Redmine - Defect #37803

schema.rb dump/load does not preserve plugin migration version

2022-10-19 21:03 - crypto gopher

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:	Plugin API	**Estimated time:**	0.00 hour
Target version:			
Resolution:		**Affected version:**	5.0.2

Description

When dumping Redmine db schema to **.rb* file:

```
bundle exec rake db:schema:dump
```

current migration version is preserved for Redmine in *db/schema.rb* file:

```
# This file is auto-generated from the current state of the database. Instead
...

ActiveRecord::Schema.define(version: 2022_02_24_194639) do
   ...
```

Thanks to that, when schema is later loaded:

```
bundle exec rake db:schema:load
```

schema_migrations table is properly filled with applied migration versions and migration status stays in sync with database structure saved in *schema.rb*

This does not work for plugins.

When you dump+purge+load database schema containing plugin migrations, status of plugin migrations is irreversibly lost. The database structure contains all the tables/columns/etc. created by migrations, but schema_migrations table lacks entries corresponding to versions of these migrations. Whenever migration of plugin is later initiated, it will re-execute all migrations, starting from the first one, yielding errors if they were already applied before.

Solution

Extend dump/load mechanism with recording/restoring plugin migration versions:

```
module SchemaDumperPatch
  def define_params
    versions = super.present? ? [super] : []
    Redmine::Plugin.all.each do |plugin|
      versions << "#{plugin.id}: #{plugin.latest_migration}" if plugin.latest_migration
    end
    versions.join(", ")
  end
end

module SchemaPatch
  ActiveRecord::ConnectionAdapters::SchemaStatements.class_eval do
    def assume_plugin_migrated_upto_version(plugin_id, version)
      plugin = Redmine::Plugin.find(plugin_id)
      version = version.to_i

      migrated = Redmine::Plugin::Migrator.get_all_versions(plugin)
      versions = plugin.migrations
      inserting = (versions - migrated).select { |v| v <= version }
      if inserting.any?
        schema_migration.create_table
        execute insert_versions_sql(inserting.map! { |v| "#{v}-#{plugin_id}" })
      end
    end
  end

  # TODO: replace arguments with argument forwarding (info, ...) in Ruby 3.0
  def define(info, &block)
    super
    info.except(:version).each { |id, v| assume_plugin_migrated_upto_version(id, v) }
  end
end

ActiveRecord::Schema.prepend SchemaPatch
ActiveRecord::SchemaDumper.prepend SchemaDumperPatch
```

This is especially useful when running tests. When executing:

```
RAILS_ENV=test bundle exec rake redmine:plugins:test NAME=plugin_id
```

db:test:prepare rake task is executed as a preparation step, and causes db purge and subsequent schema load.