A SET OF SETS

If you've ever studied set theory in whatever mathematics class you have an idea about what sets can do. If you haven't, you might want to skip over this. However, I'll just say that sets are groups of unique elements that you can compare and operate on: find which elements are in two groups, in none of them, only in one or the other, etc. There are more advanced operations letting you define relations and operate on these relations and much more. I'm not going to dive into the theory (again, it's out of the scope of this book) so I'll just describe them as it is.

There are 4 main modules to deal with sets in Erlang. This is a bit weird at first, but it makes more sense once you realize that it's because it was agreed by implementers that there was no 'best' way to build a set. The four modules are ordsets, sets, gb_sets and sofs(sets of sets):

ordsets

> Ordsets are implemented as a sorted list. They're mainly useful for small sets, are the slowest kind of set, but they have the simplest and most readable representation of all sets. There are standard functions for them such as `ordsets:new/0`, `ordsets:is_element/2`, `ordsets:add_element/2`, `ordsets:del_element/2`,`ordsets:union/1`, `ordsets:intersection/1`, and a bunch more.

sets

> Sets (the module) is implemented on top of a structure really similar to the one used in `dict`. They implement the same interface as ordsets, but they're going to scale much better. Like dictionaries, they're especially good for read-intensive manipulations, like checking whether some element is part of the set or not.

gb_sets

> Gb_sets themselves are constructed above a General Balanced Tree structure similar to the one used in the gb_trees module. gb_sets are to sets what gb_tree is to dict; an implementation that is faster when considering operations different than reading, leaving you with more control. While gb_sets implement the same interface as sets and ordsets,

they also add more functions. Like gb_trees, you have smart vs. naive functions, iterators, quick access to the smallest and largest values, etc.

sofs

Sets of sets (sofs) are implemented with sorted lists, stuck inside a tuple with some metadata. They're the module to use if you want to have full control over relationships between sets, families, enforce set types, etc. They're really what you want if you need mathematics concept rather than 'just' groups of unique elements.

Don't drink too much kool-aid:

While such a variety can be seen as something great, some implementation details can be downright frustrating. As an example, gb_sets, ordsets and sofs all use the `==` operator to compare values: if you have the numbers `2` and `2.0`, they'll both end up seen as the same one. However, sets (the module) uses the `=:=` operator, which means you can't necessarily switch over every implementation as you wish. There are cases where you need one precise behavior and at that point, you might lose the benefit of having multiple implementations.

It's a bit confusing to have that many options available. Björn Gustavsson, from the Erlang/OTP team and programmer of Wings3D mainly suggests using gb_sets in most circumstances, using ordset when you need a clear representation that you want to process with your own code and 'sets' when you need the `=:=` operator (source.)

In any case, like for key-value stores, the best solution is usually to benchmark and see what fits your application better.

Source : http://learnyousomeerlang.com/a-short-visit-to-common-data-structures