**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 4

по дисциплине: Технологии Web-программирования

тема: «**Разработка и проектирование базы данных**»

Выполнил: ст. группы ПВ-41

Таран Алексей Николаевич

Проверил: Картамышев С.В.

Белгород 2020

**Задание к лабораторной работе:**

1. Выбрать подходящую СУБД.
2. Изучить методы взаимодействия web-приложения с базой данных (ORM, Active Record).
3. Разработать структуру базы данных.
4. Разработать соответствующие модели в приложении.
5. В отчёт приложить схему базы данных, а также код одной из моделей (на своё усмотрение).

Ход работы

В качестве базы данных выберем MySQL, которая была выбрана в предыдущей лабораторной работе для создания docker-контейнера:

mysql:

image: mysql

command: --default-authentication-plugin=mysql\_native\_password --character-set-server=utf8 --collation-server=utf8\_general\_ci

environment:

- MYSQL\_ROOT\_PASSWORD=verysecret

- MYSQL\_DATABASE=iceland

- MYSQL\_USER=alexey

- MYSQL\_PASSWORD=Alexey

Тогда, в конфигурации развернутого php yii2 проекта пропишем в файле config/db.php настройки конфигурации базы данных.

<?php

return [

'class' => 'yii\db\Connection',

'dsn' => 'mysql:host=mysql;dbname=iceland',

'username' => 'alexey',

'password' => 'alexey',

'charset' => 'utf8',

];

Создадим базу данных.

Выполним команду docker-compose run php yii migrate/create create\_city\_table, тем самым создав миграцию для таблицы create\_city\_table, чтобы впоследствии была возможность использовать данную структуру в mysql базе. Приведем пример создания миграции для одной из таблиц:

<?php

use yii\db\Migration;

/\*\*

\* Handles the creation of table `{{%chapter}}`.

\*/

class m201117\_113315\_create\_chapter\_table extends Migration

{

/\*\*

\* {@inheritdoc}

\*/

public function safeUp()

{

$this->createTable('{{%chapter}}', [

'id' => $this->primaryKey()->notNull(),

'idCity' => $this->integer()->comment('Название города'),

'text' => $this->string(100)->notNull()->comment('Текст'),

'photoPath' => $this->string()->notNull()->comment('Путь к изображению'),

'link' => $this->string()->notNull()->comment('Ссылка в приложении'),

]);

$this->addForeignKey(

'fk-chapter-idCity',

'chapter',

'idCity',

'city',

'id',

'NO ACTION',

'NO ACTION'

);

}

/\*\*

\* {@inheritdoc}

\*/

public function safeDown()

{

$this->dropForeignKey(

'fk-chapter-idCity',

'chapter'

);

$this->dropTable('{{%chapter}}');

}

}

Накатить миграцию можно при помощи команды: docker-compose run php yii migrate/up

После того, как в удаленной базе данных появилась таблица, есть возможность получить класс-модель соответствующей таблицы, благодаря yii code generator <http://localhost:1199/gii>. В качестве метода взаимодействия с базой данных используется Active Record. Покажем, как осуществляется получение класса:

Graphical user interface, text, application

Description automatically generated

Заполняем необходимые поля

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Копируем полученную информацию в app\modules\v1\models\Chapter.php;

Меняем строку namespace с указаением нового пути: namespace app\modules\v1\models;

Graphical user interface, text, application

Description automatically generated

Так выглядит код сгенерированной модели таблицы разделов:

<?php

namespace app\modules\v1\models;

use Yii;

/\*\*

\* This is the model class for table "chapter".

\*

\* @property int $id

\* @property int|null $idCity Название города

\* @property string $text Текст

\* @property string $photoPath Путь к изображению

\* @property string $link Ссылка в приложении

\*

\* @property City $idCity0

\*/

class Chapter extends \yii\db\ActiveRecord

{

/\*\*

\* {@inheritdoc}

\*/

public static function tableName()

{

return 'chapter';

}

/\*\*

\* {@inheritdoc}

\*/

public function rules()

{

return [

[['idCity'], 'integer'],

[['text', 'photoPath', 'link'], 'required'],

[['text'], 'string'],

[['photoPath', 'link'], 'string', 'max' => 255],

[['idCity'], 'exist', 'skipOnError' => true, 'targetClass' => City::className(), 'targetAttribute' => ['idCity' => 'id']],

];

}

/\*\*

\* {@inheritdoc}

\*/

public function attributeLabels()

{

return [

'id' => 'ID',

'idCity' => 'Название города',

'text' => 'Текст',

'photoPath' => 'Путь к изображению',

'link' => 'Ссылка в приложении',

];

}

/\*\*

\* Gets query for [[IdCity0]].

\*

\* @return \yii\db\ActiveQuery

\*/

public function getIdCity0()

{

return $this->hasOne(City::className(), ['id' => 'idCity']);

}

}

Повторим действия с написанием и применением миграций для всех таблиц базы данных. Получим классы при помощи генератора моделей.

Text

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

Тогда, имеем для базы данных следующую схему:

Diagram

Description automatically generated

Для заполнения Базы данных используем простую функцию написанного REST-API, подобно следующей:

class AttractionController extends ApiController{

/\*\*

\* Пример запроса:

\* http://127.0.0.1:1199/api/v1/attraction/create

\*/

public function actionCreate(){

$model = new Attraction();

$model->load(Yii::$app->request->getBodyParams(), '');

$model->save();

return $model;

}

}

Тогда заполнить БД можно при помощи GET-запроса из POSTMAN, передав в тело нужные параметры. Приведем пример:

Graphical user interface, text, application, email

Description automatically generated

Проверим изменения в БД из adminer:

A picture containing table

Description automatically generated

Таким образом, заполним всю базу данных и проверим наличие данных в adminer, например, для таблицы заметок:

Graphical user interface, text, application, email

Description automatically generated