Introduction

This document is written in order to document all things pertaining to the database of the project. This document is a side file to the README one, therefore all necessary vocabulary and where does the database go and what's it used for is written there.

Introduction to the problem

As stated in the main document, one of the intents of the project is to store data in a database. Still, in order not to muddle unnecessarily it, all things related to the database will be reflected here, which includes which database should we use, how to create it, a brief comment on why the code is just made for a specific database, the database's current structure and an example on how a proper database should be made for the project.

Explanation of the solution

For starters, we have to choose a kind of database. As we want to make an organized structure, we have to choose a relational database, such as MySQL, Oracle, … For the sake of simplicity and to show how the project works with a database, a SQLite one has been chose, which always works as a local database in spite of the existence of a proper local database.

Also, to explain why the code is written just for this one, we have to refer a bit on how databases work on programming languages. The main idea of a database there is that you create a connection and then use a cursor to execute the different queries. This way we only have to change two things in-code if we want to change the kind of database we are using: the import of the connector module (sqlite3 on our case) and the parameters for the connection. The remainder of the script should work independently of the kind of database, as SQL is common to all relational databases.

The file “Creationqueries.sql” shall serve us to illustrate the structure of the database, as these are the queries one’d have to execute to create it.

The first query is trivial, which is the one that creates the database perse. The second and third ones, however, are the ones that create the tables we use on the project: Probes, where all probes are registered and Registries where all the ping data is dumped into from the local database.

Both tables use an ID field as the primary key (field which has to be unique and non-empty, plus usually it’s auto-incremental).

The Probes table has five fields, whose name and kind are stated below:

1. Dominion, which is a character chain of length less or equal to 15. This is the dominion the probe belongs to.
2. Probe_name, which is a character chain of length less or equal to 12. This is the name of the probe.
3. Address, which is a character chain of length less or equal to 15. This is the address the probe points to.
4. Ratio, integer. This is the seconds elapsed between probings.
5. Status, which is a character chain of length less or equal to 10. This is set by default to “Inactive”. This is the status of the probe.

The Registries table has six fields, whose fields are stated below. It also has the fields “Dominion”, “Probe_name” and “Address” in the same terms as the Probes table, therefore I’ll skip them.

1. Dumpdate, date type. The date when the registry was made.
2. Dumptime, time type. The time the registry was made.
3. Ping, integer. The average ping registered at the moment of the registry.
4. Status, which is a character chain of length less or equal to 15. This is the status returned by the probe of the connection.
5. Code, which is a character chain of length less or equal to 10. This is the code we have assigned to that particular status.

- # Improvements to be made

Also, how would a real database usage look like.

First of all, from a business standpoint, we’ll more than likely have multiple probes. This means many things. First, we cannot use a local database, and we’ll have to use one located on a server. Also, we’ll have to create multiple tables.

We can keep the aforementioned Registries one, which we can duplicate to create each instance. We can create one per dominion so all probes related to a single dominion store information or we can create one per probe so each probe has its own table to store information or we can do both at once.

Regarding Probes, it’d be wise to register them by dominion on duplicates of the Probes table, so we have each dominion organized. Plus, the probes on each Registries duplicate, if organized by dominions, should have the name as a foreign key (constrains which checks in another table the existence of said registry) referring to the respective Probes copy.

This way we’d have a database which allows for organized storage of different probes at once on an outside server.