Aplikacje WWW - laboratorium

(autor: Paweł Strzelczyk)

CakePHP

Celem ćwiczenia jest przygotowanie prostej aplikacji internetowej wykorzystującej architekturę szkieletową CakePHP. Ćwiczenie prezentuje podstawowe aspekty tworzenia aplikacji w oparciu o architekturę szkieletową (ang. *application framework*) bazującą na paradygmacie Model-View-Controller (MVC). W ramach tutorialu powstanie prosta aplikacja zawierająca przykładowy kontroler, obiekty modelu, obiekty widoku oraz obiekty pomocnicze. Zaprezentowane także zostaną przykłady wykorzystania funkcjonalności wbudowanej w architekturę szkieletową (walidacja danych, odwzorowanie obiektowo-relacyjne, itp.)

Ćwiczenie można wykonać na dowolnym komputerze, którym zainstalowano serwer HTTP (np. Apache) z obsługą PHP oraz bazę danych MySQL. Rozwiązania ćwiczeń omawianych w poniższym zestawie zostały przygotowane z wykorzystaniem pakietu XAMPP z PHP wersji 8.1.6.

UWAGA: CakePHP wykorzystuje bardzo wiele konwencji związanych z właściwym nazewnictwem elementów (kontrolerów, modeli, widoków). Wybór poszczególnych nazw dla klas i plików **jest nieprzypadkowy**, w celu płynnego wykonania tutorialu należy kierować się ściśle podanymi zaleceniami dotyczącymi nazw obiektów.

1. Pobierz ze strony https://github.com/cakephp/cakephp/tags wersję **4.3.9** architektury CakePHP i rozpakuj archiwum do katalogu `xampp/htdocs/cake` (należy utworzyć folder `cake` wewnątrz `htdocs`) .
2. Uruchom program XAMPP (serwer Apache i bazę danych MySQL), a następnie uruchom program phpMyAdmin (przycisk Admin na panelu kontrolnym programu XAMPP). Utwórz bazę danych o nazwie `cakephp`. Następnie, utwórz w bazie danych `cakephp` tabelę `books` o następującym schemacie (upewnij się, że łańcuchy znaków będą kodowane w utf8_polish_ci, pole Collation)

kolumna	typ i własności
id	INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
title	VARCHAR(50)
author	VARCHAR(20)
genre	VARCHAR(20)

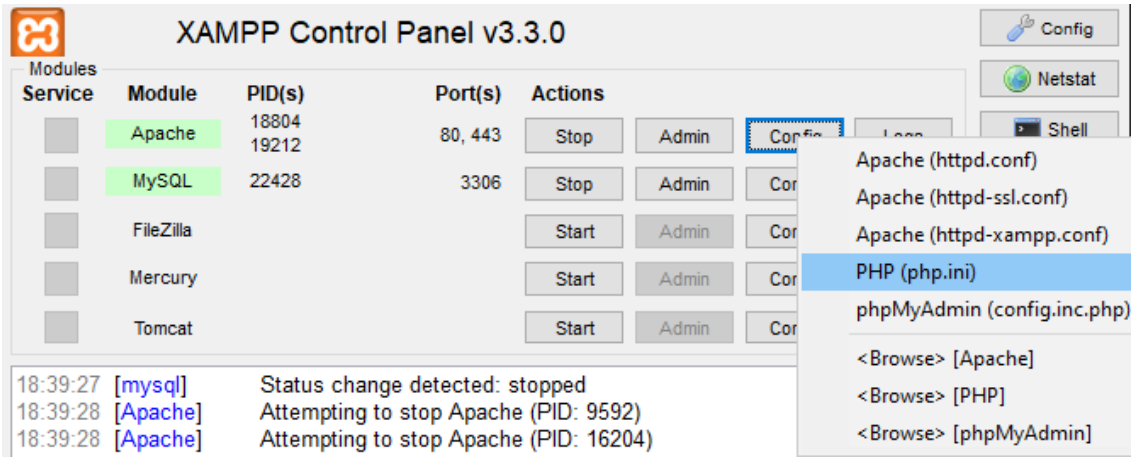
Wprowadź do nowoutworzonej tabeli kilka przykładowych rekordów.

3. Przejdź do katalogu `cake/config/` i otwórz plik `app_local.php`. Edytuj utworzony plik i ustaw w polu `Datasources` domyślne parametry logowania się do bazy danych podając nazwę utworzonej bazy danych, nazwę użytkownika ‘root’ oraz puste hasło.

```
'Datasources' => [
        'default' => [
            'host' => 'localhost',
            'username' => 'root',
            'password' => '',
            'database' => 'cakephp',
            'url' => env('DATABASE_URL', null),
        ],
...
```

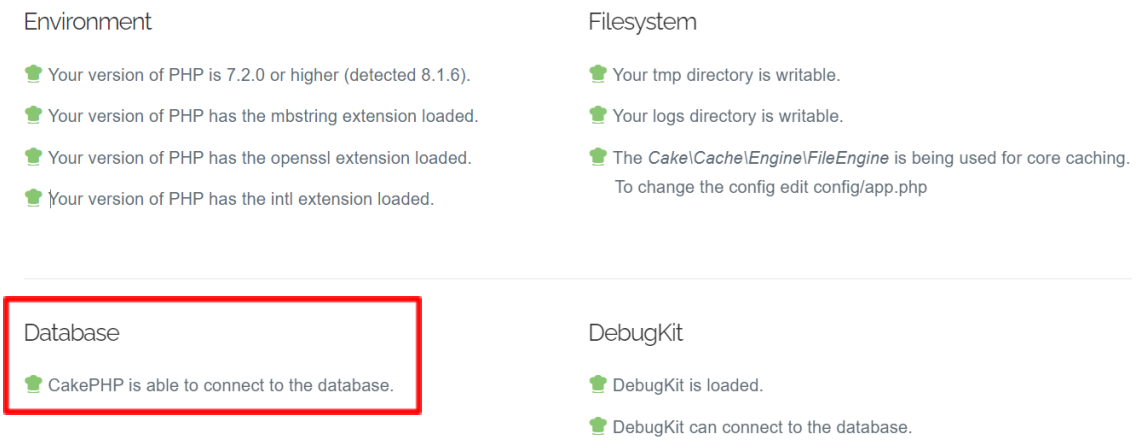
4. Przejdź do strony http://localhost/cake/ i upewnij się, że lokalna instalacja się powiodła.

   Jeśli pojawia się błąd *'You must enable the intl extension to use CakePHP.'*, przejdź do zakładki Config serwera Apache i otwórz plik php.ini.



   Odszukaj w pliku php.ini linijkę ‘;extension=intl’ i odkomentuj ją usuwając średnik. Zapisz zmiany i uruchom ponownie serwer.

   Sprawdź, czy połączenie z bazą jest możliwe.



5. Pierwszym krokiem ćwiczenia będzie utworzenie klasy modelu, która będzie reprezentowała tabelę w bazie danych. Przejdź do katalogu `cake/src/Model/Table/` i utwórz w nim plik `BooksTable.php`, a następnie umieść w pliku poniższy kod. Obiekt typu *Table* zapewnia dostęp do rekordów przechowywanych w konkretnej tabeli w bazie. CakePHP używa konwencji nazewnictwa do rozpoznania tabeli, którą ma reprezentować nasz model.

```
<?php
namespace App\Model\Table;
use Cake\ORM\Table;

class BooksTable extends Table{}
```

6. Kolejny krokiem jest utworzenie klasy reprezentującej wiersz tabeli i zdefiniowanie, które kolumny mogą być modyfikowane przez *mass assignment* (ze względów bezpieczeństwa nie powinno się ustawiać wartości '*' na *true*, ponieważ umożliwia to *mass assignment* wszystkich niewymienionych pól). Przejdź do katalogu `cake/src/Model/Entity/` i utwórz w nim plik Book.php, a następnie umieść w pliku poniższy kod.

```
<?php
namespace App\Model\Entity;
use Cake\ORM\Entity;

class Book extends Entity
{
    protected $_accessible = [
        '*' => false,
        'id' => false,
        'title' => true,
        'author' => true,
        'genre' => true
    ];}
```

7. W kolejnym kroku utworzysz najprostszy kontroler i wyposażysz go w metodę `index()`, która zostanie automatycznie wywołana w odpowiedzi na żądanie HTTP. Logika tej funkcji będzie dostępna zarówno pod adresem http://localhost/cake/books, jak i http://localhost/cake/books/index. Kontroler dla modelu `Book` powinien nosić nazwę `BooksController` i być umieszczony w pliku `BooksController.php`, w katalogu `cake/src/Controller/`. Przejdź do tego katalogu i utwórz wspomniany plik, wypełniając go poniższą treścią (akcja `set()` służy do przekazania danych z kontrolera do widoku *Template*).

```
<?php
namespace App\Controller;
use App\Controller\AppController;

class BooksController extends AppController
{
    public function index()
    {
        $this->loadComponent('Paginator');
        $books = $this->Paginator->paginate($this->Books->find());
        $this->set(compact('books'));
    }
}
```

8. Przejdź do strony http://localhost/cake/books/index. Dlaczego strona nie działa?
9. Przejdź do katalogu `cake/templates/` i utwórz w nim nowy katalog `Books`. Wejdź do nowego katalogu i utwórz w nim szablon widoku o nazwie odpowiadającej nazwie akcji, która aktywizuje dany widok (`index.php`). Umieść w pliku widoku poniższy kod:

```
<h1>Books</h1>
<table>
    <caption>Your books</caption>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Author</th>
        <th>Genre</th>
        <th>Action</th>
    </tr>
    <?php foreach ($books as $book) : ?>
        <tr>
            <td>
                <?= $book->id ?>
            </td>
            <td>
                <?= $this->Html->link($book->title, ['action' => 'view',
$book->id]) ?>
            </td>
            <td>
                <?= $book->author ?>
            </td>
            <td>
                <?= $book->genre ?>
            </td>
            <td>
                <?= $this->Html->link(
                    'Edit',
                    ['action' => 'edit', $book->id]
                ) ?>
                <?= $this->Form->postLink(
                    'Delete',
                    ['action' => 'delete', $book->id],
                    ['confirm' => 'Are you sure?']
                )
                ?>
            </td>
        </tr>
    <?php endforeach; ?>
</table>
```

Przeanalizuj starannie powyższy przykład. Zobacz, w jaki sposób następuje przejęcie danych wysłanych przez kontroler. Zauważ różne sposoby wywołania metody `Html→link()` (Html to komponent pomocniczy rejestrowany automatycznie w każdej aplikacji). Uruchom aplikację i przejdź do adresu http://localhost/cake/books/.

10. Akcje zdefiniowane w poprzednim widoku (`view, delete, edit`) nie zostały jeszcze zdefiniowane. Przejdź do kontrolera i dodaj do definicji klasy poniższą metodę.

```
public function view($id)
{
    $book = $this->Books->findById($id)->firstOrFail();
    $this->set(compact('book'));
}
```

11. Przejdź do katalogu `cake/templates/Books/` i utwórz nowy plik o nazwie `view.php`, a następnie umieść w nim poniższy kod, uruchom aplikację i przetestuj jego działanie poprzez kliknięcie w link umieszczony pod tytułem książki.

```
<h1><small>tytuł:</small> <?= h($book->title) ?></h1>
<h3><small>autor:</small> <?= h($book->author) ?></h3>
<h3><small>gatunek:</small> <?= h($book->genre) ?></h3>
<p><?= $this->Html->link('Edit', ['action' => 'edit', $book->id]) ?></p>
<a href="/cake/books">Return</a>
```

12. W następnym kroku obsłużysz procedurę dodawania nowych książek. Wróć do definicji kontrolera i umieść tam następującą funkcję:

```
public function add()
{
    $book = $this->Books->newEmptyEntity();
    if ($this->request->is('post')) {
        $book = $this->Books->patchEntity(
            $book,
            $this->request->getData()
        );
        if ($this->Books->save($book)) {
            $this->Flash->success(__('Your book has been added'));
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error(__('Unable to add book.'));
    }
    $this->set('book', $book);
}
```

W celu uporządkowania ładowania komponentów zmodyfikuj funkcję `index()` kontrolera i dodaj funkcję `initialize()`. Ich aktualny wygląd powinien prezentować się następująco:

```
public function initialize(): void
{
    parent::initialize();
    $this->loadComponent('Paginator');
    $this->loadComponent('Flash');
}

public function index()
{
    $books = $this->Paginator->paginate($this->Books->find());
    $this->set(compact('books'));
}
```

Komponent Flash umożliwia generowanie jednorazowych powiadomień wyświetlanych np. po przetworzeniu formularza.

Po zaimplementowaniu logiki dodawania nowych książek możesz przejść do widoku udostępniającego formatkę do dodawania książek. Utwórz w katalogu `cake/templates/Books/` plik `add.php` i umieść w nim:

```
<h1>Add book</h1>
<?php
$options = array('dramat' => 'dramat', 'komedia' => 'komedia', 'powieść'
=> 'powieść');
echo $this->Form->create($book);
echo $this->Form->control('author');
echo $this->Form->control('title');
echo $this->Form->control('genre', array('options' => $options, 'default'
=> 'dramat'));
echo $this->Form->button(__('Save book'));
echo $this->Form->end();
?>
```

13. Wróć do pliku `index.php` i przed znacznikiem `<table>` umieść link umożliwiający przejście do formularza dodawania nowej książki.

```
<p>
  <?= $this->Html->link('Add Book', ['action' => 'add']) ?>
</p>
```

14. CakePHP zawiera bardzo bogaty silnik walidacji danych. Przejdź do edycji klasy odpowiadającej tabeli *books* i utwórz funkcję, która zainicjuje i zwróci walidator zawierający pożądane reguły. Następnie sprawdź, czy możesz dodać nowy rekord nie podając tytułu wydarzenia. Pamiętaj o zaimportowaniu odpowiedniej klasy *Validator*.

```
function validationDefault(Validator $validator): Validator
{
    $validator
        ->notEmptyString('title')
        ->minLength('title', 4)
        ->maxLength('title', 255);

    return $validator;
}
```

15. Następnym krokiem jest oprogramowanie akcji usuwania książek. Przejdź do kontrolera i dodaj metodę, która będzie wywoływana w momencie kliknięcia na link Delete. Przetestuj działanie aplikacji.

```
public function delete($id)
{
    $book = $this->Books->findById($id)->firstOrFail();
    if ($this->Books->delete($book)) {
        $this->Flash->success(__(
            'The {0} book has been deleted.',
            $book->title
        ));
        return $this->redirect(['action' => 'index']);
    }
}
```

16. Ostatnią operacją CRUD, która nie została jeszcze zaimplementowana, jest operacja edycji. Przejdź do katalogu `cake/templates/Books/` i utwórz plik `edit.php` z następującą treścią:

```
<h1>Edit book</h1>
<?php
$options = array('dramat' => 'dramat', 'komedia' => 'komedia', 'powieść'
=> 'powieść');
echo $this->Form->create($book);
echo $this->Form->control('id', ['type' => 'hidden']);
echo $this->Form->control('author');
echo $this->Form->control('title');
echo $this->Form->control('genre', array('options' => $options, 'default'
=> 'dramat'));
echo $this->Form->button(__('Save book'));
echo $this->Form->end();
?>
```

17. Edytuj kontroler aplikacji i dodaj logikę biznesową odpowiedzialną za edycję książek.

```
public function edit($id)
{
    $book = $this->Books->findById($id)->firstOrFail();

    if ($this->request->is(['post', 'put'])) {
        $this->Books->patchEntity(
            $book,
            $this->request->getData()
        );
        if ($this->Books->save($book)) {
            $this->Flash->success(__('Your book has been updated.'));
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error(__('Unable to update your book.'));
    }

    $this->set('book', $book);
}
```

18. CakePHP dostarcza także automatycznego mechanizmu tworzenia całego szkieletu CRUD dla obiektów w bazie danych (kontroler, widoki, modele). Aby wykorzystać tę własność środowiska, uruchom aplikację PHPMyAdmin i utwórz w bazie danych następujące tabele:

```
CREATE TABLE users (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    password CHAR(40) NOT NULL,
    group_id INT(11) NOT NULL,
    created DATETIME,
    modified DATETIME
);


CREATE TABLE groups (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    created DATETIME,
    modified DATETIME
);
```

19. Upewnij się, że foldery `xampp/php` i `cake/bin` znajdują się na ścieżce. Jeśli nie, dodaj je do zmiennej `PATH` (upewnij się, że wpisujesz właściwe katalogi). Ustawienie zmiennej PATH w ten sposób ma wpływ jedynie na aktualną sesję konsoli (do zamknięcia okna).

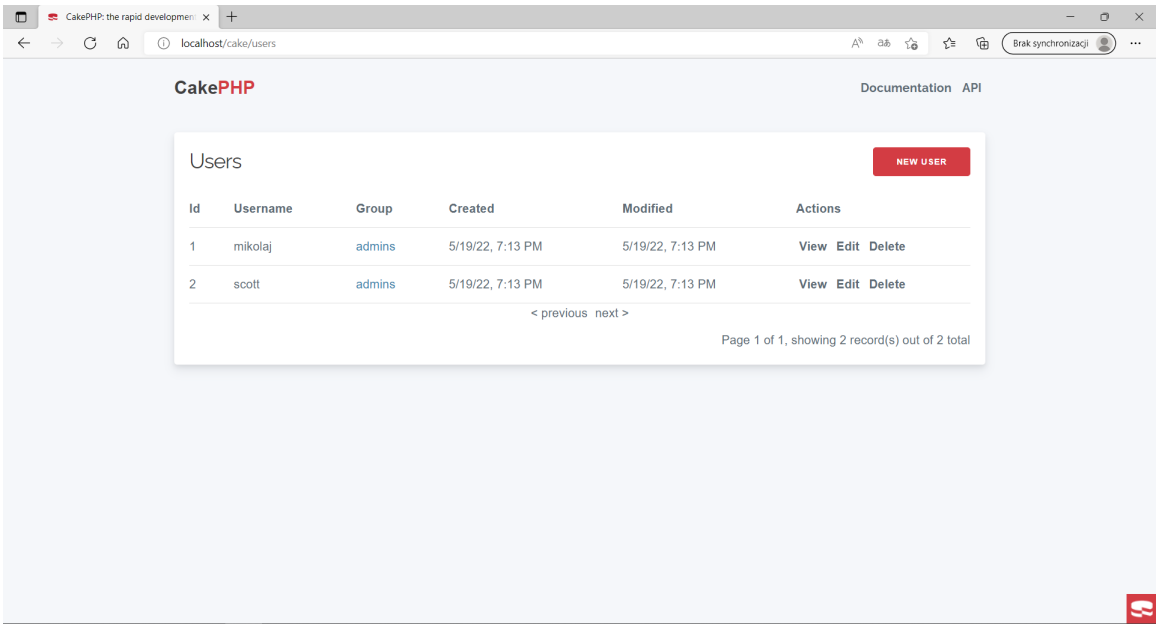
```
```

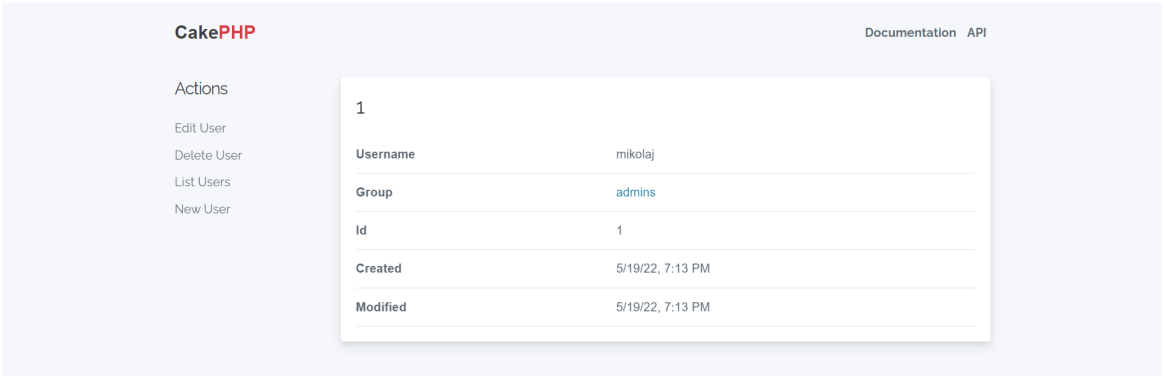
20. Przejdź do katalogu `cake/` i uruchom polecenie do automatycznej generacji szkieletu aplikacji. Wybierz najpierw tabelę `GROUPS` i wygeneruj właściwe pliki, następnie powtórz to samo dla tabeli `USERS`. Na czerwono zaznaczono polecenia, które należy wykonać.

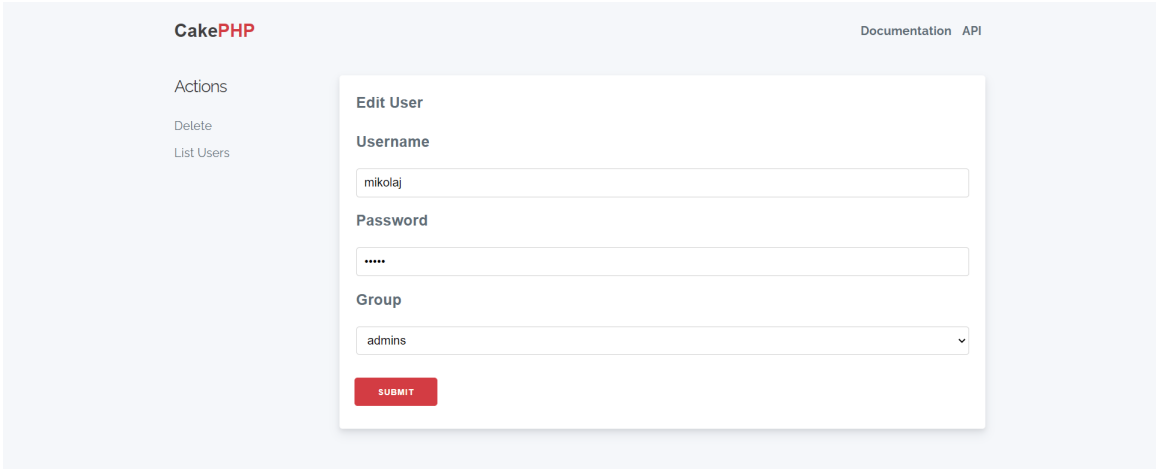
```
cake bake all

cake bake all Groups

cake bake all Users
```

Powyższe polecenia wygenerują dla każdej tabeli pliki z implementacją klas Table, Entity, Fixture (testy), AppController (kontroler) oraz szablony CRUD (add, edit, view, index).

21. Otwórz w przeglądarce adres http://localhost/cake/groups, dodaj nową grupę, a następnie pod adresem http://localhost/cake/users dodaj dwóch użytkowników. Zapoznaj się z wygenerowanymi plikami kontrolerów, modeli i widoków.







22. Wykorzystaj poniższy kod do utworzenia tabeli `PRACOWNICY` i wypełnienia jej danymi. Utwórz szkielet aplikacji CRUD do obsługi tabeli `PRACOWNICY`. Popraw wygenerowane automatycznie pliki w następujący sposób:
    - dodaj do modelu informację o tym, że nazwiska i etaty pracowników są wymagane,
    - dodaj walidację płacy podstawowej, musi się zawierać w przedziale <0-2000>,
    - pole do wprowadzania etatu powinno być statyczną listą rozwijaną.

```
DROP TABLE IF EXISTS `employees`;
CREATE TABLE `employees` (
  `id` int AUTO_INCREMENT NOT NULL,
  `nazwisko` varchar(15) default NULL,
  `imie` varchar(15) default NULL,
  `etat` varchar(10) default NULL,
  `id_szefa` decimal(4,0) default NULL,
  `zatrudniony` date default NULL,
  `placa_pod` decimal(6,2) default NULL,
  `placa_dod` decimal(6,2) default NULL,
  `id_zesp` decimal(2,0) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM;

INSERT INTO `employees` (`id`, `nazwisko`, `imie`, `etat`, `id_szefa`,
`zatrudniony`, `placa_pod`, `placa_dod`, `id_zesp`)
VALUES
(100, 'Marecki', 'Jan', 'DYREKTOR', NULL, '1968-01-01', 4730.00, 980.50, 10),
(110, 'Janicki', 'Karol', 'PROFESOR', 100, '1973-05-01', 3350.00, 610.00, 40),
(120, 'Nowicki', 'Pawel', 'PROFESOR', 100, '1977-09-01', 3070.00, NULL, 30),
(130, 'Nowak', 'Piotr', 'PROFESOR', 100, '1968-07-01', 3960.00, NULL, 20),
(140, 'Kowalski', 'Krzysztof', 'PROFESOR', 130, '1975-09-15', 3230.00, 805.00, 20),
(150, 'Grzybowska', 'Maria', 'ADIUNKT', 130, '1977-09-01', 2845.50, NULL, 20),
(160, 'Krakowska', 'Joanna', 'SEKRETARKA', 130, '1985-03-01', 1590.00, NULL, 20),
(170, 'Opolski', 'Roman', 'ASYSTENT', 130, '1992-10-01', 1839.70, 480.50, 20),
(190, 'Kotarski', 'Konrad', 'ASYSTENT', 140, '1993-09-01', 1971.00, NULL, 20),
(180, 'Makowski', 'Marek', 'ADIUNKT', 100, '1985-02-20', 2610.20, NULL, 10),
(200, 'Przywarek', 'Leon', 'DOKTORANT', 140, '1994-07-15', 900.00, NULL, 30),
(210, 'Kotlarczyk', 'Stefan', 'DOKTORANT', 130, '1993-10-15', 900.00, 570.60, 30),
(220, 'Siekierski', 'Mateusz', 'ASYSTENT', 110, '1993-10-01', 1889.00, NULL, 20),
(230, 'Dolny', 'Tomasz', 'ASYSTENT', 120, '1992-09-01', 1850.00, 390.00, NULL);
```