**Київський національний університет імені Тараса Шевченка**

Звіт до лабораторного практикуму з курсу

«Інформаційні технології»

Виконали

студенти 1 курсу магістратури

спеціальності «інформатика»

***Єрмілов Ярослав***

***Трубецкая Олександра***

***Київ***

***2013***

# Постановка задачі

Фрагментарна реалізація система управління табличними базами даних.

Вимоги:

* кількість таблиці принципово необмежена
* кількість полів та кількість записів у кожній таблиці також принципово не обмежені
* забезпечити підтримку в таблицях полів типів – integer, real, char.

Реалізувати функціональну підтримку для:

* створення бази
* створення та знищення таблиці з бази
* перегляду та редагування рядків таблиці
* збереження та зчитування табличної бази
* додаткові типи – IntInterval, TextFile, Longint, HtmlFile
* додаткові операції – вилучення дублікатів, різниця таблиць

# Виконання етапів

## Етап 1

Функціональна специфікація системи у вигляді діаграм варіантів використання UML.

*Виконання*: для створення діаграм використовувалася мова PlantUml.

Вихідний код:

client -> (show tables list)

client -> (show table signature)

client -> (create table)

client -> (remove table)

client -> (drop database)

client -> (select data from table)

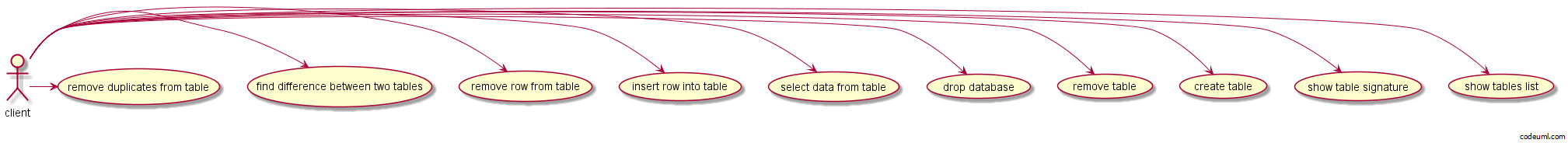
client -> (insert row into table)

client -> (remove row from table)

client -> (find difference between two tables)

client -> (remove duplicates from table)

Результат:



## Етап 2

Пропущено.

## Етап 3

Реалізувати класи управління базою даних. Надати 3 модульні тести.

*Виконання*: для створення діаграм класів використовувалася мова PlantUml.

Вихідний код:

class Table {

String name

}

class Row {

}

class Type {

String name

Value value(String stringValue)

}

class Value {

Object object

String valueAsString

String valueAsInnerString

}

Table "1" \*-- "\*" Type : specified by

Row "1" o-- "\*" Value : contains

Value "\*" \*-- "1" Type : has

Type <|-- Int

Type <|-- Char

Type <|-- Real

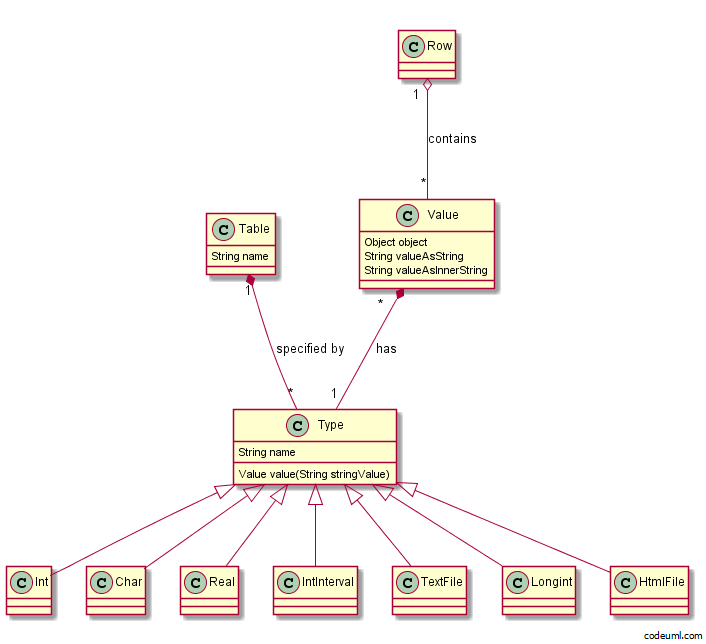
Type <|-- IntInterval

Type <|-- TextFile

Type <|-- Longint

Type <|-- HtmlFile

Результат:



Інтерфейс DatabaseManager містить в собі всі операції, що можна виконувати над базою даних.

package cyberwaste.kuzoff.core;

import java.util.Collection;

import java.util.List;

import cyberwaste.kuzoff.core.domain.Row;

import cyberwaste.kuzoff.core.domain.Table;

public interface DatabaseManager {

Collection<Table> getAllTables() throws Exception;

Table createTable(String name, String[] columnTypes) throws Exception;

Table removeTable(String tableName) throws Exception;

void removeDatabase() throws Exception;

Table getTable(String tableName) throws Exception;

Row insertRow(String tableName, String[] stringValues) throws Exception;

List<Row> getRows(String tableName) throws Exception;

List<Row> removeRow(String tableName, String[] values) throws Exception;

List<Row> removeDuplicates(String tableName) throws Exception;

List<Row> difference(String tableName1, String tableName2) throws Exception;

}

При розробці модульних тестів був використаний промисловий стандарт для Java – JUnit та Mockito для створення моків об’єктів для дійсно ізольованого тестування.

**package** cyberwaste.kuzoff.core.impl;

**import** **static** org.mockito.Mockito.*any*;

**import** **static** org.mockito.Mockito.*eq*;

**import** **static** org.junit.Assert.*assertEquals*;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.List;

**import** org.junit.Test;

**import** org.mockito.Mockito;

**import** cyberwaste.kuzoff.core.FileSystemManager;

**import** cyberwaste.kuzoff.core.domain.Row;

**import** cyberwaste.kuzoff.core.domain.Table;

**import** cyberwaste.kuzoff.core.domain.Type;

**import** cyberwaste.kuzoff.core.domain.Value;

**public** **class** DatabaseManagerImplTest {

@Test

**public** **void** test\_getAllTables() **throws** IOException {

// setup

File table1 = **new** File("table1");

File table2 = **new** File("table2");

List<File> filesList = Arrays.*asList*(table1, table2);

FileSystemManager fileSystemManager = Mockito.*mock*(FileSystemManager.**class**);

Mockito.*when*(fileSystemManager.getSubdirectories((File) *any*())).thenReturn(filesList);

Mockito.*when*(fileSystemManager.readFromFile(*eq*(table1), *eq*(DatabaseManagerImpl.*METADATA\_FILE*))).thenReturn("");

Mockito.*when*(fileSystemManager.readFromFile(*eq*(table2), *eq*(DatabaseManagerImpl.*METADATA\_FILE*))).thenReturn("");

DatabaseManagerImpl databaseManagerImpl = **new** DatabaseManagerImpl();

databaseManagerImpl.setFileSystemManager(fileSystemManager);

// run

Collection<Table> actual = databaseManagerImpl.getAllTables();

// verify

Collection<Table> expected = Arrays.*asList*(**new** Table("table1"), **new** Table("table2"));

*assertEquals*(expected, actual);

}

@Test

**public** **void** test\_getAllTables\_emptyResult() **throws** IOException {

// setup

List<File> filesList = Arrays.*asList*();

FileSystemManager fileSystemManager = Mockito.*mock*(FileSystemManager.**class**);

Mockito.*when*(fileSystemManager.getSubdirectories((File) *any*())).thenReturn(filesList);

DatabaseManagerImpl databaseManagerImpl = **new** DatabaseManagerImpl();

databaseManagerImpl.setFileSystemManager(fileSystemManager);

// run

Collection<Table> actual = databaseManagerImpl.getAllTables();

// verify

Collection<Table> expected = Arrays.*asList*();

*assertEquals*(expected, actual);

}

@Test

**public** **void** test\_removeDuplicates() **throws** IOException {

// setup

File table1 = **new** File("table1");

FileSystemManager fileSystemManager = Mockito.*mock*(FileSystemManager.**class**);

Mockito.*when*(fileSystemManager.readFromFile(*eq*(table1), *eq*(DatabaseManagerImpl.*METADATA\_FILE*))).thenReturn("Int|Char");

Mockito.*when*(fileSystemManager.readFromFile(*eq*(table1), *eq*(DatabaseManagerImpl.*DATA\_FILE*))).thenReturn("1|A\n1|B\n1|A");

DatabaseManagerImpl databaseManagerImpl = **new** DatabaseManagerImpl();

databaseManagerImpl.setFileSystemManager(fileSystemManager);

// run

List<Row> actual = databaseManagerImpl.removeDuplicates("table1");

// verify

List<Row> expected = Arrays.*asList*(

**new** Row(**new** Value[] { Type.*getType*("Int").value("1"), Type.*getType*("Char").value("A") }),

**new** Row(**new** Value[] { Type.*getType*("Int").value("1"), Type.*getType*("Char").value("B") })

);

*assertEquals*(expected, actual);

}

}

## Етап 4

Розробка локальної (нерозподіленої) версії СУТБД із власною реалізацією.

*Виконання:* реалізована на Java, доступ до бази даних відбувається через консольний інтерфейс.

Консольні команди мають наступний вигляд:

***{дієслово} {іменник} {параметри}***

Наприклад: show database; show table <table\_name>; select table <table\_name>; add row <table\_name> <row\_values>, і т.п..

За обробку команд з консолі відповідає наступний код:

package cyberwaste.kuzoff.shell;

import java.io.BufferedReader;

import java.io.IOException;

import javax.annotation.PostConstruct;

import org.apache.commons.lang.StringUtils;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import cyberwaste.kuzoff.core.IOManager;

import cyberwaste.kuzoff.core.DatabaseManager;

import cyberwaste.kuzoff.core.command.Command;

import cyberwaste.kuzoff.core.command.CommandBuilder;

public class ShellManager extends IOManager {

@SuppressWarnings("resource")

public static void main(String[] args) {

new AnnotationConfigApplicationContext(Config.class);

}

private BufferedReader reader;

private DatabaseManager databaseManager;

@PostConstruct

public void start() {

super.start();

}

public void setDatabaseManager(DatabaseManager databaseManager) {

this.databaseManager = databaseManager;

}

public void setReader(BufferedReader reader) {

this.reader = reader;

}

@Override

protected Command getNextCommand() throws IOException {

String commandAsString = StringUtils.stripToEmpty(reader.readLine());

if ("exit".equals(commandAsString)) {

return null;

}

String[] commandTokens = commandAsString.split(" ");

if (commandTokens.length < 2) {

throw new IOException("Wrong command syntax");

}

String commandVerb = commandTokens[0];

String commandNoun = commandTokens[1];

String[] commandArguments = new String[commandTokens.length - 2];

System.arraycopy(commandTokens, 2, commandArguments, 0, commandArguments.length);

Command command = CommandBuilder

.command(commandVerb, commandNoun)

.usingDatabaseManager(databaseManager)

.withArguments(commandArguments)

.build();

if (command == null) {

throw new IOException("Wrong command syntax");

}

return command;

}

@Override

protected void outputResult(String message) {

System.out.println(message);

}

@Override

protected void outputMessage(String message) {

System.err.println(message);

}

}

## Етап 5-6

Два варіанти розподілених версій систем.

*Виконання:* були обрані та реалізовані варіанти RMI та RemoteHttp.

Обидві вони використовують фреймворк Spring Remoting.

Приклад налаштування для клієнтської сторони:

package cyberwaste.kuzoff.rmi.client;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import org.springframework.beans.factory.config.BeanDefinition;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Scope;

import org.springframework.remoting.rmi.RmiProxyFactoryBean;

import cyberwaste.kuzoff.core.DatabaseManager;

@Configuration

@ComponentScan("cyberwaste.kuzoff")

class Config {

@Bean

public RmiClientManager rmiClientManager() {

RmiClientManager rmiClientManager = new RmiClientManager();

rmiClientManager.setDatabaseManager(remoteDatabaseManager());

rmiClientManager.setReader(new BufferedReader(new InputStreamReader(System.in)));

return rmiClientManager;

}

@Bean

@Scope(BeanDefinition.SCOPE\_PROTOTYPE)

public DatabaseManager remoteDatabaseManager() {

RmiProxyFactoryBean rmiProxyFactoryBean = new RmiProxyFactoryBean();

rmiProxyFactoryBean.setServiceUrl("rmi://localhost/DatabaseManager");

rmiProxyFactoryBean.setServiceInterface(DatabaseManager.class);

rmiProxyFactoryBean.afterPropertiesSet();

return (DatabaseManager) rmiProxyFactoryBean.getObject();

}

}

Приклад налаштування для серверної сторони:

package cyberwaste.kuzoff.rmi.server;

import org.springframework.beans.factory.config.BeanDefinition;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Scope;

import org.springframework.remoting.rmi.RmiServiceExporter;

import cyberwaste.kuzoff.core.DatabaseManager;

import cyberwaste.kuzoff.core.FileSystemManager;

import cyberwaste.kuzoff.core.impl.DatabaseManagerImpl;

import cyberwaste.kuzoff.core.impl.FileSystemManagerImpl;

@Configuration

@ComponentScan("cyberwaste.kuzoff")

class Config {

@Bean

public RmiServiceExporter serverDatabaseManager() {

RmiServiceExporter rmiServiceExporter = new RmiServiceExporter();

rmiServiceExporter.setService(databaseManager());

rmiServiceExporter.setServiceName("DatabaseManager");

rmiServiceExporter.setServiceInterface(DatabaseManager.class);

return rmiServiceExporter;

}

@Bean

@Scope(BeanDefinition.SCOPE\_PROTOTYPE)

public DatabaseManager databaseManager() {

DatabaseManagerImpl databaseManager = new DatabaseManagerImpl();

databaseManager.setFileSystemManager(fileSystemManager());

return databaseManager;

}

@Bean

public FileSystemManager fileSystemManager() {

return new FileSystemManagerImpl();

}

}

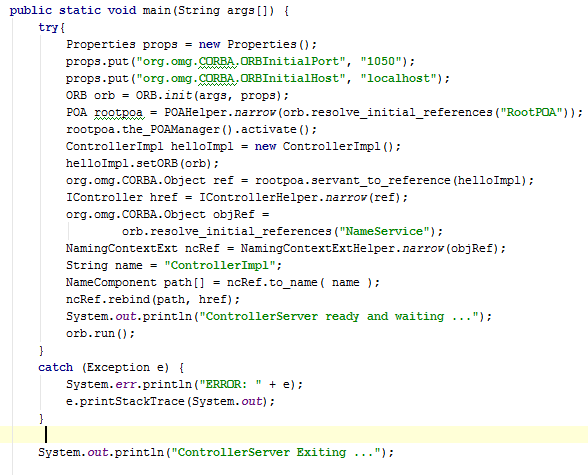
Реалізація RemoteHttp виглядає аналогічно із заміною відповідних класів.

## Етап 7

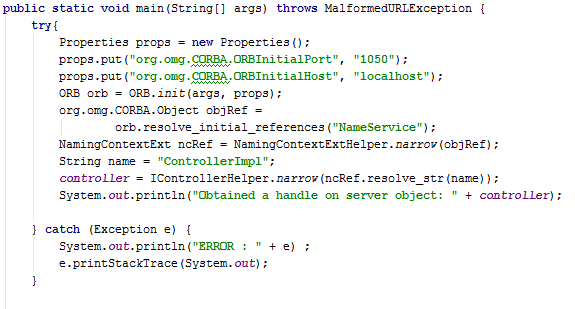
Варіант розподіленої версії на COM або CORBA.

*Виконання:* реалізація була виконана на CORBA.

Серверна частина:



Клієнтська частина має вигляд:



Скрипти збору:

version = '0.0'

apply plugin: 'java'

sourceCompatibility = 1.7

tasks.withType(Compile) {

options.encoding = 'UTF-8'

}

dependencies {

compile project(':core')

compile 'org.springframework:spring-context:3.2.4.RELEASE'

}

repositories {

mavenCentral()

}

task rmiPrepare << {

file("$buildDir/rmic").mkdirs();

}

task rmiDependencies(type: Sync) {

from configurations.compile

into "$buildDir/rmi-dependencies"

}

rmiDependencies.dependsOn rmiPrepare

task rmic(type:Exec) {

commandLine 'rmic', '-poa', '-iiop', '-d', "$buildDir/rmic/", '-classpath', "$buildDir/libs/\*;$buildDir/rmi-dependencies/\*", 'cyberwaste.kuzoff.corba.server.RemoteDatabaseManagerImpl'

}

rmic.dependsOn assemble

rmic.dependsOn rmiDependencies

build.dependsOn rmic

apply plugin:'application'

startScripts {

mainClassName = 'cyberwaste.kuzoff.corba.server.CorbaServerManager'

classpath.add(files { file("lib/rmic.jar") })

doLast {

unixScript.text = unixScript.text.replace('DEFAULT\_JVM\_OPTS=""', 'DEFAULT\_JVM\_OPTS="-Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory -Djava.naming.provider.url=iiop://localhost:1060"')

}

}

task rmiJar(type: Jar) {

from files { file("$buildDir/rmic") }

destinationDir = file("$buildDir/install/$project.name/lib")

archiveName = 'rmic.jar'

}

rmiJar.dependsOn installApp

rmiJar.dependsOn rmic

task installCorbaApp

installCorbaApp.dependsOn rmiJar

task orbd(type:Exec) {

commandLine 'sh', 'src/main/scripts/orbd', '&'

}

## Етап 8

Рефлексія. Реалізація динамічних викликів.

*Виконання*: рефлексія використана для динамічного завантаження команд і типів.

Динамічне завантаження команд:

**public** Command build() {

**try** {

String commandClassName = "cyberwaste.kuzoff.core.command.impl." + StringUtils.*capitalize*(verb) + StringUtils.*capitalize*(noun);

Class<?> commandClass = Class.*forName*(commandClassName);

**if** (commandClass == **null**) {

**return** **null**;

}

Command commandInstance = (Command) commandClass.newInstance();

commandInstance.setDatabaseManager(databaseManager);

**for** (Field field : commandClass.getDeclaredFields()) {

Argument annotation = field.getAnnotation(Argument.**class**);

**if** (annotation != **null**) {

field.setAccessible(**true**);

**if** (annotation.eager()) {

String[] argumentSuffix = (String[]) ArrayUtils.*subarray*(arguments, annotation.index(), arguments.length);

field.set(commandInstance, argumentSuffix);

} **else** {

**int** index = annotation.index();

field.set(commandInstance, arguments[index]);

}

field.setAccessible(**false**);

}

}

**return** commandInstance;

} **catch** (InstantiationException | IllegalAccessException | ClassNotFoundException e) {

**return** **null**;

}

}

Динамічне завантаження типів:

**public** **static** Type getType(String typeName) {

**try** {

String typeClassName = "cyberwaste.kuzoff.core.domain.types." + StringUtils.*capitalize*(typeName);

Class<?> typeClass = Class.*forName*(typeClassName);

**return** (Type) typeClass.newInstance();

} **catch** (ClassNotFoundException | InstantiationException | IllegalAccessException e) {

**throw** **new** IllegalArgumentException("Unknown type: " + typeName);

}

}

## Етап 9-10

Web-сервіси. Реалізація СУБД у вигляді Web-сервісу, а також реалізація клієнтської частини на альтернативній платформі.

*Виконання:* для реалізації був використаний фреймворк Spring REST.

Клієнтська частина транслює консольні команди у HTTP запити:

package cyberwaste.kuzoff.ws.client;

import java.util.Arrays;

import java.util.Collection;

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.apache.commons.lang.StringUtils;

import org.springframework.web.client.RestTemplate;

import cyberwaste.kuzoff.core.DatabaseManager;

import cyberwaste.kuzoff.core.domain.Row;

import cyberwaste.kuzoff.core.domain.Table;

public class WebServiceDatabaseManager implements DatabaseManager {

private final static String SERVICE\_URL = "http://localhost:8080/kuzoff-ws/api";

private RestTemplate restTemplate;

@Override

public Collection<Table> getAllTables() throws Exception {

Table[] result = restTemplate.getForObject(SERVICE\_URL + "/table", Table[].class, Collections.emptyMap());

return Arrays.asList(result);

}

@Override

public Table createTable(String name, String[] columnTypes) throws Exception {

Map<String, String> parameters = new HashMap<>();

parameters.put("name", name);

parameters.put("columnTypes", StringUtils.join(columnTypes, ','));

return restTemplate.postForObject(SERVICE\_URL + "/table/{name}?columnTypes={columnTypes}", null, Table.class, parameters);

}

@Override

public Table removeTable(String tableName) throws Exception {

Map<String, String> parameters = new HashMap<>();

parameters.put("tableName", tableName);

restTemplate.delete(SERVICE\_URL + "/table/{tableName}", parameters);

return new Table(tableName);

}

@Override

public void removeDatabase() throws Exception {

restTemplate.delete(SERVICE\_URL + "/database");

}

@Override

public Table getTable(String tableName) throws Exception {

Map<String, String> parameters = new HashMap<>();

parameters.put("tableName", tableName);

return restTemplate.getForObject(SERVICE\_URL + "/table/{tableName}", Table.class, parameters);

}

@Override

public Row insertRow(String tableName, String[] stringValues) throws Exception {

Map<String, String> parameters = new HashMap<>();

parameters.put("tableName", tableName);

parameters.put("data", StringUtils.join(stringValues, ','));

return restTemplate.postForObject(

SERVICE\_URL + "/table/{tableName}/data/?data={data}", null, Row.class, parameters

);

}

@Override

public List<Row> getRows(String tableName) throws Exception {

Map<String, String> parameters = new HashMap<>();

parameters.put("tableName", tableName);

Row[] result = restTemplate.getForObject(SERVICE\_URL + "/table/{tableName}/data", Row[].class, parameters);

return Arrays.asList(result);

}

@Override

public List<Row> removeRow(String tableName, String[] values) throws Exception {

Map<String, String> parameters = new HashMap<>();

parameters.put("tableName", tableName);

parameters.put("data", StringUtils.join(values, ','));

restTemplate.delete(SERVICE\_URL + "/table/{tableName}/data/?data={data}", parameters);

return Collections.emptyList();

}

@Override

public List<Row> removeDuplicates(String tableName) throws Exception {

Map<String, String> parameters = new HashMap<>();

parameters.put("tableName", tableName);

Row[] result = restTemplate.postForObject(SERVICE\_URL + "/table/{tableName}/unique", null, Row[].class, parameters);

return Arrays.asList(result);

}

@Override

public List<Row> difference(String tableName1, String tableName2) throws Exception {

Map<String, String> parameters = new HashMap<>();

parameters.put("tableName1", tableName1);

parameters.put("tableName2", tableName2);

Row[] result = restTemplate.postForObject(SERVICE\_URL + "/table/{tableName1}/diff/{tableName2}", null, Row[].class, parameters);

return Arrays.asList(result);

}

public void setRestTemplate(RestTemplate restTemplate) {

this.restTemplate = restTemplate;

}

}

Серверна частина отримує запити за допомогою Spring REST і делегує їх локальному database manager-у:

package cyberwaste.kuzoff.ws.server;

import java.util.Collection;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.bind.annotation.ResponseStatus;

import cyberwaste.kuzoff.core.DatabaseManager;

import cyberwaste.kuzoff.core.domain.Row;

import cyberwaste.kuzoff.core.domain.Table;

@Controller

@RequestMapping("/api")

public class WebServiceDatabaseManagerFacade implements DatabaseManager {

@Autowired

private DatabaseManager delegate;

@Override

@RequestMapping(value="/table", method=RequestMethod.GET, produces="application/json")

public @ResponseBody Collection<Table> getAllTables() throws Exception {

return delegate.getAllTables();

}

@Override

@RequestMapping(value="/table/{tableName}", method=RequestMethod.POST, produces="application/json")

@ResponseStatus(HttpStatus.CREATED)

public @ResponseBody Table createTable(@PathVariable("tableName") String name, @RequestParam("columnTypes") String[] columnTypes) throws Exception {

return delegate.createTable(name, columnTypes);

}

@Override

@RequestMapping(value="/database", method=RequestMethod.DELETE)

@ResponseStatus(HttpStatus.NO\_CONTENT)

public @ResponseBody void removeDatabase() throws Exception {

delegate.removeDatabase();

}

@Override

@RequestMapping(value="/table/{tableName}", method=RequestMethod.GET, produces="application/json")

public @ResponseBody Table getTable(@PathVariable("tableName") String tableName) throws Exception {

return delegate.getTable(tableName);

}

@Override

@RequestMapping(value="/table/{tableName}/data", method=RequestMethod.POST, produces="application/json")

@ResponseStatus(HttpStatus.CREATED)

public @ResponseBody Row insertRow(@PathVariable("tableName") String tableName, @RequestParam("data") String[] stringValues) throws Exception {

return delegate.insertRow(tableName, stringValues);

}

@Override

@RequestMapping(value="/table/{tableName}/data", method=RequestMethod.GET, produces="application/json")

public @ResponseBody List<Row> getRows(@PathVariable("tableName") String tableName) throws Exception {

return delegate.getRows(tableName);

}

@Override

@RequestMapping(value="/table/{tableName}/data", method=RequestMethod.DELETE, produces="application/json")

@ResponseStatus(HttpStatus.NO\_CONTENT)

public @ResponseBody List<Row> removeRow(@PathVariable("tableName") String tableName, @RequestParam("data") String[] values) throws Exception {

return delegate.removeRow(tableName, values);

}

@Override

@RequestMapping(value="/table/{tableName}/unique", method=RequestMethod.POST, produces="application/json")

public @ResponseBody List<Row> removeDuplicates(@PathVariable("tableName") String tableName) throws Exception {

return delegate.removeDuplicates(tableName);

}

@Override

@RequestMapping(value="/table/{tableName}", method=RequestMethod.DELETE, produces="application/json")

@ResponseStatus(HttpStatus.NO\_CONTENT)

public @ResponseBody Table removeTable(@PathVariable("tableName") String tableName) throws Exception {

return delegate.removeTable(tableName);

}

@Override

@RequestMapping(value="/table/{tableName1}/diff/{tableName2}", method=RequestMethod.POST, produces="application/json")

public @ResponseBody List<Row> difference(@PathVariable("tableName1") String tableName1, @PathVariable("tableName2") String tableName2) throws Exception {

return delegate.difference(tableName1, tableName2);

}

public void setDelegate(DatabaseManager delegate) {

this.delegate = delegate;

}

}

## Етап 11

Клієнт web-сервісу на альтернативній платформі.

*Виконання:* було використано мову програмування Groovy.

Код програми:

**package** cyberwaste.kuzoff.ws.groovyclient

**import** groovyx.net.http.RESTClient

**class** WebServiceGroovyClient {

**static** **def** *client* = **new** RESTClient('http://localhost:8080/kuzoff-ws/api/')

**static** *main*(args) {

**while** (**true**) {

**def** command = System.*console*().readLine ">>>"

**if** (command == 'exit') **return**

**def** commandTokens = command.split(" ")

**if** (commandTokens[0] == 'show' && commandTokens[1] == 'database') {

*showDatabase*();

}

**if** (commandTokens[0] == 'show' && commandTokens[1] == 'table') {

*showTable*(commandTokens[2]);

}

**if** (commandTokens[0] == 'select' && commandTokens[1] == 'table') {

*selectTable*(commandTokens[2]);

}

**if** (commandTokens[0] == 'unique' && commandTokens[1] == 'table') {

*uniqueTable*(commandTokens[2]);

}

}

}

**static** *showDatabase*() {

**try** {

**def** result = *client*.get(path : "table")

*println* "Found ${result.data.*size*()} tables"

**for** (**def** table : result.data) {

*println* "${table.name}"

}

} **catch** (e) {

*println* e.message

}

}

**static** *showTable*(**def** tableName) {

**try** {

**def** result = *client*.get(path : "table/${tableName}")

*print* "table ${result.data.name} ("

**for** (**def** column : result.data.typeNames) {

*print* "${column}"

}

*println* ")"

} **catch** (e) {

*println* e.message

}

}

**static** *selectTable*(**def** tableName) {

**try** {

**def** result = *client*.get(path : "table/${tableName}*/data"*)

*println* "Found ${result.data.values.*size*()} rows"

**for** (**def** row : result.data) {

**for** (**def** value : row.values) {

*print* "${value.valueAsString} "

}

*println*()

}

} **catch** (e) {

*println* e.message

}

}

**static** *uniqueTable*(**def** tableName) {

**try** {

**def** result = *client*.post(path : "table/${tableName}*/unique"*)

*println* "Found ${result.data.values.*size*()} rows"

**for** (**def** row : result.data) {

**for** (**def** value : row.values) {

*print* "${value.valueAsString} "

}

*println*()

}

} **catch** (e) {

*println* e.message

}

}

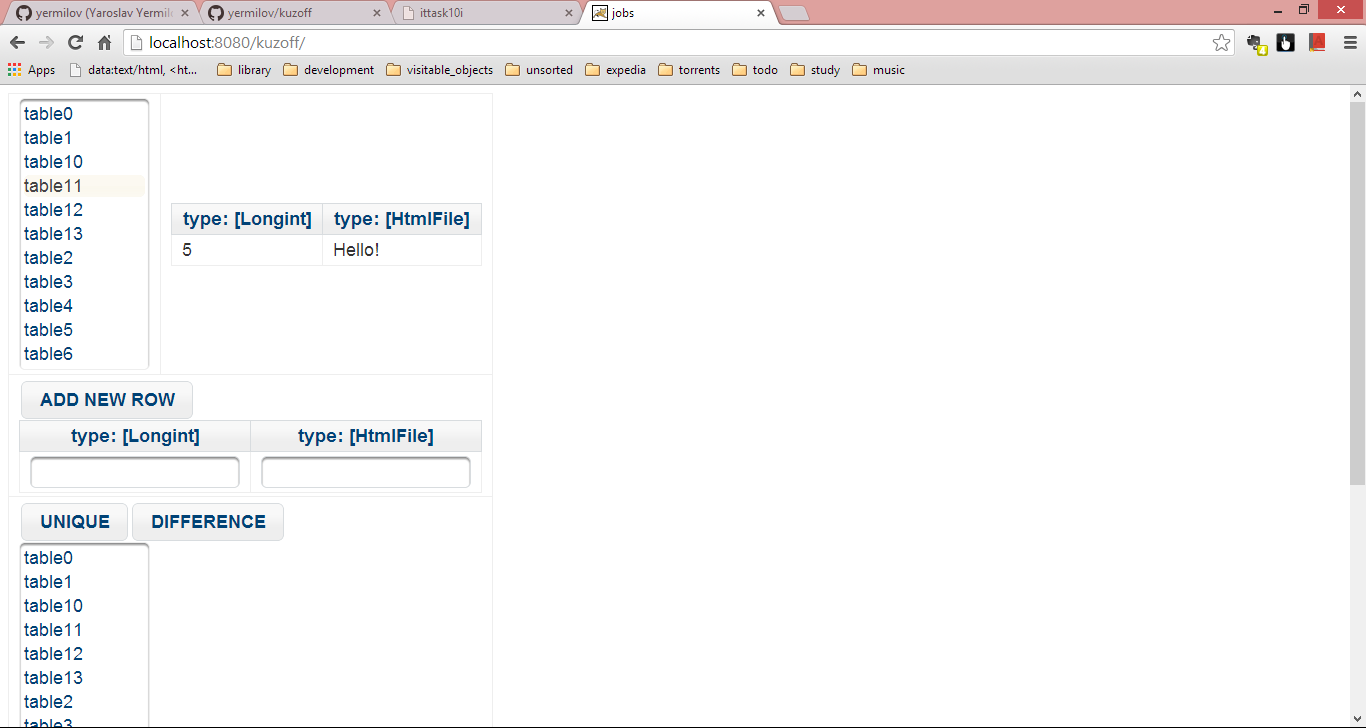
}

## Етап 12-13

Web-проект з використанням AJAX

*Виконання:* було обрано фреймворки Spring та JSF.

Основна сторінка має такий вигляд:



Веб-сторінка:

<!DOCTYPE html>

<html

xmlns=*"http://www.w3c.org/1999/xhtml"*

xmlns:f=*"http://java.sun.com/jsf/core"*

xmlns:h=*"http://java.sun.com/jsf/html"*

xmlns:p=*"http://primefaces.org/ui"*>

<h:head>

<title>

jobs

</title>

</h:head>

<h:body>

<p:growl id=*"growl"* showDetail=*"true"* sticky=*"true"* autoUpdate=*"true"* />

<h:form>

<p:panelGrid>

<p:row>

<p:column>

<p:selectOneListbox id=*"tables"* value=*"#{jobsWebApp.primaryTable}"*>

<f:selectItems value=*"#{jobsWebApp.allTables}"* />

<p:ajax event=*"change"* update=*"table new-row"* />

</p:selectOneListbox>

</p:column>

<p:column>

<p:dataTable id=*"table"* var=*"row"* value=*"#{jobsWebApp.tableData}"*>

<p:columns value=*"#{jobsWebApp.tableColumnTypes}"* var=*"column"* columnIndexVar=*"colIndex"*>

<f:facet name=*"header"*>

type: [#{column}]

</f:facet>

<h:outputText value=*"#{row.values[colIndex].valueAsString}"* />

</p:columns>

</p:dataTable>

</p:column>

</p:row>

<p:row>

<p:column colspan=*"2"*>

<p:commandButton value=*"ADD NEW ROW"* update=*"table new-row"* actionListener=*"#{jobsWebApp.addRow}"* ajax=*"true"* />

<br/>

<p:dataTable id=*"new-row"* var=*"newRow"* value=*"#{jobsWebApp.newRowValues}"*>

<p:columns value=*"#{jobsWebApp.tableColumnTypes}"* var=*"column"* columnIndexVar=*"colIndex"*>

<f:facet name=*"header"*>

type: [#{column}]

</f:facet>

<p:inputText value=*"#{newRow[colIndex]}"* />

</p:columns>

</p:dataTable>

</p:column>

</p:row>

<p:row>

<p:column colspan=*"2"*>

<p:commandButton

value=*"UNIQUE"* actionListener=*"#{jobsWebApp.unique}"* ajax=*"true"*

update=*"tables tables-for-operation table new-row"* />

<p:commandButton

value=*"DIFFERENCE"* actionListener=*"#{jobsWebApp.diff}"* ajax=*"true"*

update=*"tables tables-for-operation table new-row"* />

<p:selectOneListbox id=*"tables-for-operation"* value=*"#{jobsWebApp.secondaryTable}"*>

<f:selectItems value=*"#{jobsWebApp.allTables}"* />

<p:ajax event=*"change"* update=*"table new-row"* />

</p:selectOneListbox>

</p:column>

</p:row>

<p:row>

<p:column>

<a href=*"http://localhost:8080/kuzoff/view/ritchie.xhtml"*>

<img src=*"http://media.boingboing.net/wp-content/uploads/2011/10/dennis\_ritchie.jpg"* width=*"100px"*/>

</a>

</p:column>

<p:column>

<a href=*"http://localhost:8080/kuzoff/view/jobs.xhtml"*>

<img src=*"http://tabtimes.com/ckfinder/userfiles/images/TWIT/Steve-Job-Ipad.jpg"* width=*"300px"*/>

</a>

</p:column>

</p:row>

</p:panelGrid>

</h:form>

</h:body>

</html>

Backing-bean сторінки:

package cyberwaste.kuzoff.web.jsf;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collection;

import java.util.Collections;

import java.util.List;

import cyberwaste.kuzoff.core.DatabaseManager;

import cyberwaste.kuzoff.core.domain.Row;

import cyberwaste.kuzoff.core.domain.Table;

public class JobsWebApp {

private DatabaseManager databaseManager;

private String primaryTable;

private String secondaryTable;

private List<String> newRowValues;

public String getPrimaryTable() {

return primaryTable;

}

public void setPrimaryTable(String primaryTable) throws IOException {

this.primaryTable = primaryTable;

clearNewRowValue();

}

public String getSecondaryTable() {

return secondaryTable;

}

public void setSecondaryTable(String secondaryTable) {

this.secondaryTable = secondaryTable;

}

public Collection<String> getAllTables() throws IOException {

try {

Collection<String> tables = new ArrayList<String>();

for (Table table : databaseManager.getAllTables()) {

tables.add(table.getName());

}

return tables;

} catch (Exception e) {

return Collections.emptyList();

}

}

public List<Row> getTableData() throws IOException {

try {

return databaseManager.getRows(primaryTable);

} catch (Exception e) {

return Collections.emptyList();

}

}

public List<String> getTableColumnTypes() throws IOException {

try {

return Arrays.asList(databaseManager.getTable(primaryTable).getTypeNames());

} catch (Exception e) {

return Collections.emptyList();

}

}

public List<List<String>> getNewRowValues() {

return Collections.singletonList(newRowValues);

}

public void setNewRow(List<List<String>> newRow) {

this.newRowValues = newRow.get(0);

}

public void addRow() {

try {

databaseManager.insertRow(primaryTable, newRowValues.toArray(new String[0]));

clearNewRowValue();

} catch (Exception e) {

throw new RuntimeException(e);

}

}

public void unique() {

try {

databaseManager.removeDuplicates(primaryTable);

clearNewRowValue();

} catch (Exception e) {

throw new RuntimeException(e);

}

}

public void diff() {

try {

databaseManager.difference(primaryTable, secondaryTable);

clearNewRowValue();

} catch (Exception e) {

throw new RuntimeException(e);

}

}

private List<String> clearNewRowValue() throws IOException {

newRowValues = new ArrayList<String>();

for (int i = 0; i < getTableColumnTypes().size(); i++) {

newRowValues.add("");

}

return newRowValues;

}

public void setDatabaseManager(DatabaseManager databaseManager) {

this.databaseManager = databaseManager;

}

}

## Етап 14

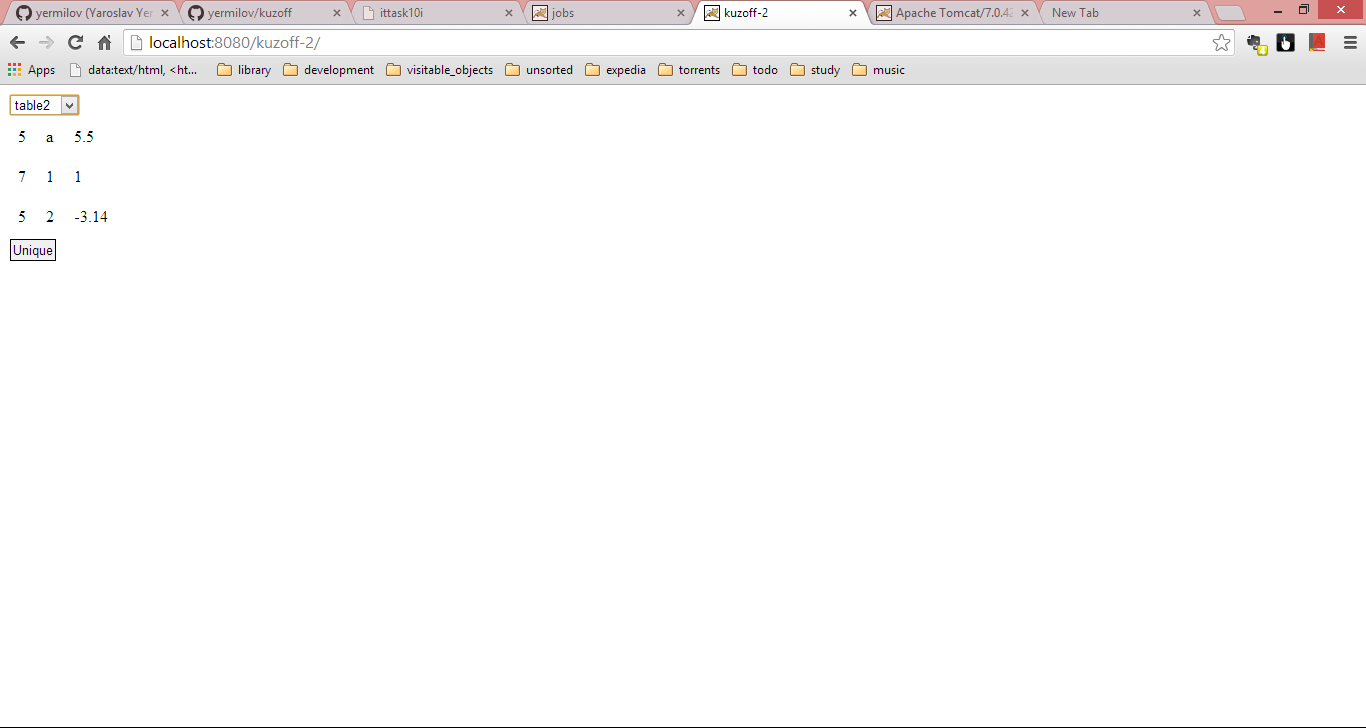
Пропущено.

## Етап 15

Web-проект з використанням альтернативної технології.

*Виконання:* було обрано jQuery.

Основна сторінка має такий вигляд:



Веб-сторінка:

<!DOCTYPE html>

<html>

<head>

<title>kuzoff-2</title>

<script src="js/jquery-1.7.js" type="text/javascript"></script>

<script src="js/jquery.tmpl.js" type="text/javascript"></script>

<script src="js/main.js" type="text/javascript"></script>

<link rel="stylesheet" type="text/css" href="css/styles.css"/>

<script id="tablesTmpl" type="text/x-jquery-tmpl">

<option value="${name}">${name}</option>

</script>

<script id="rowTmpl" type="text/x-jquery-tmpl">

<div class="drow">

{{tmpl($data.values) "#cellTmpl"}}

</div>

</script>

<script id="cellTmpl" type="text/x-jquery-tmpl">

<div class="dcell">

${valueAsString}

</div>

</script>

</head>

<body>

<select id="selectTable">

</select>

<div>

<div id="tableData" class="dtable">

</div>

</div>

<input id="buttonUnique" type="submit" value="Unique" />

</body>

</html>

JavaScript код:

$(document).ready(function() {

$.get("http://localhost:8080/kuzoff-ws/api/table", null,

function(responseData) {

$('#tablesTmpl').tmpl(responseData).appendTo('#selectTable');

loadDataForTable($('#selectTable').val());

}

);

$('#selectTable').on('change', function() {

loadDataForTable($(this).val());

});

$("#buttonUnique").click(function(e){

$.post("http://localhost:8080/kuzoff-ws/api/table/" + $('#selectTable').val() + "/unique", null,

function(responseData) {

$('#tableData').empty();

$('#rowTmpl').tmpl(responseData).appendTo('#tableData');

}

);

e.preventDefault();

});

});

function loadDataForTable(tableName) {

$.get("http://localhost:8080/kuzoff-ws/api/table/" + tableName + "/data", null,

function(responseData) {

$('#tableData').empty();

$('#rowTmpl').tmpl(responseData).appendTo('#tableData');

}

);

}