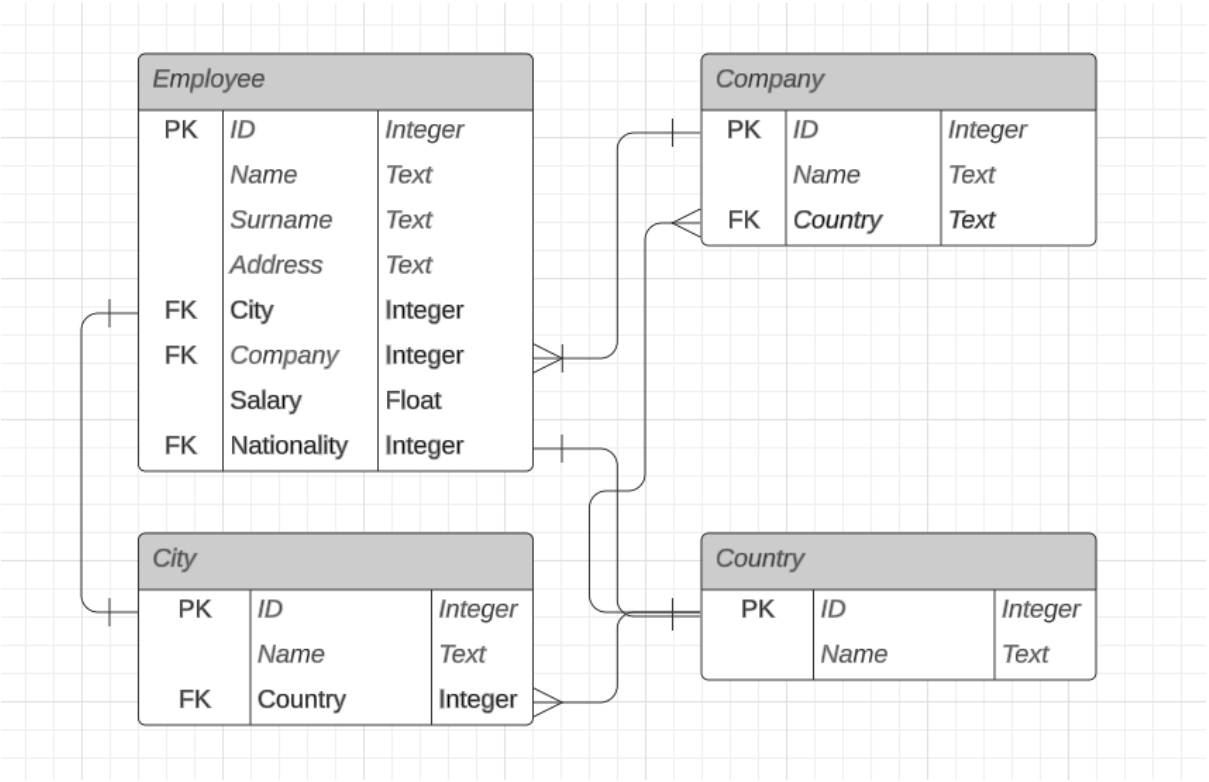

Figure 1. Database design

We used a struct variable where we store the data of a tuple before showing it on the screen. This struct has the same intern variables as the table's columns, so every data is stored in each own variable.
After storing the data, we can export the data or modify it as we please.

Personal tasks & workgroup plan

We divided the workload into 3 parts, one for each member of the project. We used a trello-like software that is integrated with GitHub so we could link any card to the project and the collaborators implicated.

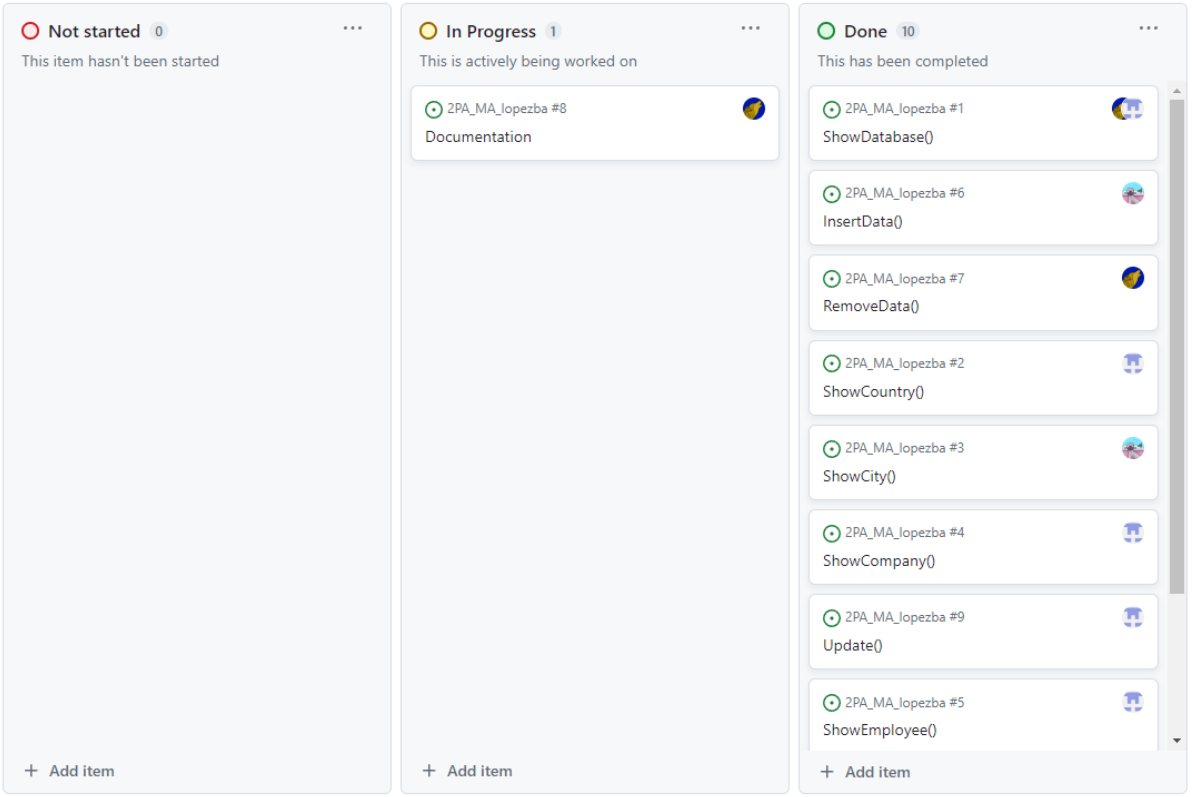

Figure 2. Example of how we organized the workload

Federico Sanjuan

Showing database's tables

One of the first tasks we tried to solve was showing a table on screen. This was a difficult task as we had not worked with callbacks yet and it was a tricky function to implement. It consumed us the first few days in order to understand fully how to design our functions in order to work properly.

Update data

We developed functionality where we could update any tuple of data in any table using a SQL query. After understanding showing data on screen, this task was easier to implement.

Sergio Madaleno

Showing table

We needed several people the first days to work faster in order to finish everything by our timetable, so Sergio and Federico worked together to finished our first task as fastly as they could.

Insert data

We added the functionality of inserting whole tuples into the database using SQL queries. It was not a difficult task as it was one of our last functionalities to implement.

Documenting the code

When we finished programming our functions, Sergio documented the functions using the Doxygen system of commentaries. This was easy as Sergio was implicated in the development of almost every function in our code.

Quique López

Design of the interface

We had to redesign the interface in order to be more user friendly, as the first attempt was understandable just to the development team.

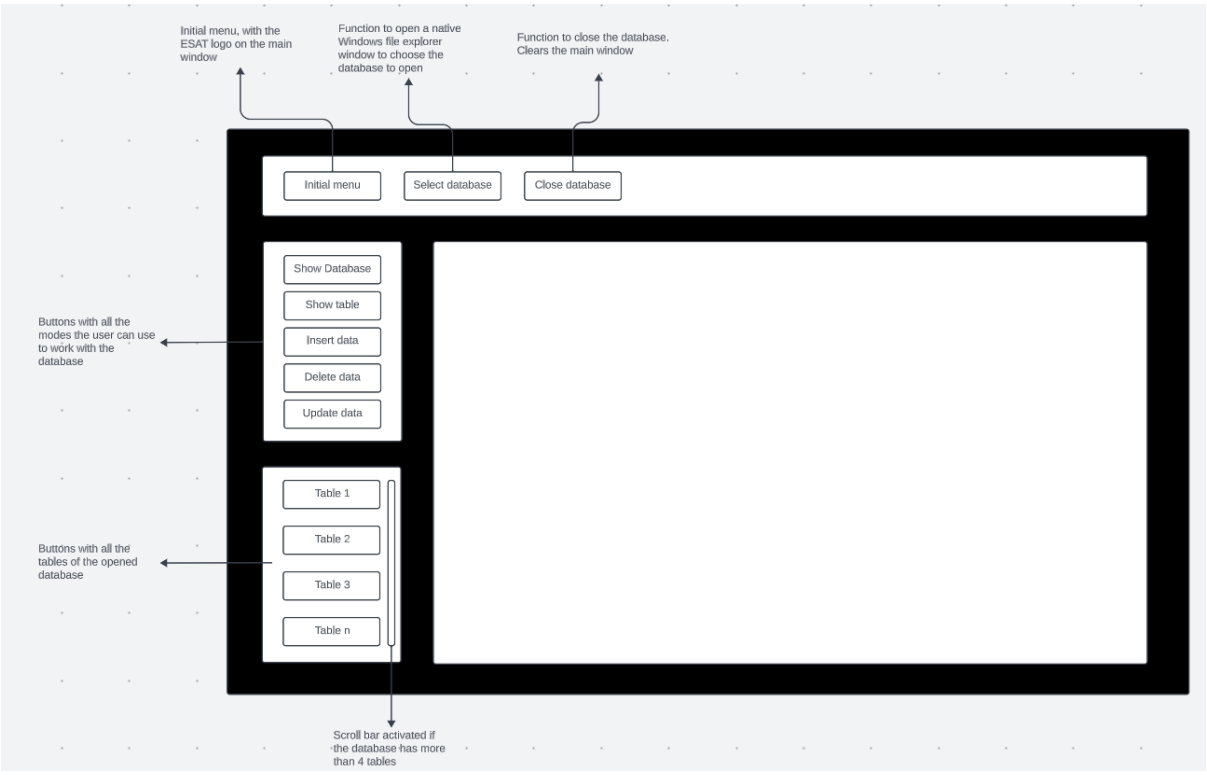

Figure 3. Redesign of the user interface

Delete data

We added a functionality to remove whole tuples from a table of the database. Once we understood how the callback functions worked, it was not a difficult task.

Technical documentation

buttons_window.h

Functions

Name	Data type	Parametres	Description
TopWindow	void	GlobalData *info	Create GUI window with a button
MenuSelectionWindow	void	GlobalData *info	Create a menu selection window for choosing different application options.
TableSelectionWindow	void	GlobalData *info	Create a window for selecting a database table

count_rows.h

Functions

Name	Data type	Parametres	Description
ReadRows	int	void *data int argc char **field_values char **colNames	Reads and processes data from a database result row
ShowRows	void	GlobalData *info int table_identifier	Queries a SQLite database for the number of rows in specific tables.

global_data.h

Structs

GlobalData

Struct that englobes all the global variables that will be used in the program.

Enumerators

TableSelector

Enumerator that allows the program to know which table is selected at the moment.

MenuSelector

Enumerator that allows the program to know which menu is selected at the moment.

Constants

Name	Data type	Description
kWindow_width	const int	Sets the width of the screen
kWindow_height	const int	Sets the height of the screen
kNumTables	const int	Sets the maximum number of tables the database has

global_functions.h

Functions

Name	Data type	Parametres	Description
LogError	void	const char* errorMessage	Logs an error message to a buffer
ShowErrorWindow	void		Displays an error window with error messages
ExecuteSQL	void	const char *sql	Execute a SQL query on a SQLite database

CreateWindow	void	const char *name ImVec2 pos ImVec2 size	Create a GUI window with specified position and size
CloseWindow	void		Close the currently open GUI window

query_window.h

Functions

Name	Data type	Parametres	Description
ExecuteUserQuery	void	GlobalData *info	Execute a user-provided SQL query and display the results
BottomWindow	void	GlobalData *info	Display the bottom user interface window

show_tables.h

Structs

TableEmployee

Struct that englobes every data field of the Employee table.

TableCompany

Struct that englobes every data field of the Company table.

TableCity

Struct that englobes every data field of the City table.

TableCountry

Struct that englobes every data field of the Country table.

Functions

void InitTable(GlobalData *info, int table_identifier)

Name	Data type	Parametres	Description
InitTable	void	GlobalData *info Int table_identifier	Initialize a database table by resetting its contents.
TableEmployeeCallback	static int	void *data int argc char **argv char **colNames	Callback function for processing database results and populating the Employee table.
TableCompanyCallback	static int	void *data int argc char **argv char **colNames	Callback function for processing database results and populating the Company table.
TableCityCallback	static int	void *data int argc char **argv char **colNames	Callback function for processing database results and populating the City table.
TableCountryCallback	static int	void *data int argc char **argv char **colNames	Callback function for processing database results and populating the Country table.
DatabaseStructureCallback	int	void *data int argc char **argv char **colNames	Callback function for displaying the structure of a database table.
EmptyCallback	int	void *data int argc char **argv char **colNames	Callback function for processing the results of an SQL query
ShowQuery	void	GlobalData *info	Execute a user-defined SQL query to display database table structure.
ShowDatabaseStructure	void	GlobalData *info	Display the structure of a database table based on user selection.
ShowDatabaseTable	int	GlobalData *info	Display the contents of a selected

			database table.
Updatevalues	void	GlobalData *info	Update values in a selected database table
InitTableValues	int	int i	Init values in a database table
InsertDataTable	void	GlobalData *info	Insert new rows into the table
RemoveData	int	GlobalData *info	Remove data from specific tables and update the user interface

User manual

Opening the program

Go to the /bin folder, where the executable file will be stored. Double click on the file and it will open.

Selecting the database

In this project, we used a generic company's employee database, so it would fit any enterprise. All the project has been created around using only one database, so the user can not select a different database. However, this feature will be developed in the next few months.

Selecting menu

We have several options in our program to visualize, edit or remove data from a table of the database.

Select mode

The select mode allows the user to visualize a table with all its columns.

Click into the *Select tables* button on the upper left box to start visualizing the data in the main window.
The user can select any table from the bottom left box.

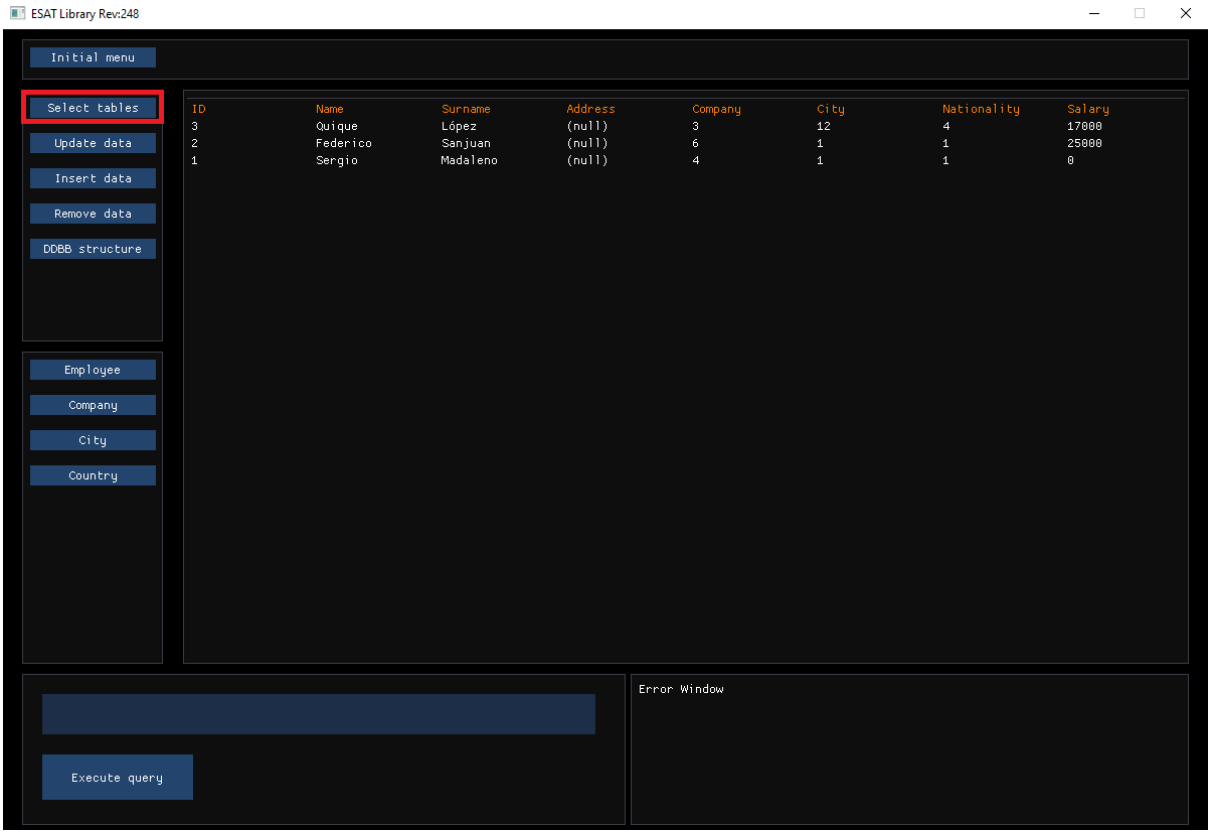

Figure 4. Select table mode

Update mode

The update mode allows the user to change any data from a tuple of a table.

Click on the *Update data* button on the upper left box to start updating the data. The data will be shown on the main window.
The user can select any table from the bottom left box.

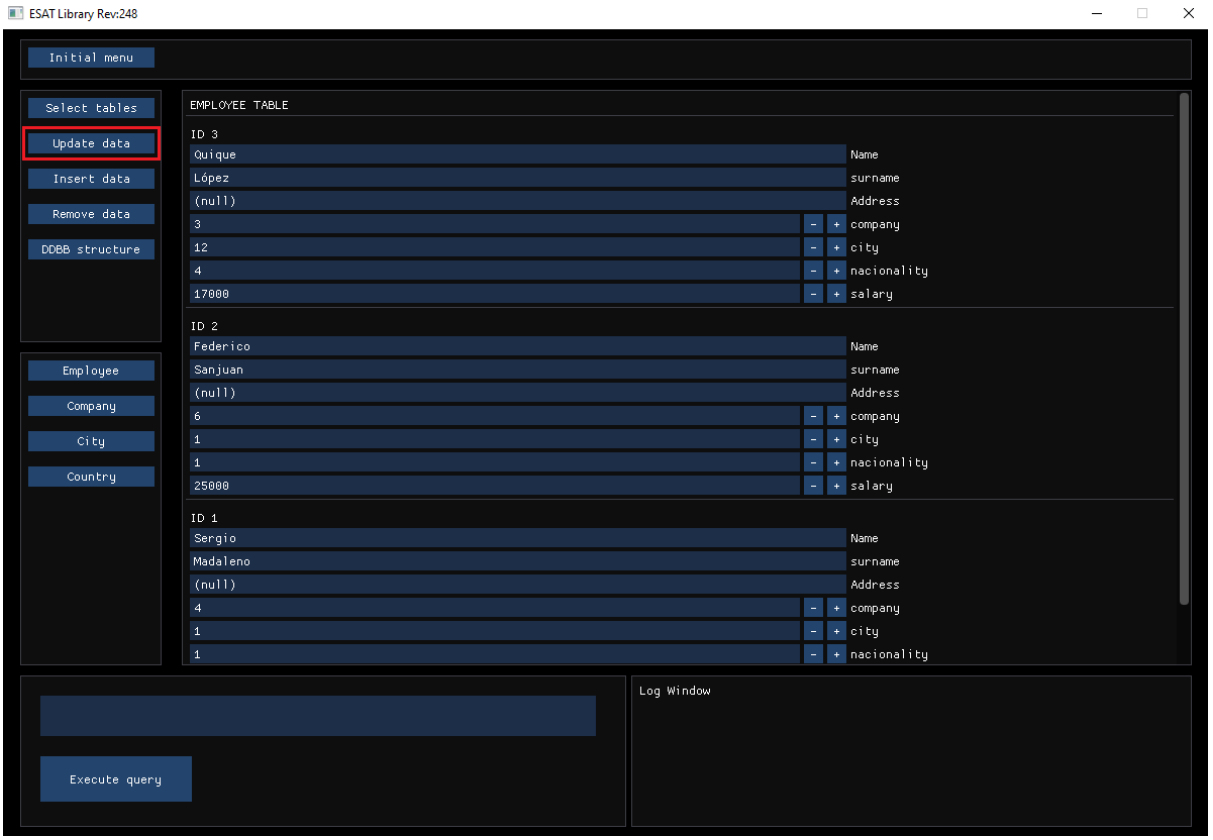

Figure 5. Update data mode

Insert data mode

The inserting mode allows the user to insert data into a tuple of a table.

Click on the *Insert data* button on the upper left box to start inserting the data. The data will be shown on the main window. To validate the data written, the user must click on the *Insert* button at the end of the input windows.
The user can select any table from the bottom left box.

Figure 7. Insert data mode

Delete data mode

The deleting mode allows the user to remove any tuple from a table.

Click on the *Remove data* button on the upper left box to select which ID the user wants to delete. The data will be shown on the main window in real time just under the ID selecting button.
The user can select any table from the bottom left box.

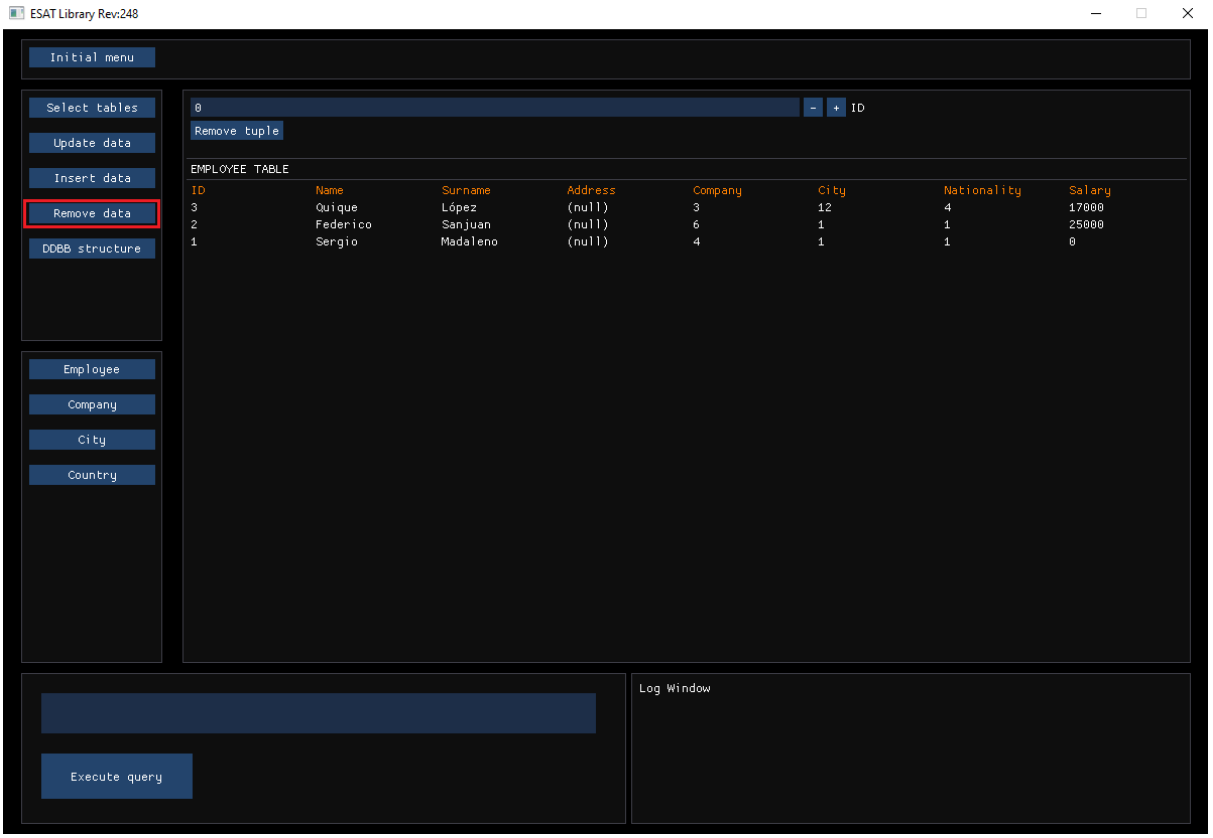

Figure 6. Remove data mode

Show structure mode

The showing structure mode allows the user to visualize how many tables there are in the database at any time.

Click on the *Show DDBB structure* button on the upper left box to visualize the data. The data will be shown on the main window.

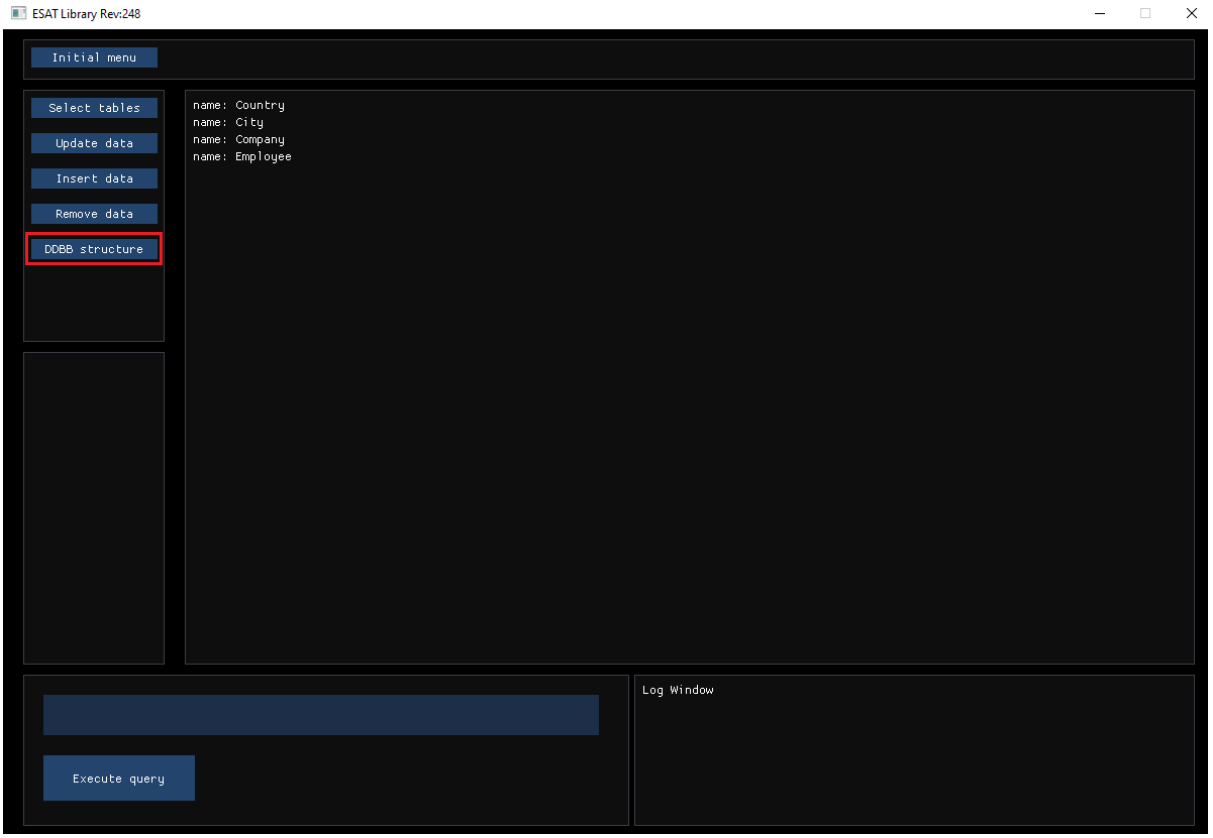

Figure 8. Show database structure mode

Ask a query

On the bottom left window, the user can write up any query they want and it will be executed when the user clicks on the *Show query* button below. The result of the query will be displayed on the main window.

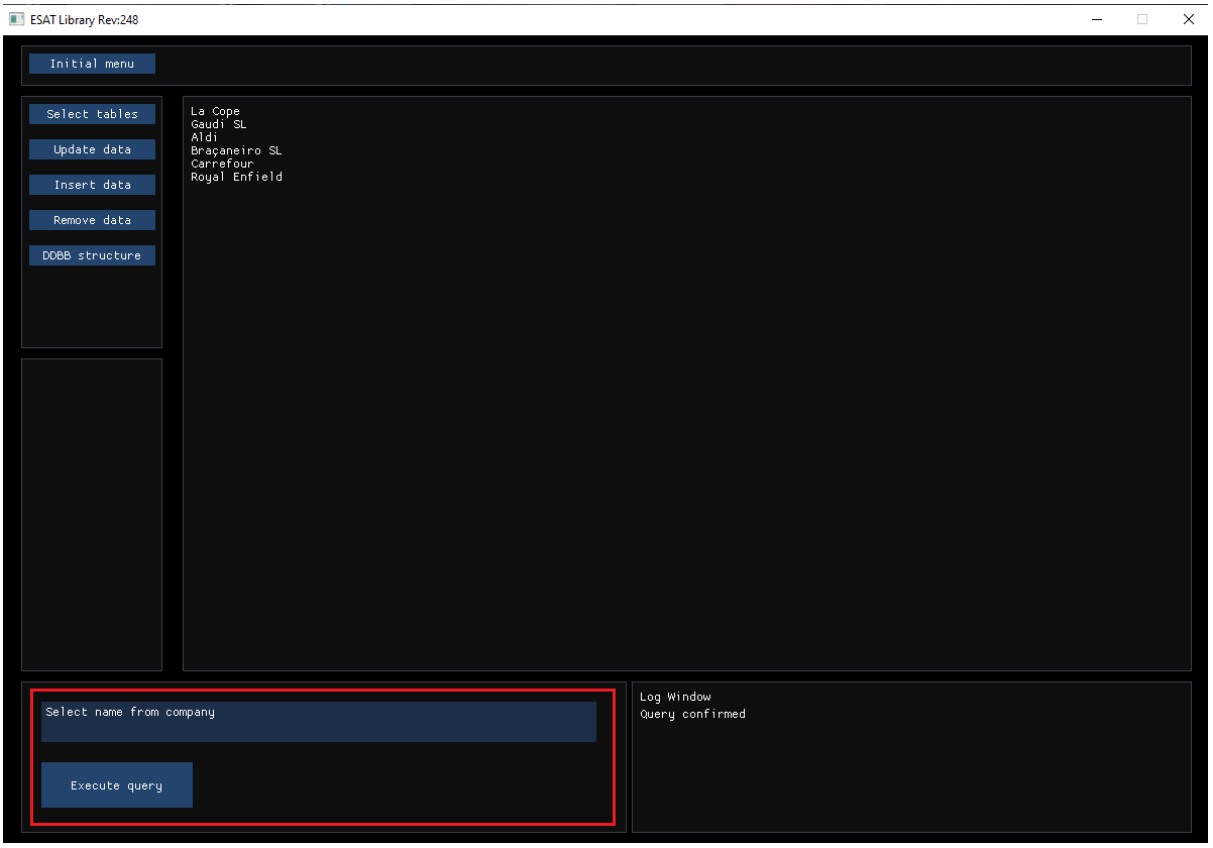

Figure 9. Example of a query

Error message window

If a query on any mode has found a problem inserting, removing or getting the data, an error message will be displayed on the bottom right window specifying the error found.

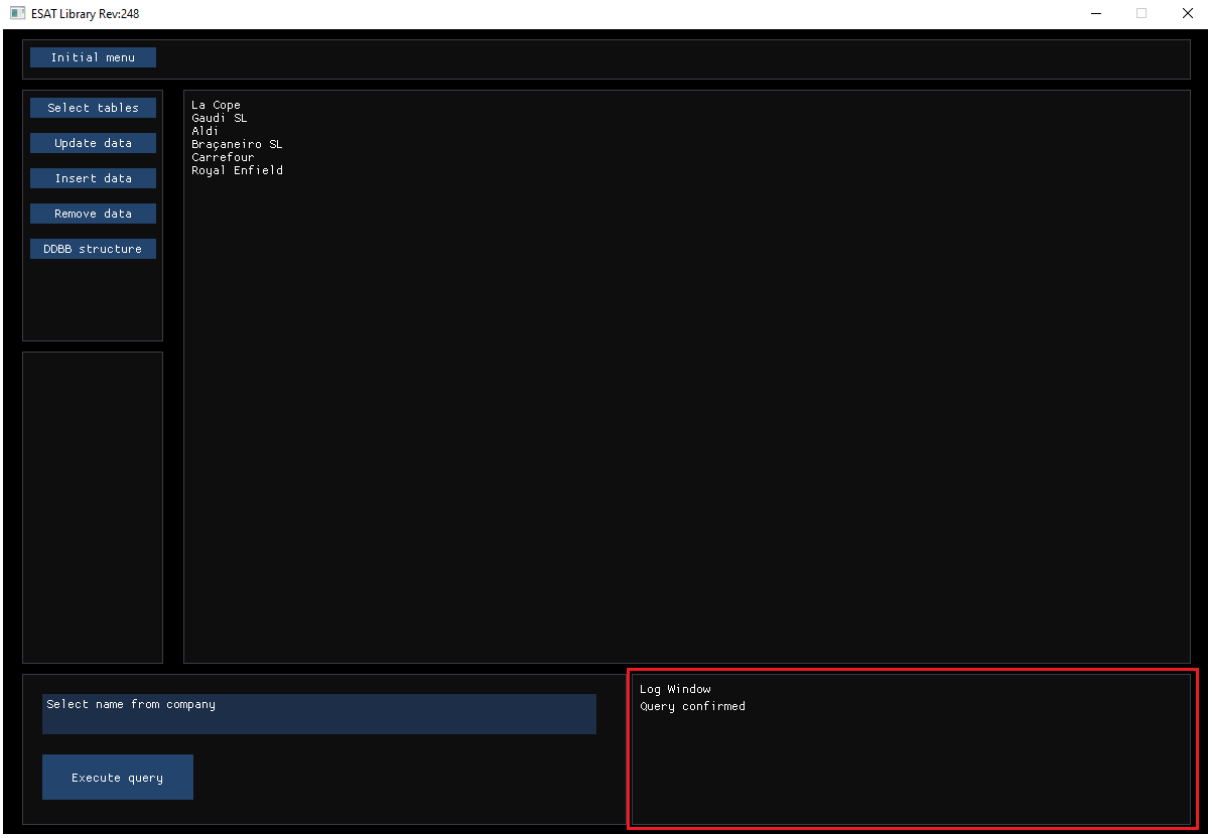

Figure 10. Example of a confirmation message in the log window

Closing the program

To end all processes, the user can press the *Escape key* or click on the cross located at the top right corner of the window.

Conclusions and Future Work

In this project we would like to implement in a foreseeable future because of lack of time, such as the following points:

- Allowing the program to read from any database the user opens.

  We would like to implement a new functionality that allows the user to open any database he wants using a Windows file explorer native window. This also implies that we would need to develop a variable struct to store any type of data we receive from the callbacks, as we would not know what that table contains.

- Improving the query window.

  At this moment, the program is designed to accept only a database and its tables, however, if we would implement the previous point, we would need to develop a system to store any type of data we receive from the callback.

- Error messages

  Right now we have an output window where the user can see if the query that he tried to send was correct and if any error has occurred in the process.
  However, we would like to create confirmation windows to validate the intentionality of the user. Also, it would be great to have a history of the error messages, as now the new messages overwrite the old ones on screen.