Pandora FMS Engineering

Pandora FMS Engineering

Pandora FMS Database Design

The early versions of Pandora FMS, from 0.83 to 1.1, were based on a very simple idea: one data, one database insertion. This allowed the software to perform easy searches, insertions and other operations quickly.

Apart from all the advantages of this development, there was a drawback: *scalability* (rapid growth without affecting or with little effect on operations and work routines). This system had a defined limit in the maximum number of modules it could support, and with a certain amount of data (more than 5 million elements), performance was affected.

MySQL cluster-based solutions, on the other hand, are difficult: although they allow you to handle a larger load, they add some extra problems and difficulties, and also do not offer a long-term solution to the performance problem with large amounts of data.

Currently, Pandora FMS implements a data compaction in *real time* for each insertion, besides performing a data compression based on interpolation. On the other hand, the maintenance task allows to automatically delete data that exceeds a certain age.

The Pandora FMS processing system stores only “new” data: if a duplicate value enters the system, it will not be stored in the database. This is very useful to keep the database reduced, and it works for all Pandora FMS module types (numeric, incremental, *boolean* and text string).

These modifications involve big changes when reading and interpreting data. In the latest versions of Pandora FMS, the graphic engine has been redesigned from scratch to be able to represent the data quickly with the new data storage model.

The compaction mechanisms also have certain implications when reading and interpreting the data graphically: Currently there is a graphical configuration menu which allows adding percentiles, real-time data, when events and/or alerts occurred, in addition to other options.

Additionally, Pandora FMS allows the total disaggregation of components, so that the load of data file processing and execution of network modules in different servers can be balanced.

Other technical aspects of the DB

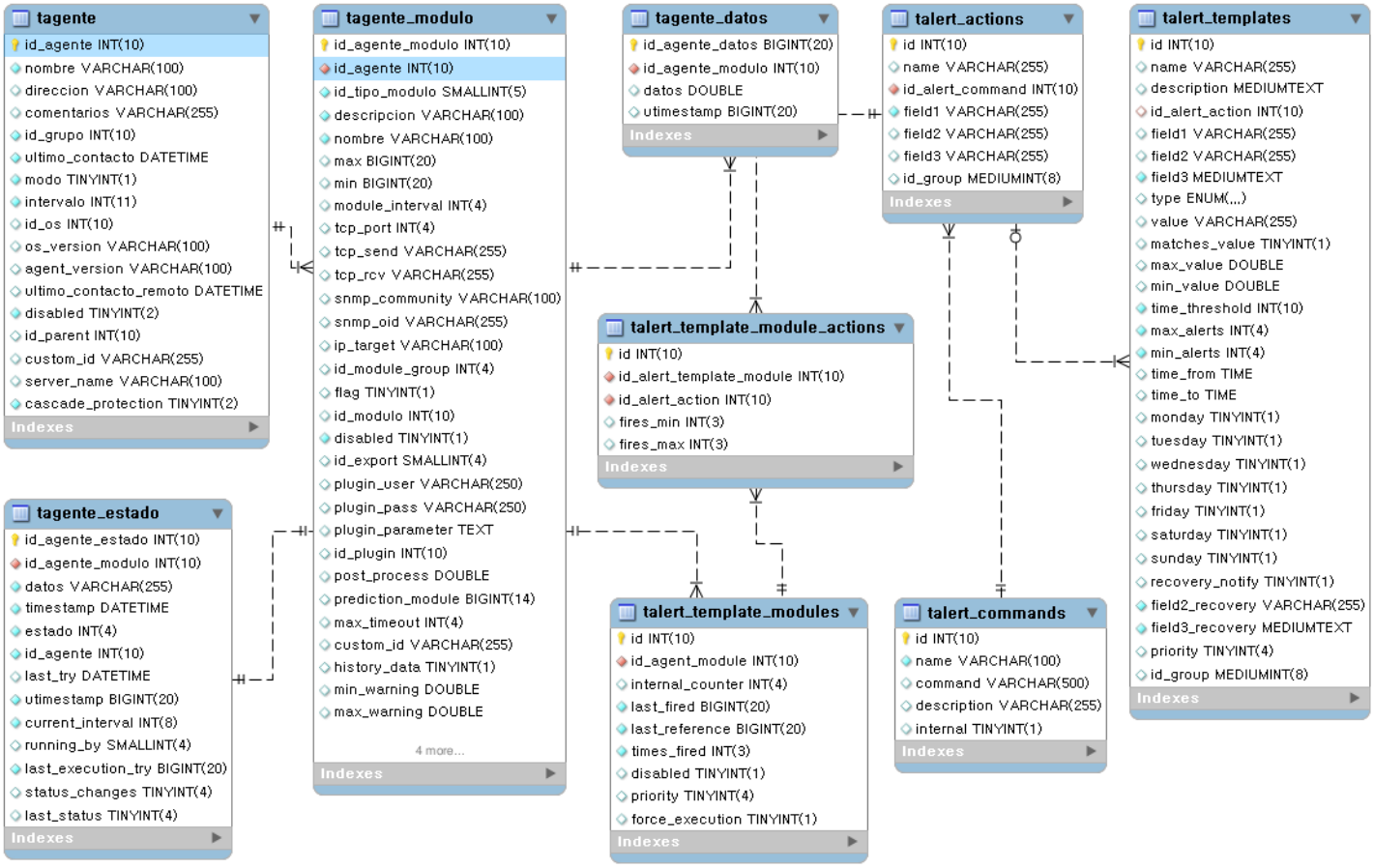
Throughout the software updates, improvements have been implemented in the relational model

of the Pandora FMS database. One of the changes introduced has been indexing the information based on *different types of modules*. This way, Pandora FMS can access much faster to the information, because it is distributed in different tables.

It is possible to partition the tables (by timestamps) to further improve the performance of access to historical data.

In addition, factors such as the numerical representation of timestamps (`_timestamp_` UNIX format), speed up searches for date ranges, date comparisons, etcetera. This work has led to a considerable improvement in search times and insertions.

Main database tables



Real-time data compression

To avoid overloading the database, the server performs a simple insertion time compression. A piece of data is not stored in the database unless it is different from the previous one or there is a difference of more than 24 hours between the two.

For example, assuming an approximate 1 hour interval, the sequence 0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0 is stored in the database as 0,1,0,1,1,0. Another consecutive 0 will not be stored unless 24 hours have passed.

Compression greatly affects data processing algorithms, both metrics and graphs, *and it is important to keep in mind that the "gaps" caused by compression need to be filled*.

Taking into account all the above, to perform calculations with the data of a module given an interval and an initial date, the following steps must be followed:

- Search for the previous data, out of the given interval and date. If it exists, bring it to the beginning of the interval. If it did not exist previously, there would be no data.
- Search for the next data, out of the given interval and date up to a maximum equal to the module interval. If it exists, bring it to the end of the interval. If it does not exist, the last available value should be extended to the end of the interval.
- All data is traversed, taking into account that a data is valid until a different data is received.

Data Compacting

Pandora FMS has included a system for *compacting* the database information. This system is oriented to small and medium size scenarios (from 250 to 500 agents and with less than 100,000 modules) that want to have an extensive information history but *losing* details.

The maintenance of the Pandora FMS database that is executed every hour, and among other cleaning tasks allows to perform a compaction of old data. This compaction uses a simple and linear interpolation, as it is an interpolation, this makes that detail is lost in this information, but it will still be informative enough for the generation of reports and monthly, annual graphs, et cetera.

In large databases, this behavior can be quite costly in terms of performance and would have to be disabled; instead, it is recommended to opt for the historical database model.

Historical database

The historical database is a feature used to store all the past information, which is not used in recent day views, such as data older than one month. This data is automatically migrated to a different database, which must be on a different server with a different storage (disk) than the main database.

When a graph or a report with old data is displayed, Pandora FMS will search the first days in the main database, and when it reaches the point where it migrates to the historical database, it will search in it. Thanks to this, the performance is maximized even when accumulating a large

amount of information in the system.

Advanced configuration

The Pandora FMS default configuration does not transfer string text string type data to the history database, however if this configuration has been modified and the history database is receiving this type of information it is essential to configure its purging, otherwise it will end up occupying too much, causing big problems and obtaining a negative impact in the performance.

To configure this parameter, a query must be executed directly in the database to determine the days after which this information will be purged. The table in question is tconfig and the field string_purge. To set 30 days for the purge of this type of information, the following query would be executed directly on the history database:

```
UPDATE tconfig SET VALUE = 30 WHERE token = "string_purge";
```

The database is maintained by a script called pandora_db.pl, to check that the database maintenance is executed correctly the maintenance script is executed manually:

```
/usr/share/pandora_server/util/pandora_db.pl /etc/pandora/pandora_server.conf
```

No debería reportar ningún error. Si otra instancia está utilizando la base de datos, bien puede utilizar la opción `-f` que fuerza la ejecución; con el parámetro `-p` no realiza un compactado de los datos. Esto es especialmente útil en entornos de Alta disponibilidad (HA) con base de datos histórica, ya que el script se asegura de realizar en un orden y modo correcto los pasos necesarios para dichos componentes.

Pandora FMS module statuses

When is each status established?

- On the one hand, each module has `WARNING` and `CRITICAL` thresholds in its configuration.
  - These thresholds define the values of your data for which these statuses will be activated.
  - If the module provides data outside these thresholds, it is considered to be in `NORMAL` status.
- Each module has, in addition, a time interval that will establish the frequency with which it will obtain the data.
  - This interval will be taken into account by the console to collect the data.
  - If the module has not collected data for twice its interval, the module is considered to be in `UNKNOWN` status.

- Finally, if the module has alerts configured and any of them has been triggered and has not been validated, the module will have the corresponding status of *Alert triggered*.

Propagation and priority

Within the Pandora FMS organization, certain elements depend on others, as is the case of the modules of an agent or the agents of a group. This can also be applied to the case of Pandora FMS policies, which have associated certain agents and certain modules that are considered associated to each agent.

Such a structure is especially useful for *evaluating module statuses at a glance*. This is achieved by propagating upwards in this organization the statuses, thus giving status to agents, groups and policies.

Which status will have an agent?

An agent will be shown with the *most critical status* of the statuses of its modules. In turn, a group will have the *most critical status* of the statuses of the agents that belong to it, and the same for the policies, which will have the *most critical status* status of their assigned agents.

Thus, *when you see a group with a critical status*, for example, you will know that at least one of its agents has the same status. To locate it, you will have to go down another level, to the agents' level, to narrow the circle and find the module or modules causing the propagation of this critical status.

What is the priority of the statuses?

As the *most critical status* of the statuses is propagated, it must be clear which statuses take priority over the others:

1. Alerts triggered.
2. Critical condition.
3. Warning status.
4. Status unknown.
5. Normal condition.

When a module has triggered alerts, its status has priority over all the others and the agent to which it belongs will have that status and the group to which that agent belongs, in turn, will also have that status.

On the other hand, for a group, for example, to have normal status, all its agents must have such

status; which means that all the modules of such groups will have normal status.

Color coding

Status of triggered alerts.

Critical Condition.

Warning Status.

Condition Unknown.

Normal condition.

Pandora FMS Charts

Graphs are one of the most complex implementations of Pandora FMS, they extract data in real time from the database and no external system (RRDtool or similar) is used.

There are several behaviors of the graphs depending on the type of source data:

- Asynchronous modules. It is assumed that there is no data compaction. The data in the database are all real samples of the data, there is no compaction. Produces much more “accurate” graphs with no possibility of misinterpretation.
- String type modules. They show graphs with the data rate over time.
- Numerical data modules. Most modules report this type of data.
- Boolean data modules. Correspond to numerical data on monitors, *PROC*: e.g. Ping checks, interface status, etc. The value 0 corresponds to the Critical status, and the value 1 to the “Normal” status.

Back to Pandora FMS Documentation Index