Yggdrasil documentation

Temporal storage of simple objects

Overview of Yggdrasil

Terje Kvernes & David Ranvig

4th August 2008

Contents

1 What is Yggdrasil?

Yggdrasil aims to a "dynamic relational temporal object database". In essence, Yggdrasil aims to add two abstractions to the traditional view of a relational database: implicit temporal storage and a simple object model to represent the data stored. In addition to this Yggdrasil allows the relations of the entities stored within to be altered, and new entities and their relations to be added while the system is running. The relations are described by the administrator of the system and as soon as any relation is described to Yggdrasil, it is added to the overall structure of the installation.

In Yggdrasil lingo you can think of an entity as a "class" from the OO world. The "object" is an instance of this entity, each object existing in several temporal versions within the system.

Initializing Yggdrasil happens as follows, the parameters are all sent to the back end storage layer, and both their meaning and their necessity varies depending on said layer. Look at the documentation for Yggdrasil::Storage and its engines if in doubt. The "namespace" parameter defines which namespace will house the entities we'll create later. This is, in essence, your class hierarchy. You'll want to ensure it's uncluttered.

```
new Yggdrasil(
              user      => user,
              password  => password,
              host      => host,
              db        => databasename,
              engine    => engine,
              namespace => 'Ygg',
             );
```

Listing 1: Initializing Yggrasil

1.1 Objects

An object is an instance of an entity within the system. This instance is the primary working set that Yggdrasil operates on. Objects contain properties which are key / value pairs. Objects are identified by a unique name within each entity.

An object within Yggdrasil isn't a singular instance of grouped data. As any change to this object is kept, every version of the object throughout its existence is stored. The default object is the "current" object, defined as the set of properties that are currently active and not expired.

The limitations are currently bound to the objects being fairly simple, they are not allowed to store anything more complex than anything that can be mapped into a text field, and their only relations to other objects are the ones defined by the setup of Yggdrasil.

To define an entity "Host" within Yggdrasil. This will create access to the class "Ygg::Host", as we earlier defined the namespace for Yggdrasil to work in previously to be named "Ygg". We'll then create a "Host" object called "nommo" and one called "ninhursaga".

```
$hostclass = define Yggdrasil::Entity 'Host';
my $nommo = $hostclass->new( 'nommo' );
my $ninhursaga = Ygg::Host->new( 'ninhursaga' );
```

Listing 2: Defining entities

The return value from a "define" of "Yggdrasil::Entity" is the class the structure represents. It will always be "Namespace::Entityname", and using the return value lets you rely on Perls warnings and strict pragmas (assuming you use them) in case of a typo.

1.1.1 Properties

Depending on the way Yggdrasil is set up you may or may note have defined types for your properties, and you may or may not have constraints to the data stored for each property.

Yggdrasil can either be flexible and treat all property values as a default type, or you can select between a set of types Yggdrasil guarantees no matter the back end it's running on.

```
define Yggdrasil::Entity 'ip', 'Type' => 'IP';
define $hostclass 'comment';
```

Listing 3: Defining properties

1.2 Relations

All relations within Yggdrasil exist between entities. To create relations between entities, define them, but we'll need another entity to relate things to, since we have some hosts, let's make ourself some rooms.

```
# Create another entity, "Room".
my $roomclass = define Yggdrasil::Entity 'Room';
# Create a room object, 'b701'.
my $b701 = $roomclass->new( 'B701' );

# Now create the relation between rooms and hosts.
# $hostclass = 'Ygg::Host', $roomclass = 'Ygg::Room'
define Yggdrasil::Relation $hostclass, $roomclass;
```

Listing 4: Defining relations

This however doesn't do us much good, we need to link objects together, and we do that as follows:

```
# Now, links the host 'nommo' to the room 'b701'
$nommo->link( $b701 );
```

Listing 5: Linking "nommo" to "b701"

1.3 Temporal dimension

A principal idea of Yggdrasil is that data is only inserted, never deleted. This also means that there is a clear distinction between "available" and "current" data contained within the system. As long as no time frame, or slice, is requested, all requests work on the "current" dataset.

Deletion only happens if and only if the object structure ("entity" in Yggdrasil lingo) they belong to is deleted, and that deleted structure is "purged". Deletion of an entity is therefore, in some circumstances, reversible.

Changes in the structure will retain the information if possible, the system will inform you at any time if any action you take will permanently delete any data.

1.4 Dynamic storage

New entities and new relations between entities can be issued on the fly while the system is running live.