[TopCoder]

Acts_As_Id_Generator 1.0 Component Specification

1. Design

While Ruby on Rails provides a built-in mechanism to generate ids for primary keys, TopCoder's internal systems use table-driven id sequences through the Java Id Generator component. The purpose of this component is to provide a Rails plugin that will allow ActiveRecord models to easily generate their primary key ids from legacy Id Generator sequences.

1.1 Design Patterns

None

1.2 Industry Standards

None

1.3 Required Algorithms

1.3.1 How to create acts_as_id_generator plugin

Step 1
In rails app dir, run
jruby script/generate plugin acts_as_id_generator
It will generate necessary files for acts_as_id_generator plugin at dir
/vendor/plugins/acts_as_id_generator/

Step 2
Implement the init.rb and lib/acts_as_id_generator.rb in /vendor/plugins/acts_as_id_generator/ dir.

Step 3
Use the plugin

Reference to http://wiki.rubyonrails.org/rails/pages/HowTosPlugins for details.

1.3.2 Implementation Principles (acts_as_id_generator method)

We can use ActiveRecord::Base before_validation_on_create callback to generate the table driven id for the record. But we should use the macro style, like this:

```
ActiveRecord::Base.send(:before_validation_on_create, :generate_table_driven_id)
ActiveRecord::Base.class_eval do
  private
  def generate_table_driven_id()
    self.id = @@id_generator.next_id
  end
end
```

Also, we need to read id configuration from the database table. The table has following structure:

```
CREATE TABLE id_sequences (
        name VARCHAR(255) PRIMARY KEY,
        next_block_start DECIMAL(12,0) NOT NULL,
        block_size DECIMAL(10,0) NOT NULL,
        exhausted DECIMAL(1,0) NOT NULL DEFAULT 0
);
CREATE UNIQUE INDEX 166_166 ON id_sequences(name);
```

We can use ActiveRecord to read/update id_sequences table, remember the table name can be oter than “id_sequences”, it will can be specified as parameter table_name in acts_as_id_generator method. Like this:

```
id_sequence_class = Class.new(ActiveRecord::Base)
id_sequence_class.set_table_name(table_name)
id_sequence_class.set_primary_key(“name”)
```

1.3.3 How to get next id (IdGenerator.next_id method)

To simplify the logic, the exception handling is not included in the flowing code

```
@mutex.synchronize do
  if @id_left <= 0 #if there is no ids left for current block
    record  = @id_sequence_class.find(:first,
        :conditions => [ "name = ? AND  exhausted = 0", @sequence_name ],
        :lock => true) # This also lock the sequence row in database
    @next_id = record.next_block_start
    @id_left = record.block_size
    record.next_block_start += record.block_size
    # we should also check whether this sequence should be exhausted
      if record.next_block_start >= 10^12
          record.exhausted = 1
          record.next_block_start = -1
      end
     record.save
  end
  @id_left -= 1 #decrease the id_left
  @next_id += 1 #increase the next_id
end
return @next_id - 1
```

1.4 Component Class Overview

TopCoder::Acts::IdGenerator::ClassMethods [module]

This is the module which will be extended by ActiveRecord::Base

TopCoder::Acts::IdGenerator::IdGenerator[class]

This class is the class to generate table driven id.

It uses a dynamic class which extends ActiveRecord::Base to read/modify sequence table in database.

It uses mutex to lock the next_id method, it also uses ActiveRecord::Locking::Pessimistic to lock the appropriate sequence row.

1.5 Component Exception Definitions

ArgumentError [System]:

This exception is thrown if argument is invalid in this component.

ExhaustedSequenceError

This exception will be thrown if the specified sequence is exhausted.

MissingSequenceError
This exception will be thrown if the specified sequence is not found.

MissingTableError
This exception will be thrown if the specified sequence table is not found.

1.6 Thread Safety

This component is not completely thread safe. But it can be used in thread-safe way.
The IdGenerator and Errors have mutable properties, but they will not be set in multiple threads.
The method next_id is synchronized which is the only method used in multiple threads, so this component can be used in thread-safe way.

2. Environment Requirements

2.1 Environment

Development language: Ruby
Compile target: JRuby 1.1.5
QA Environment:
Ruby on Rails 2.1.2
ActiveRecord JDBC Adapter with Informix support
Informix 10
Informix JDBC Adapter 3.00.JC3
Sun JDK 1.6.0_10

2.2 TopCoder Software Components

None

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

TopCoder::Acts::IdGenerator

3.2 Configuration Parameters

None

3.3 Dependencies Configuration

None

4. Usage Notes

4.1 Required steps to test the component

None

4.2 Required steps to use the component

Please see the demo.

4.3 Demo

1. Set up database first

```
CREATE TABLE id_sequences (
    name VARCHAR(255) PRIMARY KEY,
    next_block_start DECIMAL(12,0) NOT NULL,
```

```
    block_size DECIMAL(10,0) NOT NULL,
    exhausted DECIMAL(1,0) NOT NULL DEFAULT 0
);

CREATE UNIQUE INDEX 166_166 ON id_sequences(name);
```

2. Define AcitveRecord::Base's subclasses

```
#Assume we have a table foos
#Also we assume there is one row in id_sequences table
#It is (name=>"foo_seq", next_block_start=>1, block_size=>10, exhausted=>0)
class Foo < ActiveRecord::Base
        acts_as_id_generator :sequence=>"foo_seq", :table=>"id_sequences"
        #the table is optional here, as its default value is "id_sequences"
        #the default value of sequence name is "foos_seq"
end

#create 20 foos
20.times do |i|
        foo = Foo.new
        foo.save
end

#The generated ids for foos should be from 1 to 20
#The foo_seq row will be  (name=>"foo_seq", next_block_start=>21, block_size=>10, exhausted=>0)
```

5. Future Enhancements

The component may be extended to allow distribution as a Ruby Gem rather than using the Rails plugin system.