

Active Warehouse

Marty Haught

mghaught@gmail.com

<http://martyhaught.com>

What is Active Warehouse?

- Rails plugin by Anthony Eden
- Goal to simplify the building of data warehouses in Rails
- Based on techniques from The Data Warehouse Toolkit by Ralph Kimball
- Uses ActiveRecord under the hood

Features

- Generators for Facts, Dimensions, Cubes and Bridges
- Supports calculated fields
- ETL tool (Extract Transform Load)
- View helpers for reports with drill down

Getting Started

- Start with an existing Rails application
- Install the plugin:

```
marty$ script/plugin install svn://rubyforge.org/var/svn/activewarehouse/activewarehouse/trunk
```

Installing ETL

```
marty$ sudo gem install activewarehouse-etl -y
```

- Requires the following gems:
 - ActiveSupport
 - ActiveRecord
 - FasterCSV
 - AdapterExtensions
- Optional - only install if you need the ETL

Database Config

```
development:  
  adapter: mysql  
  database: dw_demo_dev  
  username: root  
  password:  
  host: localhost  
  
etl_execution:  
  adapter: mysql  
  database: etl_execution  
  username: root  
  password:  
  host: localhost  
  encoding: utf8  
  
oltp_development:  
  adapter: mysql  
  database: dw_oltp_dev  
  username: root  
  password:  
  host: localhost
```

Dimensions

- Location Dimension
 - state, city, zip code, retailer, store
- Date Dimension
 - year, quarter, month, day

```
marty$ script/generate dimension location  
marty$ script/generate dimension date
```

Dimension Declarations

```
class LocationDimension < ActiveWarehouse::Dimension
  define_hierarchy :zip_code, [:zip_code, :store]
  define_hierarchy :state, [:state, :city, :store]
  define_hierarchy :retailer, [:retailer, :city, :store]
end
```

- Each hierarchy has several levels
- Array determines order of levels
- Each level will have a set of values

Dimension Migration

```
class CreateLocationDimension < ActiveRecord::Migration
  def self.up
    create_table :location_dimension do |t|
      t.column :store_id, :integer
      t.column :store, :string
      t.column :retailer, :string
      t.column :city, :string
      t.column :state, :string
      t.column :zip_code, :string
    end
    add_index :location_dimension, :store_id
    add_index :location_dimension, :store
    add_index :location_dimension, :retailer
    add_index :location_dimension, :city
    add_index :location_dimension, :state
    add_index :location_dimension, :zip_code
  end

  def self.down
    drop_table :location_dimension
  end
end
```

Facts

- Sales Fact
 - number of sales per day
 - total amount of sales per day
 - average amount of sales per day

```
marty$ script/generate fact sales
```

Fact Declarations

```
class SalesFact < ActiveWarehouse::Fact
  aggregate :id, :type => :count, :label => "Purchases"
  aggregate :sales_amount, :label => "Total Amount"
  aggregate :sales_amount, :type => :avg, :label => "Average"

  dimension :location
  dimension :date

  belongs_to :date, :class_name => "DateDimension", :foreign_key => "date_id"
  belongs_to :location, :class_name => "LocationDimension", :foreign_key => "location_id"
end
```

- Aggregates default to a sum calculation
- Can do other calculations such as average, standard deviation, etc.

Fact Migration

```
class CreateSalesFacts < ActiveRecord::Migration
  def self.up
    create_table :sales_facts do |t|
      t.column :sales_id, :integer
      t.column :sales_amount, :decimal, :precision => 8, :scale => 2
      t.column :date_id, :integer
      t.column :location_id, :integer
    end
    add_index :sales_facts, :sales_id
    add_index :sales_facts, :date_id
    add_index :sales_facts, :location_id
  end

  def self.down
    drop_table :sales_facts
  end
end
```

Cubes

- `date_location_cube`
 - joins the Date and Location dimensions

```
marty$ script/generate cube date_location
```

Cubes

```
class DateLocationCube < ActiveWarehouse::Cube
  reports_on :sales
  pivots_on :date, :location
end
```

- Allows you to dive deeper into the data through your hierarchies

Using the ETL

- Transforms data from one source into *AW*
- Provides a lot of flexibility; solved our issues
- Not well-documented

** Will not go deeply into ETL

Parts of the ETL

- Source
- Transform
- Destination
- Post Processing

Source Directive

- Flat File
 - Delimited Parser (csv)
 - Fixed-width
 - SAX
 - Apache Combined Logs
- Database
- Enumerable

Transform Directive

- **Field-based**
 - decode
 - date-to-string
 - default
 - Block form with name, value and row references
- **Row-based**
 - check for existing row
 - check for uniqueness
 - copy field

Destination Directive

- File
- Database

Post Process Directive

- Bulk Import
- Encode

Running ETL

- Make a directory for etl related files
- Create ctl files for each dimension and fact
- Execute with the etl command:

```
marty$ etl sales_fact.ctl -c ../config/database.yml
```

- Be sensitive to dependencies in load order
- Use the -c command to reference your database config

ActiveWarehouse::Report::TableReport

- requires:
 - cube_name
 - column_dimension_name
 - row_dimension_name

```
report_params = {
  :title => "Sales Report",
  :cube_name => :date_location_cube,
  :column_dimension_name => :date,
  :row_dimension_name => :location,

  :format => {
    :sales_amount => :currency
  }
}

@report = ActiveWarehouse::Report::TableReport.new(report_params)
@view = @report.view(params)
```

TableReport Config

- :title - names report
- :column_hierarchy - specifies hierarchy
- :row_hierarchy
- :format - sets format on value per attribute
- :fact_attributes - limits attributes from fact
- :column_filters - eliminates hierarchy values
- :conditions - additional sql AND clause

TableView Use

```
@view = @report.view(params, :with_totals => true)
```

- `TableReport.view(params, options)`
 - Params - HTML params via Controller
 - Options:
 - `sortable`
 - `with_totals`
 - `sortable_with_totals` (convenience option)
 - `ignore_columns` (only with totals)

Rendering Reports

- `render_report_from(table_view)`
- `render_crumbs(dimension_crumbs)`

```
<p><%= render_crumbs(@view.column_crumbs) %></p>  
<p><%= render_crumbs(@view.row_crumbs) %></p>  
  
<%= render_report_from(@view) %><br/>
```

Sorting with YUI

- enables client-side sortable tables
- install plugin:

```
marty$ script/plugin install svn://rubyforge.org/var/svn/yui4rails/trunk
```

- run initialize_yui task from plugin:

```
/vendor/plugins/yui4rails marty$ rake initialize_yui
```

- include yui resources on require pages:

```
<%= include_yui :datatable %>
```

- set the skin:

```
<body class="yui-skin-sam">
```

Rails SQL Views

- Gem by Anthony Eden
- Gives the ability to create or drop views via migrations
- Works with MySQL, PostgreSQL and SQL server

```
create_view :v_people, "select * from people" do |t|
  t.column :id
  t.column :name
  t.column :social_security
end
```

Resources

- Website: <http://activewarehouse.rubyforge.org/>
- Books: The Data Warehouse Toolkit by Ralph Kimball
- Mailing List: (via RubyForge project)
- Anthony Eden's blog: <http://anthonyeden.com/>

mghaught@gmail.com
<http://martyhaught.com>