EAV Antipattern Database Structure

by

Tom Anderson

EAV Tables

EAV Splits traditional table attributes into atomic tables

Columns

Rows	Value	Value	Value
	Value	Value	Value
	Value	Value	Value
	Value	Value	Value

Attributes

Entity	Value	Value	Value
	Value	Value	Value
	Value	Value	Value
	Value	Value	Value

But attributes must be grouped in order to
make sense of table data

EAV Tables

Attribute Sets Table

attribute_sets
attribute_set_key - Primary key
ref_user - The user that owns the attribute set.
name - The name of the attribute set

EAV Tables

Attributes Table

Attribute tables contain column descriptions

attributes
attribute_key - Primary Key
ref_renderer - What renderer does this attribute use?
name - The name of this attribute
is_required - Is the attribute required?
is_unique - Should values of this attribute be unique across all instances of this attribute?

EAV Tables

Renderers Table

renderers
renderer_key - Primary key
name - The renderer name such as URL, Date or Text
model - The PHP class to handle rendering a form input element and formatting the value. This is an object dictionary.

And Datatype: one of datetime, decimal, int, text or varchar stored in each model.

EAV Tables

Renderer Model

```php
<?php

class Renderer_Integer extends Zend_Form_Element_Text implements Collections_Renderer
{
    public $icon = '13.png';
    public $datatype = 'int';

    public function postCreate() {
        $this->addValidator('Int');
    }

    public function formatValue($value) {
        return $value;
    }

    // Preformat value when editing such as date times > dates
    public function formatEditValue($value) {
        return $value;
    }

}
```

EAV Tables

Values Tables

items_datetime, items_decimal, items_int, items_text, or items_varchar
value_key
ref_attribute - Which attribute does this value implement?
ref_item - Reference to the entity
value - A column type reflecting the table name

EAV encourages no particular way to store values so many ways have been concocted. Many EAV implementations incorrectly use a single value table with a value column of type text. This limits you and doesn't apply the flexibility of datatypes available in a database. Magento got it right and uses five value tables for each entity table and I mimic their work.

EAV Tables

Items Table

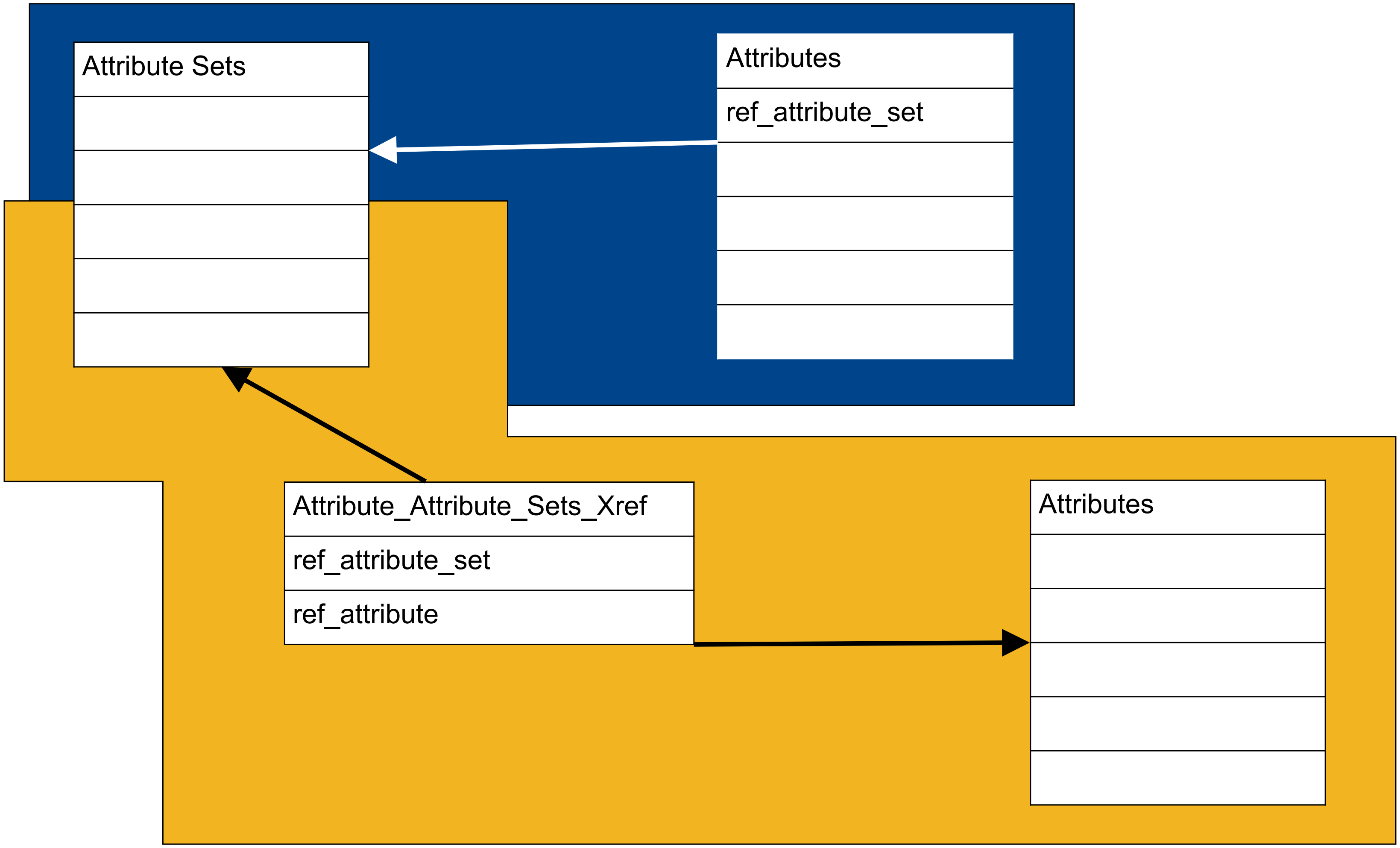
Entity tables are named after the data it represents e.g. **Items**

Items
items_key - Primary Key
ref_attribute_set - Which attribute set does this item implement?
ref_item - Does this item have a parent item?
ref_user - The user that created the item.

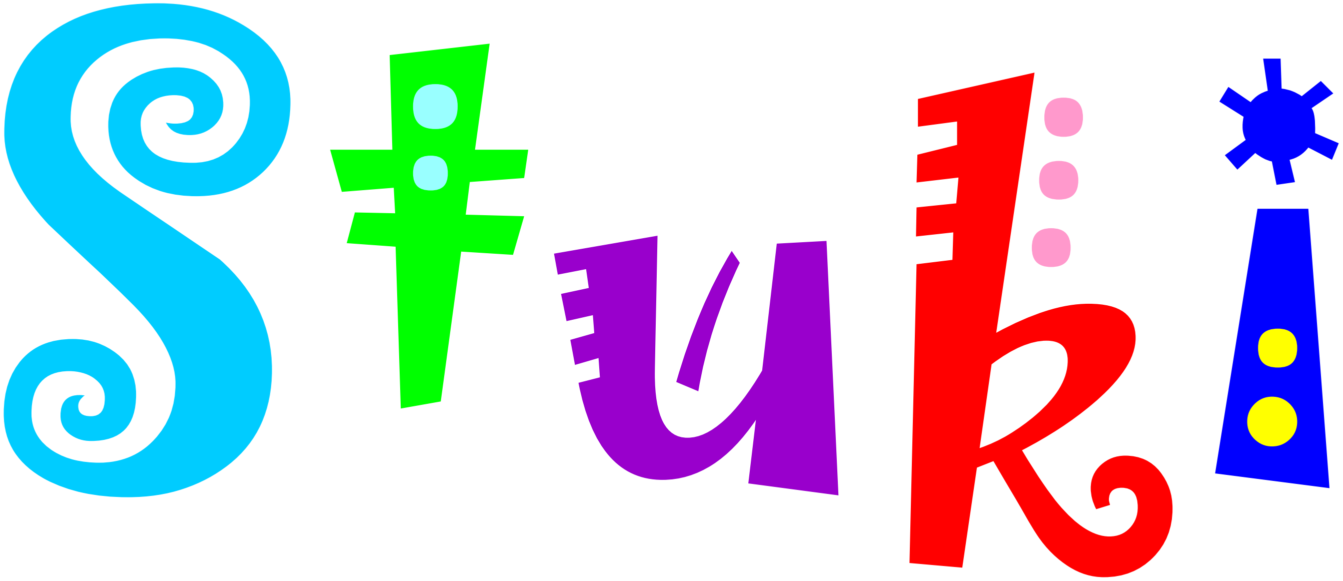
EAV Tables

Attributes - Attributes Sets relationship

Attribute sets may have a many to many or one to many relationship with attributes



Introducing



An MVC & EAV, Domain Model, 100% OO, Collections Engine

Advantages of Stuki over Omeka

Stuki and Omeka were designed from the same epiphany: that EAV tables could store collections of items with dynamic attributes.

Stuki

Includes Dublin Core Metadata Elements
Peer Reviewed Edits
Pictures
Full Table Auditing
Watchlists
Merge Requests
HTML Purifier
Ownership Records & User Ownership

17 Renderers
18 Plugins

Plugins Include:

- Dublin Core COinS
- Favorites
- Footnotes
- Forums
- Intense Debate
- Library of Congress Subject Headings
- PDF Report with URL Barcodes
- Proxy Email
- Social Bookmarks
- Wantlists

Omeka

Includes Dublin Core Metadata Elements
Files
Tags

31 plugins

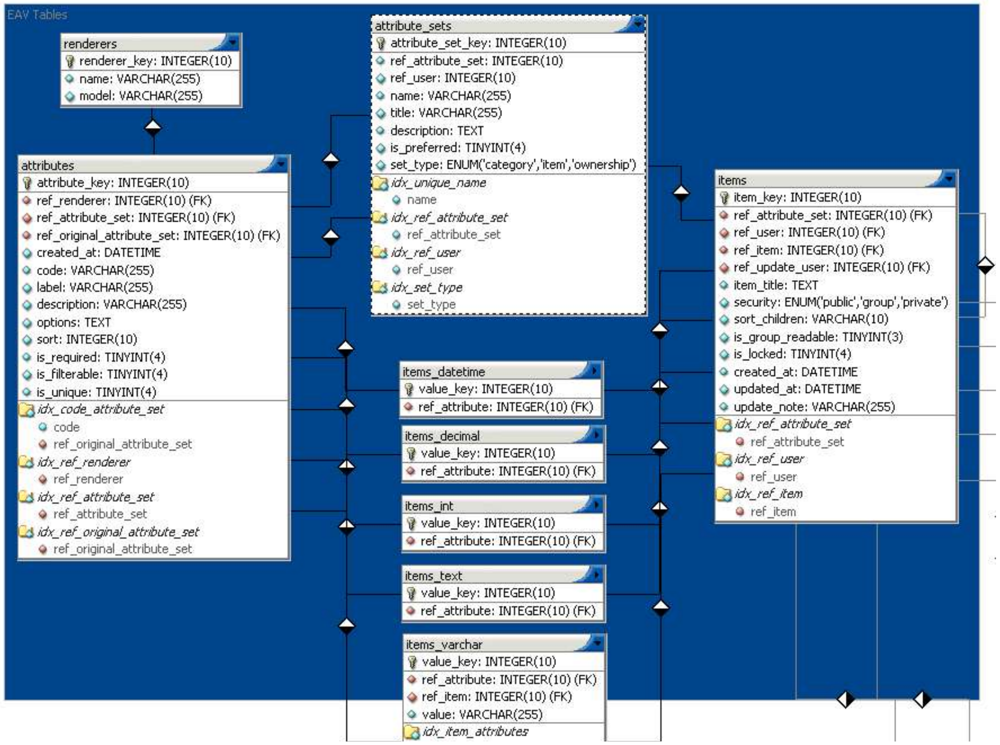
Including:

- **Bar Codes and Reports**
- **Docs Viewer**
- **Dublin Core COinS**
- **HTML Purifier**
- **Intense Debate**
- **Library of Congress Subject Headings**
- **Social Bookmarks**

Advantages of Stuki over Omeka

Stuki has precise, well named EAV tables and values are stored in their appropriate data type following Magento's example.

Stuki



EAV Tables

renderers
renderer_key: INTEGER(10)
name: VARCHAR(255)
model: VARCHAR(255)

attributes
attribute_key: INTEGER(10)
ref_renderer: INTEGER(10) (FK)
ref_attribute_set: INTEGER(10) (FK)
ref_original_attribute_set: INTEGER(10) (FK)
created_at: DATETIME
code: VARCHAR(255)
label: VARCHAR(255)
description: VARCHAR(255)
options: TEXT
sort: INTEGER(10)
is_required: TINYINT(4)
is_filterable: TINYINT(4)
is_unique: TINYINT(4)
idx_code_attribute_set
code
ref_original_attribute_set
idx_ref_renderer
ref_renderer
idx_ref_attribute_set
ref_attribute_set
idx_ref_original_attribute_set
ref_original_attribute_set

attribute_sets
attribute_set_key: INTEGER(10)
ref_attribute_set: INTEGER(10)
ref_user: INTEGER(10)
name: VARCHAR(255)
title: VARCHAR(255)
description: TEXT
is_preferred: TINYINT(4)
set_type: ENUM('category','item','ownership')
idx_unique_name
name
idx_ref_attribute_set
ref_attribute_set
idx_ref_user
ref_user
idx_set_type
set_type

items_datetime
value_key: INTEGER(10)
ref_attribute: INTEGER(10) (FK)

items_decimal
value_key: INTEGER(10)
ref_attribute: INTEGER(10) (FK)

items_int
value_key: INTEGER(10)
ref_attribute: INTEGER(10) (FK)

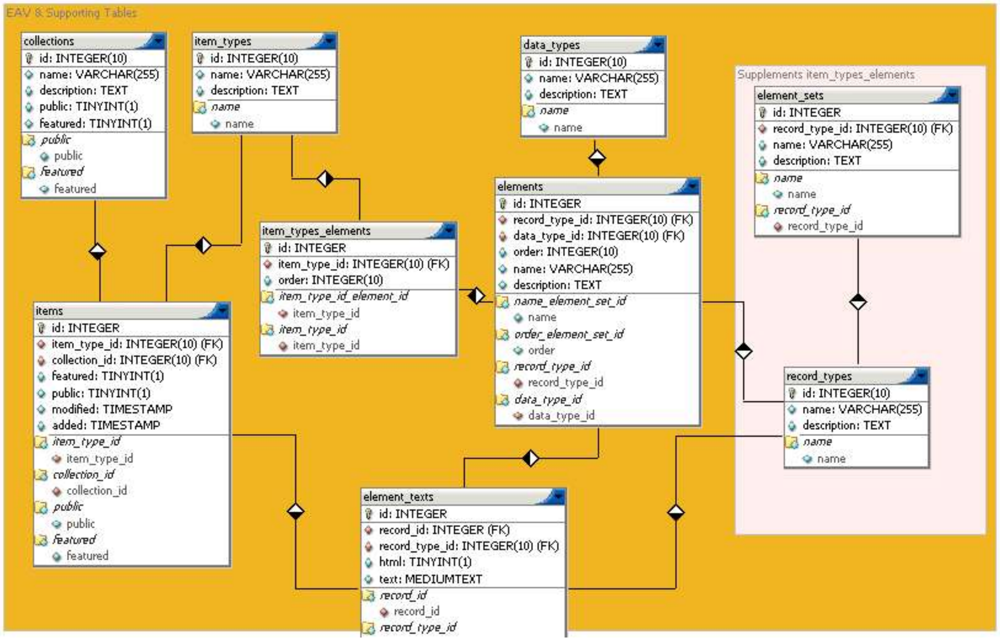
items_text
value_key: INTEGER(10)
ref_attribute: INTEGER(10) (FK)

items_varchar
value_key: INTEGER(10)
ref_attribute: INTEGER(10) (FK)
ref_item: INTEGER(10) (FK)
value: VARCHAR(255)
idx_item_attributes

items
item_key: INTEGER(10)
ref_attribute_set: INTEGER(10) (FK)
ref_user: INTEGER(10) (FK)
ref_item: INTEGER(10) (FK)
ref_update_user: INTEGER(10) (FK)
item_title: TEXT
security: ENUM('public','group','private')
sort_children: VARCHAR(10)
is_group_readable: TINYINT(3)
is_locked: TINYINT(4)
created_at: DATETIME
updated_at: DATETIME
update_note: VARCHAR(255)
idx_ref_attribute_set
ref_attribute_set
idx_ref_user
ref_user
idx_ref_item
ref_item

- attribute_sets is similar to item_types_elements.
- attributes is similar to elements
- renderers is similar to data_types but renderers stores the class each type uses to render it's form element and value. Omeka uses data_types values in code throughout the application.
- item_types has no correlation because the 'type' is defined in attribute_sets.name & description.
- In Stuki a collection is stored like any other item giving them the same flexibility. No collections table is necessary.
- element_texts.record_type_id duplicates elements.record_type_id.

Omeka



EAV & Supporting Tables

collections
id: INTEGER(10)
name: VARCHAR(255)
description: TEXT
public: TINYINT(1)
featured: TINYINT(1)
public
public
featured
featured

item_types
id: INTEGER(10)
name: VARCHAR(255)
description: TEXT
name
name

data_types
id: INTEGER(10)
name: VARCHAR(255)
description: TEXT
name
name

Supplements item_types_elements

element_sets
id: INTEGER
record_type_id: INTEGER(10) (FK)
name: VARCHAR(255)
description: TEXT
name
name
record_type_id
record_type_id

elements
id: INTEGER
record_type_id: INTEGER(10) (FK)
data_type_id: INTEGER(10) (FK)
order: INTEGER(10)
name: VARCHAR(255)
description: TEXT
name_element_set_id
name
order_element_set_id
order
record_type_id
record_type_id
data_type_id
data_type_id

item_types_elements
id: INTEGER
item_type_id: INTEGER(10) (FK)
order: INTEGER(10)
item_type_id_element_id
item_type_id
item_type_id
item_type_id

items
id: INTEGER
item_type_id: INTEGER(10) (FK)
collection_id: INTEGER(10) (FK)
featured: TINYINT(1)
public: TINYINT(1)
modified: TIMESTAMP
added: TIMESTAMP
item_type_id
item_type_id
collection_id
collection_id
public
public
featured
featured

record_types
id: INTEGER(10)
name: VARCHAR(255)
description: TEXT
name
name

element_texts
id: INTEGER
record_id: INTEGER (FK)
record_type_id: INTEGER(10) (FK)
html: TINYINT(1)
text: MEDIUMTEXT
record_id
record_id
record_type_id

- Items is similar to Items but Stuki includes enhanced security, how to sort child elements, a lock, and an update note used in auditing changes.
- MySQL only recognizes one TIMESTAMP, not two per table like items.
- element_texts is analogous to items_datetime, items_decimal, items_int, items_text, and items_varchar. The significant difference is Stuki stores every value in it's appropriate datatype inside the database. Omeka stores everything in a mediumtext field.
- Please note element_texts.record_id references items.id. This is not represented in the schema and must be divined.
- element_sets and record_types have no analog in Stuki. They were added into Omeka when other tables were removed.
- In Stuki the attribute references the attribute set. In Omeka there is a many to many relationship between item_types and elements. Stuki's simplistic approach means attributes cannot be reused between attribute sets but creating new attributes is trivial.
- In Omeka collections are static objects which must be enhanced with code. In Stuki categories, items, regular items, or ownership records are all stored in the items table.

Advantages of Stuki over Omeka

Searching

Stuki

Scalable, High-Performance Indexing

* over 20MB/minute on Pentium M 1.5GHz
* small RAM requirements -- only 1MB heap
* incremental indexing as fast as batch indexing
* index size roughly 20-30% the size of text indexed

Powerful, Accurate and Efficient Search Algorithms

* ranked searching -- best results returned first
* many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
* fielded searching (e.g., title, author, contents)
* date-range searching
* sorting by any field
* multiple-index searching with merged results
* allows simultaneous update and searching

http://lucene.apache.org/java/docs/features.html

Omeka

MySQL Full Text Search

Example:

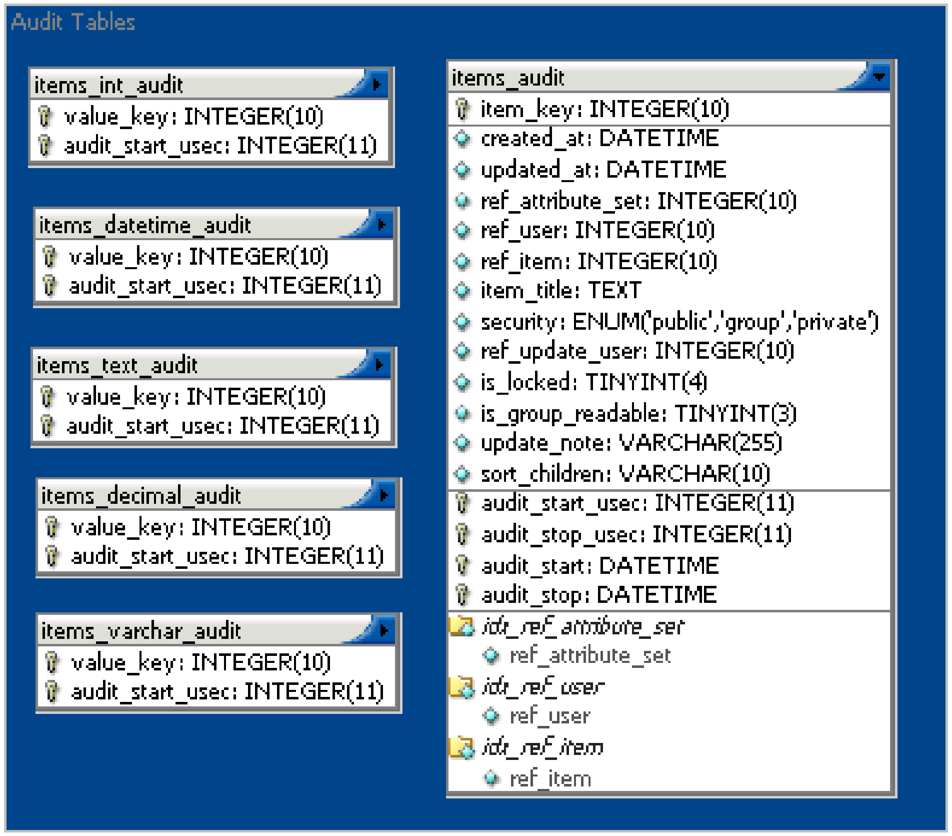
```
SELECT i.id as item_id,
MATCH (etx.text) AGAINST ('test test') as rank
FROM ...
```

Note Omeka does not use boolean mode thereby reducing the user’s options when making searches.

Advantages of Stuki over Omeka

Stuki has full items auditing extending Joe Celko’s approach in SQL For Smarties by adding a microtime stamp to each change.

Stuki



Audit Tables

items_int_audit
- value_key: INTEGER(10)
- audit_start_usec: INTEGER(11)

items_datetime_audit
- value_key: INTEGER(10)
- audit_start_usec: INTEGER(11)

items_text_audit
- value_key: INTEGER(10)
- audit_start_usec: INTEGER(11)

items_decimal_audit
- value_key: INTEGER(10)
- audit_start_usec: INTEGER(11)

items_varchar_audit
- value_key: INTEGER(10)
- audit_start_usec: INTEGER(11)

items_audit
- item_key: INTEGER(10)
- created_at: DATETIME
- updated_at: DATETIME
- ref_attribute_set: INTEGER(10)
- ref_user: INTEGER(10)
- ref_item: INTEGER(10)
- item_title: TEXT
- security: ENUM('public','group','private')
- ref_update_user: INTEGER(10)
- is_locked: TINYINT(4)
- is_group_readable: TINYINT(3)
- update_note: VARCHAR(255)
- sort_children: VARCHAR(10)
- audit_start_usec: INTEGER(11)
- audit_stop_usec: INTEGER(11)
- audit_start: DATETIME
- audit_stop: DATETIME
- *idx_ref_attribute_set*
  - ref_attribute_set
- *idx_ref_user*
  - ref_user
- *idx_ref_item*
  - ref_item

Note the items tables are the only audited tables in Stuki at this time. However, Stuki is crippled at it’s current host and triggers cannot be added to all other significant tables in the database.
Adding these triggers is a day’s job.

Omeka

No Auditing

Advantages of Stuki over Omeka

Final Thoughts

Stuki was developed with no knowledge of Omeka.

Stuki has been in development by a solo developer since July 2010. Omeka has been in development for the past two years. Omeka has had a longer period and more eyes for testing and development than Stuki.

The database design and choices made about how to implement the functionality which makes this sort of application applicable are superior in the Stuki. For instance, the choice to create unique attributes for each definition in Stuki is a stronger position than creating attributes and assigning them to definitions, forcing the user to select from a long list of attributes and giving them no preview of what their selection looks like. Stuki considered this approach and decided it was too confusing and complicated the database structure. Stuki, instead, uses unique attributes for each definition thereby making the user choose only the data type and giving them a preview of each as they choose. In turn this streamlined the database structure into a single reference from attributes to attribute types.

Additionally Stuki took it's EAV design from Magento, the undisputed leader in e-commerse applications today. The Magento team chose to store values in five tables so the data types of each could be directly used. This makes it impossible to change a to a different datatype renderer once data exists which uses a definition.

The design of renderers which use specific Zend_Form_Element definitions is far superior to Omeka's use of data_types, the values of which are scattered, hard coded, throughout the application.

In Stuki, every dynamic piece of information is stored in the ini file. There are 2 DEFINE's in Stuki as recommended by Zend. There are no global variables in Stuki. paths.php defines 39 variables essentially used as global variables throughout the application.

Omeka uses global functions thereby breaking a true object oriented approach. Stuki has no globals or global functions and is a true 100% object oriented application.

Advantages of Stuki over Omeka

Thoughts on Modeling

Omeka defines base directories so theoretically you could move your directories to different machines or drives. This is unnecessary in code intended for Unix installations since mapping devices and drives to a directory structure is simplistic. Windows provides analogous pathways.

Labeled “helpers” in Omeka, the application loads file after file of global functions(1). This approach to PHP programming lost favour with PHP 4. Fully OO programming has been preferred in every OO language since java's rise to popularity. PHP supports fully OO applications and frameworks like Zend are the preferred approach today. Omeka is a hybrid of structural programming with OO libraries.

Omeka uses a Domain Model architecture (Active Record). This approach is widely accepted: “…the diffused opinion is that the more complex the business logic and the data involved, the more the application benefits from a rich Domain Model.”(2) I argue the simplistic nature of EAV tables doesn't warrant the use of such a structured model. Leaving each model to make SQL queries as they see fit it imbues flexibility in the core of the application.

Omeka takes the Domain Model in directions it's not designed. The ancestor Omeka_Controller_Action is assigned a table which that controller controls. This irreparably breaks the MVC design because it is establishing business logic on the controllers. An MVC application should be scriptable from the command line with no use of controllers.

Stuki uses business logic models which in turn use each other. The loading of models in Stuki is done with $items = Collections_ModelBroker::get('Items'); The model broker then creates a new Collections_Model_Wrapper(new $modelname); This wrapper and it's enclosed model are then placed in static memory so they are only created once (unless a model's flagged property to always recreate is true).

The Collections_Model_Wrapper allows plugins to wrap around a specific model function and override the function completely or change parameters sent to or received from the function. The mechanism is tested but I have not created a plugin which takes advantage of this; yet but the overhead is trivial. Plugins are generally satisfied with access to the Smarty template before it's displayed.

Plugins are modelled from Zend_Controller_Action_Helper and all plugins are descended from Zend_Controller_Action_Helper_Abstract(3). These are not related to Omeka's “helpers”. See reference article.

Stuki uses and loads models on an as-needed basis. No preloading of 'mixins' such as Omeka_Record_Mixin which does pseudo-multiple inheritance. I'm not a proponent of multiple inheritance.

1: ~/application/helpers/all.php
2: http://css.dzone.com/books/practical-php-patterns/practical-php-patterns-domain
3: http://devzone.zend.com/article/3372

EAV Antipattern Database Structure

by

Tom Anderson