

# MyOwnDB presentation

Raphaël Bauduin

25 october 2008 - T-Dose

## Introduction

Origins

Features

Foundations

## Data Structure

Concepts

## Customisation

Data types

Localisation

Rails engines

## Customisation

- Data types
- Localisation
- Rails engines

## Openness

- REST API
- Export facilities
- Open Source

## Future

- Development
- Project

## Enf of presentation

- Questions?

# Origins

- ▶ experiment started march in 2005, after FOSDEM, with the goals to have a web administered database system with
  - ▶ extreme flexibility
  
- ▶ first available in march 2006 (always after FOSDEM)
- ▶ joined forces with Zeropoint.IT in november 2007
- ▶ publication of source code under an OSI license in july 2008

# Origins

- ▶ experiment started march in 2005, after FOSDEM, with the goals to have a web administered database system with
  - ▶ extreme flexibility
  - ▶ **easy extensibility**
- ▶ first available in march 2006 (always after FOSDEM)
- ▶ joined forces with Zeropoint.IT in november 2007
- ▶ publication of source code under an OSI license in july 2008

# Origins

- ▶ experiment started march in 2005, after FOSDEM, with the goals to have a web administered database system with
  - ▶ extreme flexibility
  - ▶ easy extensibility
  - ▶ **complex data types**
- ▶ first available in march 2006 (always after FOSDEM)
- ▶ joined forces with Zeropoint.IT in november 2007
- ▶ publication of source code under an OSI license in july 2008

# Origins

- ▶ experiment started march in 2005, after FOSDEM, with the goals to have a web administered database system with
  - ▶ extreme flexibility
  - ▶ easy extensibility
  - ▶ complex data types
  - ▶ **easy administration**
- ▶ first available in march 2006 (always after FOSDEM)
- ▶ joined forces with Zeropoint.IT in november 2007
- ▶ publication of source code under an OSI license in july 2008

# Origins

- ▶ experiment started march in 2005, after FOSDEM, with the goals to have a web administered database system with
  - ▶ extreme flexibility
  - ▶ easy extensibility
  - ▶ complex data types
  - ▶ easy administration
  - ▶ **localised user interface**
- ▶ first available in march 2006 (always after FOSDEM)
- ▶ joined forces with Zeropoint.IT in november 2007
- ▶ publication of source code under an OSI license in july 2008

## So, *what* is MyOwnDB

### Web database providing

- ▶ web-based management and end-users tools
- ▶ multi-lingual
- ▶ UTF-8 support at all levels
- ▶ easy information management: no need to create primary keys or tables to store Many to Many relations.
- ▶ advanced data types such as file attachments, URLs, email addresses, . . .
- ▶ export to a standard RDBMS
- ▶ a complete API to integrate the data store in your applications

## Built on Free and Open Source

Postgresql is the RDBMS storing all the data

Ruby on Rails is the web application framework used

YUI is the javascript framework we chose for the web interface

## Underlying database

### Definition zone

entities ↔ table

details ↔ column

relations ↔ constraint or  
many to many  
relation table

### Instanciation zone

instances ↔ record

detail\_values ↔ column's value

links ↔ record in a many  
to many relation  
table

## Example

id	name	first_name	age	country_id
1	Torvalds	Linus	38	34

## Example

id	name	first_name	age	country_id
1	Torvalds	Linus	38	34

The definition of the table is stored in tables:

- ▶ entities
- ▶ details
- ▶ data types
- ▶ relations

## Example

id	name	first_name	age	country_id
1	Torvalds	Linus	38	34

The definition of the table is stored in tables:

- ▶ entities
- ▶ details
- ▶ data types
- ▶ relations

And entries are stored in other tables:

- ▶ instances
- ▶ detail values
- ▶ detail value propositions
- ▶ links

## Postgresql to the rescue

tablefunc : contrib module reassembling virtual tables split over multiple tables

## Postgresql to the rescue

tablefunc : contrib module reassembling virtual tables split over multiple tables

- ▶ Used to display tables in the web interface
- ▶ Unusable in the code for the REST API

## Advantages

- ▶ Consistency: same approach, whatever the datatype
- ▶ Flexibility: easy to get at each data element individually
- ▶ Extensibility

## Data types

Defining a new data type is writing a Ruby class inheriting  
DetailValue and implementing

- ▶ `to_form_row`
- ▶ `self.format_detail`
- ▶ `self.valid?`

## Data types

Defining a new data type is writing a Ruby class inheriting DetailValue and implementing

- ▶ to\_form\_row
- ▶ self.format\_detail
- ▶ self.valid?

Data types can be very simple: date\_detail\_value

## Data types



Defining a new data type is writing a Ruby class inheriting `DetailValue` and implementing

- ▶ `to_form_row`
- ▶ `self.format_detail`
- ▶ `self.valid?`








Data types can be very simple: `date_detail_value` or more complex: store a file on S3

You can also overwrite all methods provided by `ActiveRecord`, as is done in `ddl_detail_value` to implement the “choice in a list” data type

## List formatting shot

Filter on field  with value corresponding to     

Page:     (100 items)

Name	Website	Email	View
Dryades	<a href="http://www.ylu.net">http://www.ylu.net</a>	<a href="mailto:info@ylu.net">info@ylu.net</a>	
Orthrus	<a href="http://www.Gourmetstores.Com">http://www.Gourmetstores.Com</a>	<a href="mailto:info@Gourmetstores.Com">info@Gourmetstores.Com</a>	
Zetes	<a href="http://www.Basketsbypatty.Com">http://www.Basketsbypatty.Com</a>	<a href="mailto:info@Basketsbypatty.Com">info@Basketsbypatty.Com</a>	
Nereids	<a href="http://www.vtj.net">http://www.vtj.net</a>	<a href="mailto:info@vtj.net">info@vtj.net</a>	
Meliads	<a href="http://www.Diamondengagmentring.Com">http://www.Diamondengagmentring.Com</a>	<a href="mailto:info@Diamondengagmentring.Com">info@Diamondengagmentring.Com</a>	
Lakthesis	<a href="http://www.Prosecutes.Com">http://www.Prosecutes.Com</a>	<a href="mailto:info@Prosecutes.Com">info@Prosecutes.Com</a>	
Oreades	<a href="http://www.shoppingjewelry.com">http://www.shoppingjewelry.com</a>	<a href="mailto:info@shoppingjewelry.com">info@shoppingjewelry.com</a>	

## Form row formatting shot

website:	<input type="text"/>
pays:	<input type="text" value="belgique"/>
email:	<input type="text"/>
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

file:	<input type="text"/>	<input type="button" value="Browse..."/>
-------	----------------------	--

## Dedomenon habla tu lengua

We use the Rails plugin Globalite:

<http://code.google.com/p/globalite/>

Translating the UI is done in one file.

To make your own, take the english to start from.

## Beyond customisation

- ▶ Rails engine go further than normal plugins.
- ▶ You can add completely new functionalities to Dedomenon
- ▶ We build our REST API as an engine!

## Get to your data

Work in progress nearing completion!

- ▶ Access to all functionalities
- ▶ We start with JSON but XML, yaml and others easy to add
- ▶ built as an engine

## Freedom to move

In validation phase

- ▶ Export your data to standard RDBMS
- ▶ Export all data types and data schema
- ▶ Initial support of Postgresql
- ▶ Other export added easily: postgres specific code 80 lines long!

## Freedom to code

- ▶ Project hosted at <http://www.dedomenon.org>
- ▶ Code managed with Git
- ▶ Mailing list at <http://lists.dedomenon.org>
- ▶ Platform with bug tracking, source code browsing, documentation, etc. . .

## Build on the foundation

- ▶ We have a solid foundation
- ▶ New data types
- ▶ Right-to-left interface
- ▶ Advanced reporting
- ▶ Import functionality

## Build a community

- ▶ We want to build a community
- ▶ We're listening to feedback
- ▶ Contributions are welcome

## Let's start the discussion

Raphaël Bauduin [rb@dedomenon.org](mailto:rb@dedomenon.org)  
<http://www.dedomenon.org>