

Encje w Drupalu

Tworzenie własnych encji i ich wpływ na poprawę wydajności

DrupalCamp Wrocław 2015

Grzegorz Bartman

<https://twitter.com/grzegorzbartman>



Droptica[™]
Experts in Enterprise **Drupal** Development

O mnie

Grzegorz Bartman

- <http://twitter.com/grzegorzbartman>
- <https://drupal.org/user/363120> ponad 6 lat na drupal.org,
- PHP od 2002 roku
- Od 2013 Droptica
 - <http://www.droptica.pl/oferty-pracy> zapraszamy do nas :)



Plan prezentacji

- Czym jest encja
- Jak łatwo zrobić własne encje w Drupalu
- Porównanie własnych encji z node'ami

Dla kogo jest ta prezentacja

- Znasz Drupala i PHP
 - Pisziesz własne proste moduły
- Nie tworzyłeś własnych encji w Drupalu



Czym jest Encja?

- Pojęcie często używane w projektowaniu baz danych
- Encja to reprezentacja wyobrażonego lub rzeczywistego obiektu (wikipedia)
- Encja posiada atrybuty/właściwości
- Przykłady encji:
 - Samochód (id_samochodu, marka, kolor, pojemność silnika)
 - Osoba (id_osoby, imię, nazwisko, rok urodzenia)
 - Właściciel samochodu (id, id_samochodu, id_osoby)



Standardowe podejście do tworzenia struktury danych w Drupalu

- Tworzymy nowy rodzaj zawartości
- Tworzymy pola do rodzaju zawartości

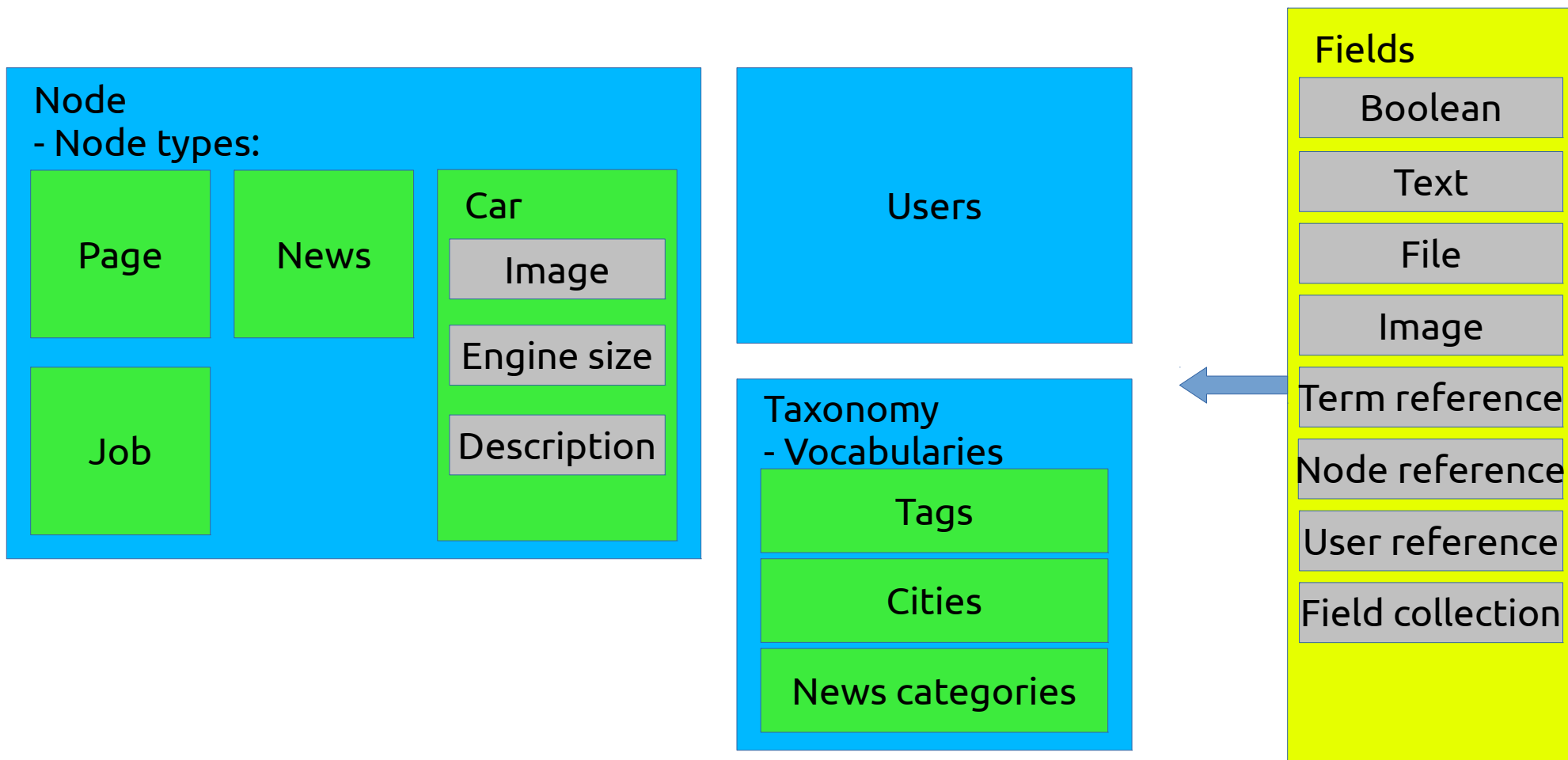


Standardowe podejście do tworzenia struktury danych w Drupalu

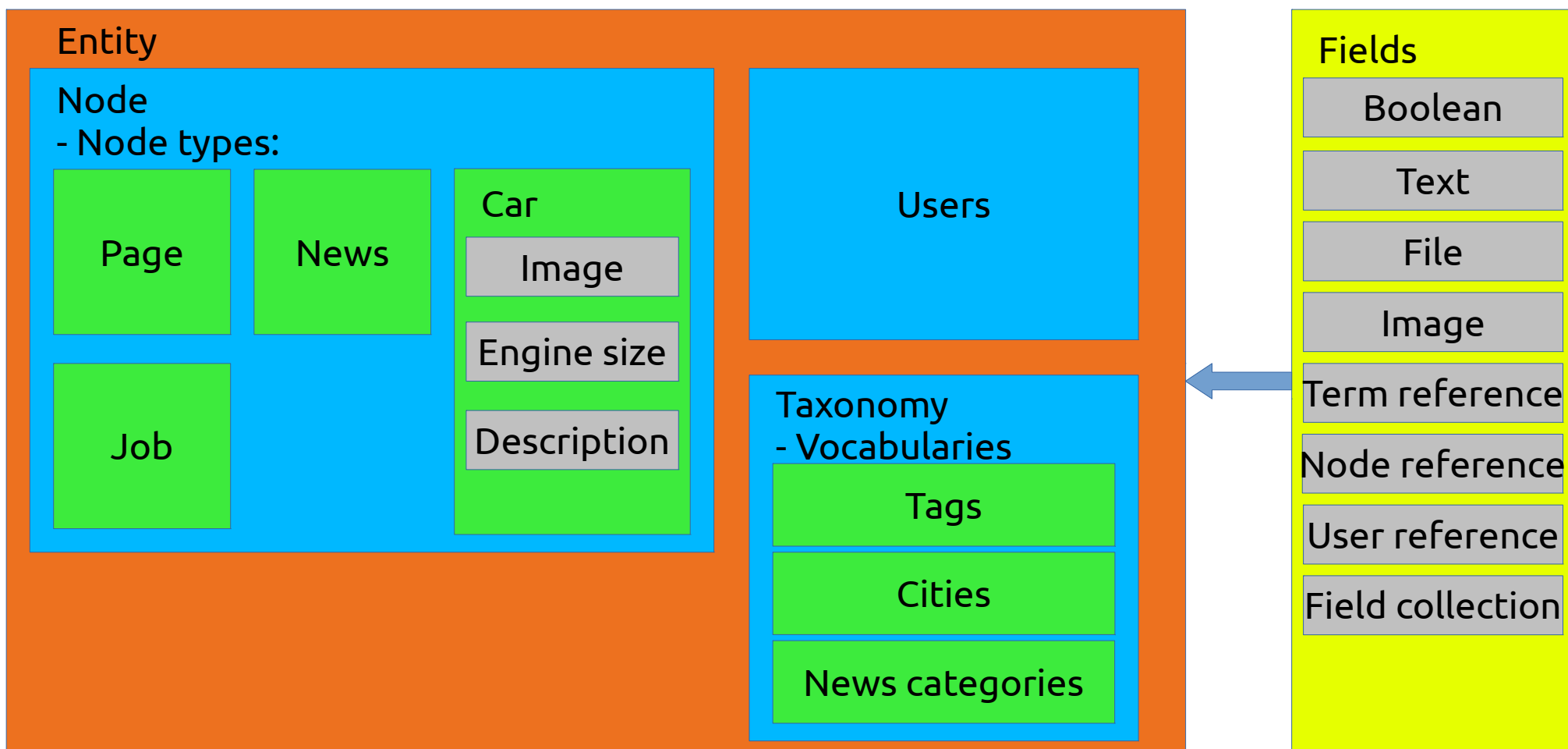
- Zalety:
 - Wygodne i szybkie tworzenie struktury danych
 - Łatwa edycja wyglądu i kolejności elementów na formularzu tworzenia treści

Z tego powodu Drupal jest chwalony przez początkujących użytkowników

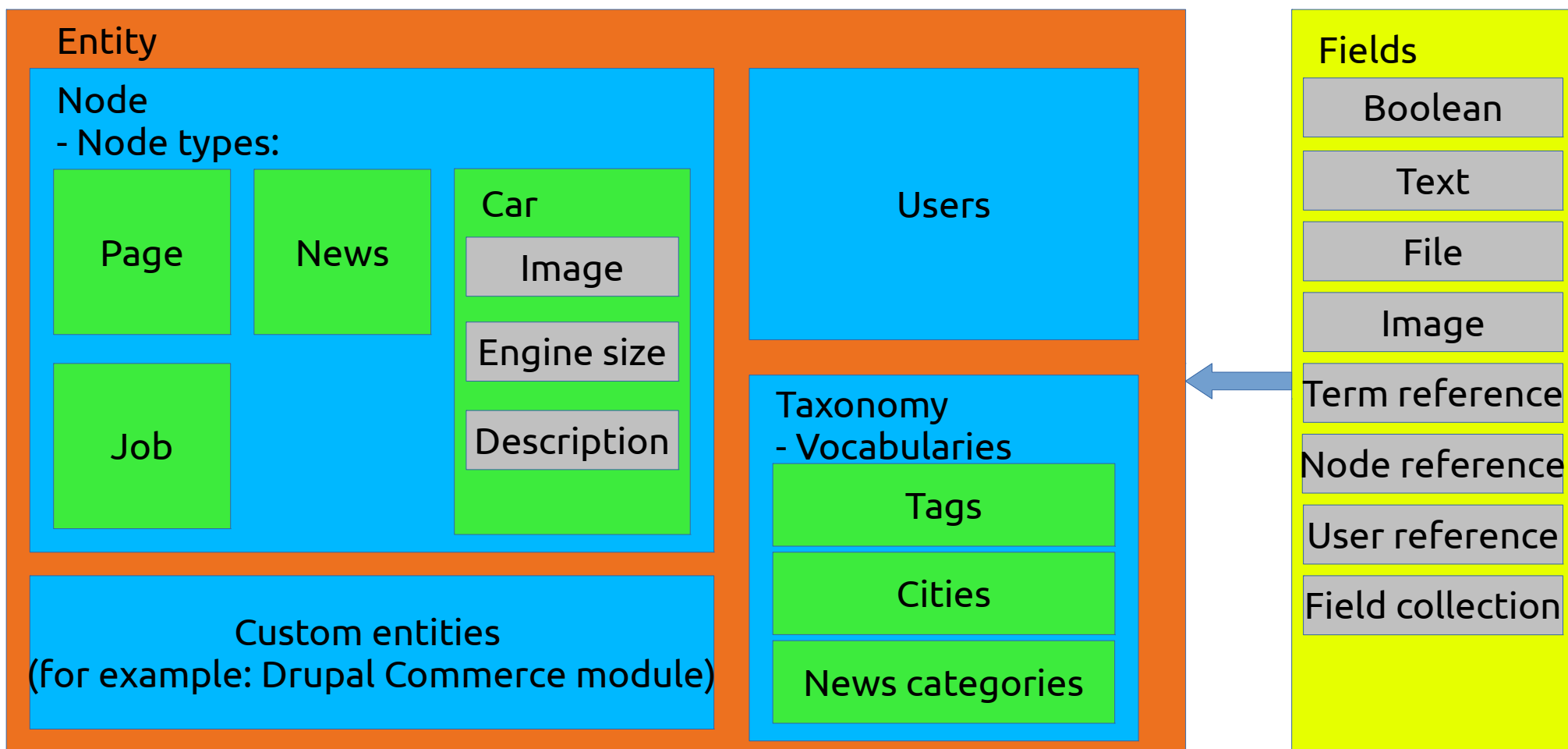
Struktura danych w Drupalu



Struktura danych w Drupalu



Struktura danych w Drupalu



Encje w Drupalu

- Rdzeń Drupala definiuje własne encje
- Encje w rdzeniu Drupala 7:
 - Node
 - Taxonomy
 - User



Po co tworzyć własne
encje?



Droptica[™]
Experts in Enterprise **Drupal** Development

Standardowe podejście do tworzenia struktury danych w Drupalu

- Wady:
 - Nie mamy kontroli nad tym co się dzieje w bazie danych
 - Każde pole to 2 dodatkowe tabele w bazie danych (field_data... i field_revision...)
 - Im więcej pól i powiązań tym dłuższe zapytania SQL (JOIN, JOIN, JOIN.....)

**Z tych powodów doświadczeni programiści nienawidzą
Drupala**

Dzięki encjom możemy dowolnie definiować strukturę tabel w bazie danych



Encje to nie tylko własna tabela w bazie danych

- **Views** może pobierać dane z encji
- Do encji można dodawać **pola**
- Moduł **Rules** może działać np. w momencie dodawania, edycji lub usuwania encji

Kiedy warto

Potrzebna duża szybkość ładowania się serwisu/aplikacji

Serwis/aplikacja będzie używana i rozwijana przez wiele lat, i chcemy wyeliminować późniejsze problemy z wydajnością oraz mieć możliwość elastycznej rozbudowy

Dużo pól w nodach, użytkownikach lub innych encjach

Chcemy mieć kontrole nad bazą danych



Kiedy nie warto

Serwisy z treściami dla niezalogowanych użytkowników bez konieczności szybkiego odświeżania wyświetlanych danych po ich zmianie w bazie

Lepiej i szybciej zrobić strukturę na node'ach oraz cachowanie serwisu (varnish, memcache, inne) niż tworzyć dla danych własne encje

Plusy

Dokładna kontrola nad strukturą danych oraz sposobem ich zarządzania przez Drupala (formularze dodawania i edycji, strony wyświetlania encji)

Poprawa wydajności w stosunku do node'ów z polami



Minusy

Bardziej czasochłonne w porównaniu do node'ów

Dodatkowe czynności do wykonania:

- zaprojektowanie i utworzenie tabel w bazie danych
- utworzenie formularzy do dodawania treści
- szablony do wyświetlania encji
- testowanie czy wszystko działa poprawnie (zapis, edycja, usuwanie, powiązania)



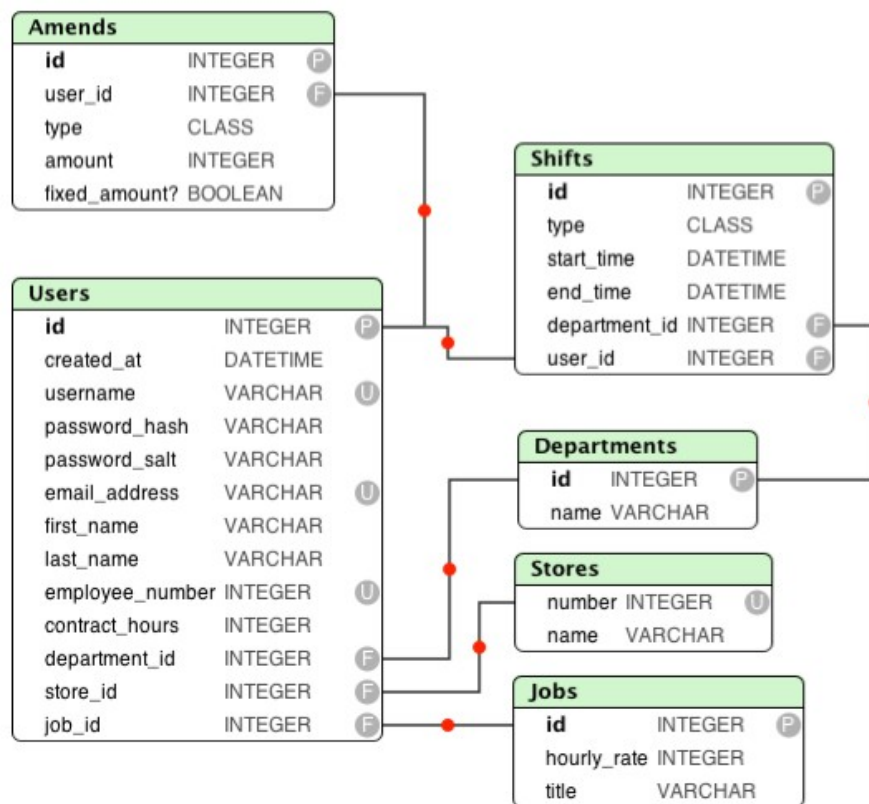
Od czego zacząć?



Droptica[™]
Experts in Enterprise **Drupal** Development

Diagram związków encji (ERD)

- Warto mieć wizualizację tabel w bazie danych w postaci diagramu



Moduł

- Encje tworzymy korzystając z odpowiednich hook'ów we własnym module*
- `hook_entity_info()` definiuje strukturę encji
- `hook_schema()` definiuje tabelę w bazie danych do przechowywania encji



Analiza przykładów – node.module

```
node.module x
164  /**
165   * Implements hook_entity_info().
166   */
167  function node_entity_info() {
168    $return = array(
169      'node' => array(
170        'label' => t('Node'),
171        'controller class' => 'NodeController',
172        'base table' => 'node',
173        'revision table' => 'node_revision',
174        'uri callback' => 'node_uri',
175        'fieldable' => TRUE,
176        'entity keys' => array(
177          'id' => 'nid',
178          'revision' => 'vid',
179          'bundle' => 'type',
180          'label' => 'title',
181          'language' => 'language',
182        ),
183        'bundle keys' => array(
184          'bundle' => 'type',
185        ),
186        'bundles' => array(),
187        'view modes' => array(
188          'full' => array(
189            'label' => t('Full content'),
190            'custom settings' => FALSE,
191          ),
192          'teaser' => array(
193            'label' => t('Teaser'),
194            'custom settings' => TRUE,
195          ),
196          'rss' => array(
197            'label' => t('RSS'),
198            'custom settings' => FALSE,
199          )
200        )
201      )
202    );
203  }
```



Inne przykłady modułów z encjami

- Profile2
- Drupal Commerce
- Message



Moduł ECK

<https://www.drupal.org/project/eck>

Pozwala ma „wyklikanie” własnych encji, ale posiada ograniczenia dotyczące typów danych w encji (properties)



Moduł Model

<https://www.drupal.org/project/model>

Przykład encji, na podstawie kodu można szybko tworzyć własne encje

Struktura encji + strony administracyjne

Moduł Model

Pobierz moduł z drupal.org

Zmień nazwę katalogu (np. na car)

Zmień nazwy plików (model zamień na car)

Zmień nazwy funkcji w plikach (model->car)



Moduł Model

The image shows a code editor interface with a file explorer on the left, PHP code in the center, and a 'Replace in Path' dialog box on the right.

File Explorer:

- modules
 - contrib
 - admin_menu
 - backup_migrate
 - context
 - ctools
 - features
 - module_filter
 - strongarm
 - views
 - custom
 - application
 - car
 - views
 - car.admin.inc
 - car.info
 - car.install
 - car.module
 - car.tpl.php
 - car-sample-data.tpl.php
 - car_type.admin.inc
 - LICENSE.txt
 - README.txt
 - dev

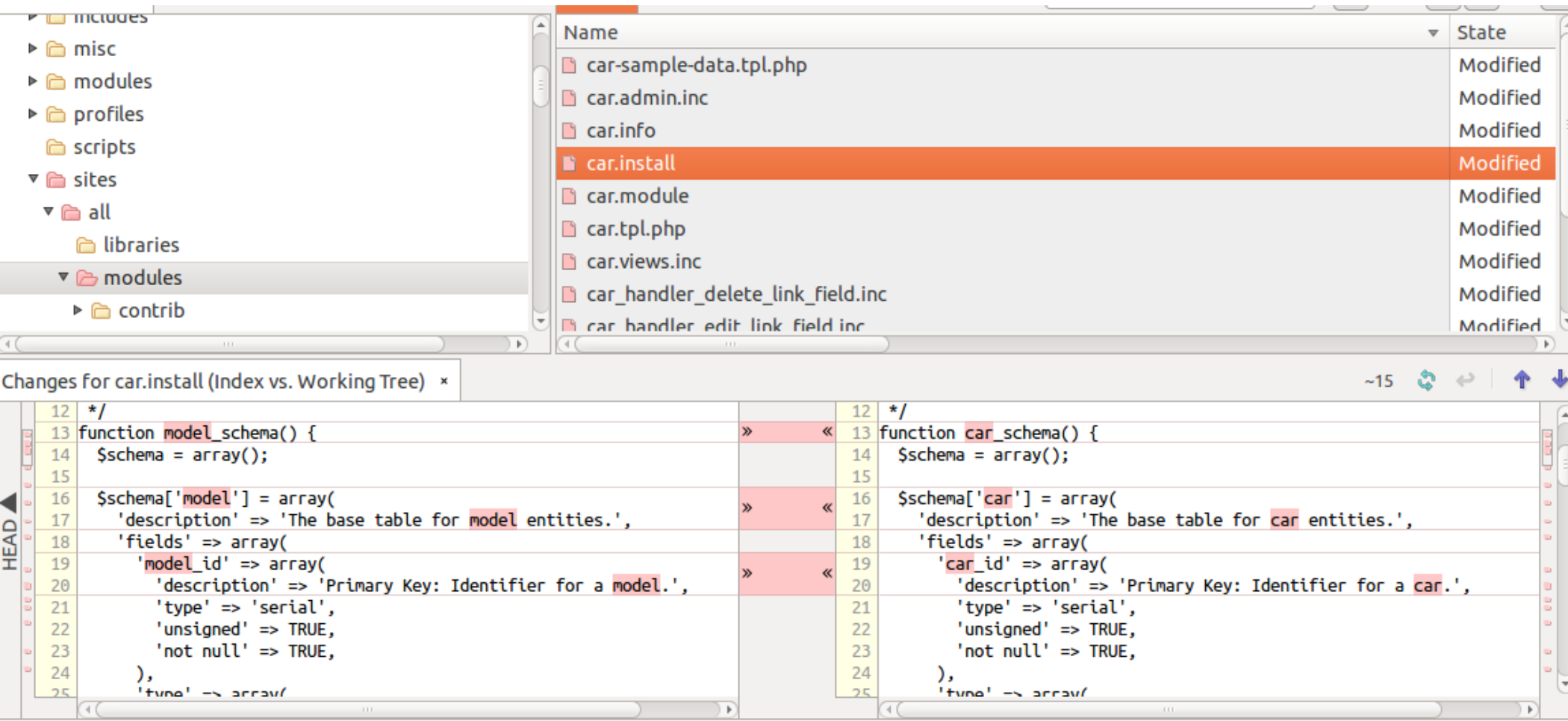
Code Editor:

```
9
10 /**
11  * Implements hook_schema().
12  */
13 function model_schema($schema = array()) {
14     $schema = array();
15
16     $schema['model'] = array(
17         'description' => 'Model',
18         'fields' => array(
19             'model_id' => array(
20                 'description' => 'Model ID',
21                 'type' => 'serial',
22                 'unsigned' => true,
23                 'not null' => true,
24             ),
25             'type' => array(
26                 'description' => 'Model type',
27                 'type' => 'varchar',
28                 'length' => 255,
29                 'not null' => true,
30                 'default' => 'other',
31             ),
32             'language' => array(
33                 'description' => 'Model language',
34                 'type' => 'varchar',
35                 'length' => 32,
36                 'not null' => true,
37                 'default' => 'other',
38             ),
39         ),
40     );
41 }
```

Replace in Path Dialog:

- Text to find: model
- Replace with: car
- Options:
 - Case sensitive
 - Preserve case
 - Whole words only (may be faster)
 - Regular expression [Help]
- Context: anywhere
- Scope:
 - Whole project
 - Directory: /opt/sites/all/modules/custom/car
 - Recursively
 - Custom: Project Files

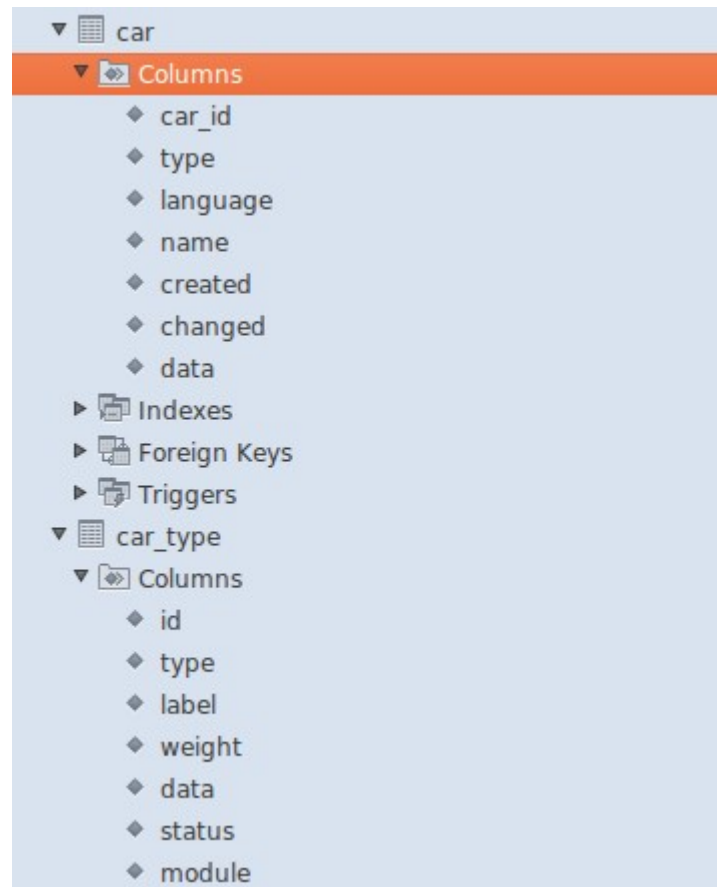
Moduł Model



The screenshot displays a code editor interface. On the left, a file explorer shows a directory structure with folders like 'includes', 'misc', 'modules', 'profiles', 'scripts', 'sites', and 'all'. The 'modules' folder is expanded, showing sub-folders 'libraries' and 'modules', and a 'contrib' folder. The main editor area shows a file list with 'car.install' selected. Below the file list, a diff view titled 'Changes for car.install (Index vs. Working Tree)' is shown, comparing the 'HEAD' version (left) with the 'Working Tree' version (right). The diff highlights changes in the 'model_schema()' function, which is replaced by a 'car_schema()' function. The 'description' and 'fields' arrays are updated to reflect the 'car' entity instead of the 'model' entity.

| Line | HEAD | Working Tree |
|------|--|--|
| 12 | */ | */ |
| 13 | function model_schema() { | function car_schema() { |
| 14 | \$schema = array(); | \$schema = array(); |
| 15 | | |
| 16 | \$schema['model'] = array(| \$schema['car'] = array(|
| 17 | 'description' => 'The base table for model entities.', | 'description' => 'The base table for car entities.', |
| 18 | 'fields' => array(| 'fields' => array(|
| 19 | 'model_id' => array(| 'car_id' => array(|
| 20 | 'description' => 'Primary Key: Identifier for a model.', | 'description' => 'Primary Key: Identifier for a car.', |
| 21 | 'type' => 'serial', | 'type' => 'serial', |
| 22 | 'unsigned' => TRUE, | 'unsigned' => TRUE, |
| 23 | 'not null' => TRUE, | 'not null' => TRUE, |
| 24 |), |), |
| 25 |), |), |
| | 'type' => array(| 'type' => array(|

Moduł Model



Co zmienić w plikach

Struktura bazy danych

- Plik: modulename.install
- Funkcja modulename_schema

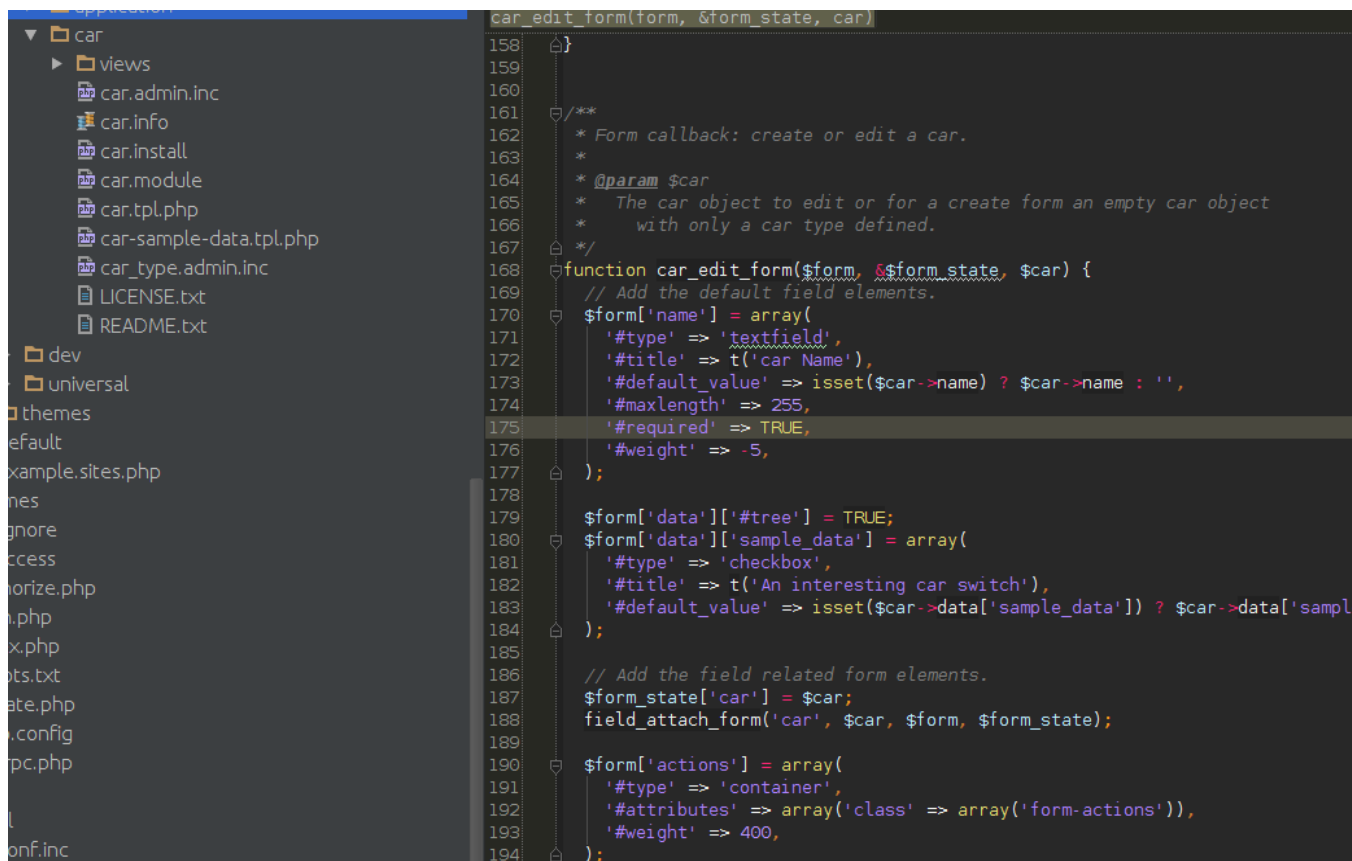
```
1 <?php
2
3 /* @file ... */
4
5
6
7
8
9
10 /**
11  * Implements hook_schema().
12  */
13 function car_schema() {
14     $schema = array();
15
16     $schema['car'] = array(
17         'description' => 'The base table for car entities.',
18         'fields' => array(
19             'car_id' => array(
20                 'description' => 'Primary Key: Identifier for a car.',
21                 'type' => 'serial',
22                 'unsigned' => TRUE,
23                 'not null' => TRUE,
24             ),
25             'type' => array(
26                 'description' => 'The {car_type}.type of this car.',
27                 'type' => 'varchar',
28                 'length' => 255,
29                 'not null' => TRUE,
30                 'default' => '',
31             ),
32             'language' => array(
33                 'description' => 'The language of the car.',
34                 'type' => 'varchar',
35                 'length' => 32,
36                 'not null' => TRUE,
37                 'default' => '',
38             ),
39         ),
40     );
41 }
```



Co zmienić w plikach

Formularz dodawania i edycji encji:

- Plik: modulename.admin.inc
- Funkcja modulename_edit_form



```
car_edit_form(form, &form_state, car)
158 }
159
160
161 /**
162  * Form callback: create or edit a car.
163  *
164  * @param $car
165  *   The car object to edit or for a create form an empty car object
166  *   with only a car type defined.
167  */
168 function car_edit_form($form, &$form_state, $car) {
169   // Add the default field elements.
170   $form['name'] = array(
171     '#type' => 'textfield',
172     '#title' => t('car Name'),
173     '#default_value' => isset($car->name) ? $car->name : '',
174     '#maxlength' => 255,
175     '#required' => TRUE,
176     '#weight' => -5,
177   );
178
179   $form['data']['#tree'] = TRUE;
180   $form['data']['sample_data'] = array(
181     '#type' => 'checkbox',
182     '#title' => t('An interesting car switch'),
183     '#default_value' => isset($car->data['sample_data']) ? $car->data['sample_data'] : FALSE,
184   );
185
186   // Add the field related form elements.
187   $form_state['car'] = $car;
188   field_attach_form('car', $car, $form, $form_state);
189
190   $form['actions'] = array(
191     '#type' => 'container',
192     '#attributes' => array('class' => array('form-actions')),
193     '#weight' => 400,
194   );
195 }
```

Co zmienić w plikach

Wyświetlanie encji

- Plik: modulename.module
- Metoda buildContent w klasie carController

```
/**
 * Overriding the buildContent function to add entity specific fields
 */
public function buildContent($entity, $view_mode = 'full', $langcode = NULL, $content = array()) {
    $content = parent::buildContent($entity, $view_mode, $langcode, $content);
    $content['car_sample_data'] = array(
        '#markup' => theme('car_sample_data', array('car_sample_data' => check_plain($entity->data['sample_data'])), 'car'
    );

    return $content;
}
```


Co zmienić w plikach

Adresy stron wyświetlania i edycji encji

- Plik: modulename.admin.inc
- Metoda hook_menu w klasie carUIController

```
*/
class carUIController extends EntityDefaultUIController {

  /**
   * Overrides hook_menu() defaults. Main reason for doing this is that
   * parent class hook_menu() is optimized for entity type administration.
   */
  public function hook_menu() {

    $items = array();
    $id_count = count(explode('/', $this->path));
    $wildcard = isset($this->entityInfo['admin ui']['menu wildcard']) ? $this

    $items[$this->path] = array(
      'title' => 'cars',
      'description' => 'Add edit and update cars.',
      'page callback' => 'system_admin_menu_block_page',
      'access arguments' => array('access administration pages'),
      'file path' => drupal_get_path('module', 'system'),
      'file' => 'system.admin.inc',
    );

    // Change the overview menu type for the list of cars.
    $items[$this->path]['type'] = MENU_LOCAL_TASK;
  }
}
```



SQL: Encje vs Node


Struktura danych do przechowywania informacji o samochodach i ich atrybutach:

- Nazwa
- Kolor
- Typ silnika
- Pojemność silnika
- Liczba drzwi

SQL: Encje vs Node

Node car i pola

[Home](#) » [Administration](#) » [Structure](#) » [Content types](#) » [Car](#)

Car 

EDIT








MANAGE FIELDS

MANAGE DISPLAY

COMMENT FIELDS

COMMENT

[Show](#)

| LABEL | MACHINE NAME | FIELD TYPE | WIDGET | OPER |
|---|-------------------|---------------------------------------|--|----------------------|
|  Title | title | Node module element | | |
|  Body | body | Long text and summary | Text area with a summary | edit |
|  color | field_color | Text | Text field | edit |
|  engine_size | field_engine_size | Text | Text field | edit |
|  doors | field_doors | Text | Text field | edit |
|  engine_type | field_engine_type | Text | Text field | edit |
|  Add new field | | | | |



Droptica[™]
Experts in Enterprise **Drupal** Development

SQL: Encje vs Node

Pobieranie danych z node'a i pól:

```
SELECT * FROM node n
LEFT JOIN field_data_field_color ON n.nid = field_data_field_color.entity_id
LEFT JOIN field_data_field_engine_size ON n.nid =
field_data_field_engine_size.entity_id
LEFT JOIN field_data_field_doors ON n.nid = field_data_field_doors.entity_id
LEFT JOIN field_data_field_engine_type ON n.nid =
field_data_field_engine_type.entity_id
```



SQL: Encje vs Node

Pobieranie danych z encji

```
SELECT * FROM car;
```



Podział pracy przy projekcie w Drupalu

Struktura danych w node'ach

- Site builder tworzy typy nodów i pola
- Site builder tworzy widoki, rulsy, dodaje pola
- Frontend developer dodaje css/html do widokow

Struktura danych we własnych encjach

- Developer tworzy encje (modeluje baze danych),
- Site builder tworzy widoki, rulsy, dodaje pola
- Frontend developer dodaje css/html do widokow

Pytania?



Droptica[™]
Experts in Enterprise **Drupal** Development

Dziękuję za uwagę

i zapraszam na stronę

<http://www.droptica.pl/oferty-pracy>

Oceń prezentację

<https://www.surveymonkey.com/s/DJFJ3QP>



Droptica[™]
Experts in Enterprise **Drupal** Development