**Київський національний університет імені Тараса Шевченка  
Факультет комп’ютерних наук та кібернетики**

Звіт  
про виконання лабораторної роботи на тему

«Фрагментарна реалізація систем управління табличними базами даних»

з курсу «Інформаційні технології»

Виконала:

студентка 4 курсу групи МІ-4  
Чучук Ольга

Київ – 2017

**Постановка задачі:**

1. Загальні вимоги

Основні вимоги щодо структури бази:

* кількість таблиць принципово не обмежена (реляції між таблицями не враховувати);
* кількість полів та кількість записів у кожній таблиці також принципово не обмежені.

У кожній роботі треба забезпечити підтримку (для полів у таблицях) наступних типів:

* integer
* real
* char

Також у кожній роботі треба реалізувати функціональну підтримку для:

* створення бази;
* створення та знищення таблиці з бази;
* перегляду та редагування рядків таблиці;
* збереження табличної бази на диску та, навпаки, зчитування її з диску.

2. Індивідуальний варіант

Потрібно забезпечити підтримку (для можливого використання у таблицях) додаткових типів:

* picture-файли (один з форматів)
* realInvl (інтервальний тип)

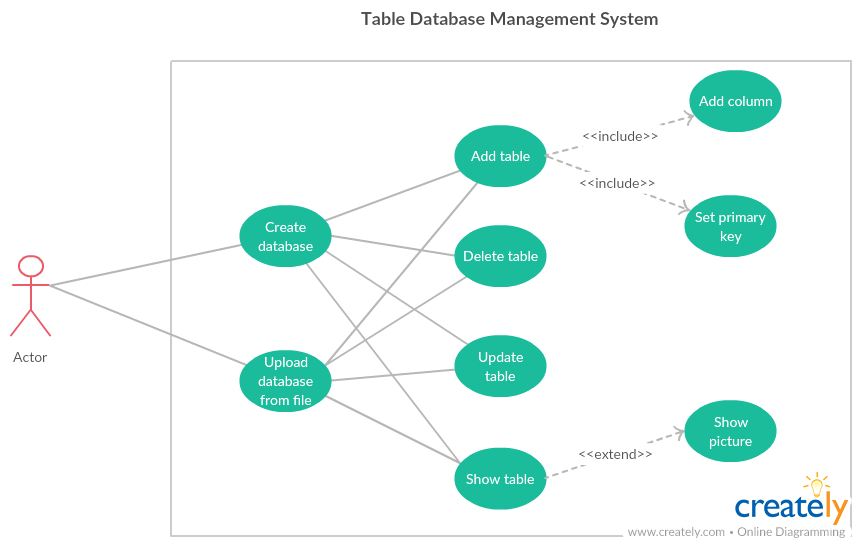
Потрібно реалізувати операцiї над таблицями у відповідності з варіантом:

* рiзниця таблиць

**Виконання етапів:**

***Попередній етап***

Функціональна специфікація системи управління табличними базами даних (СУТБД) у вигляді однієї або кількох діаграм прецедентів UML.



***Перший та другий етапи***

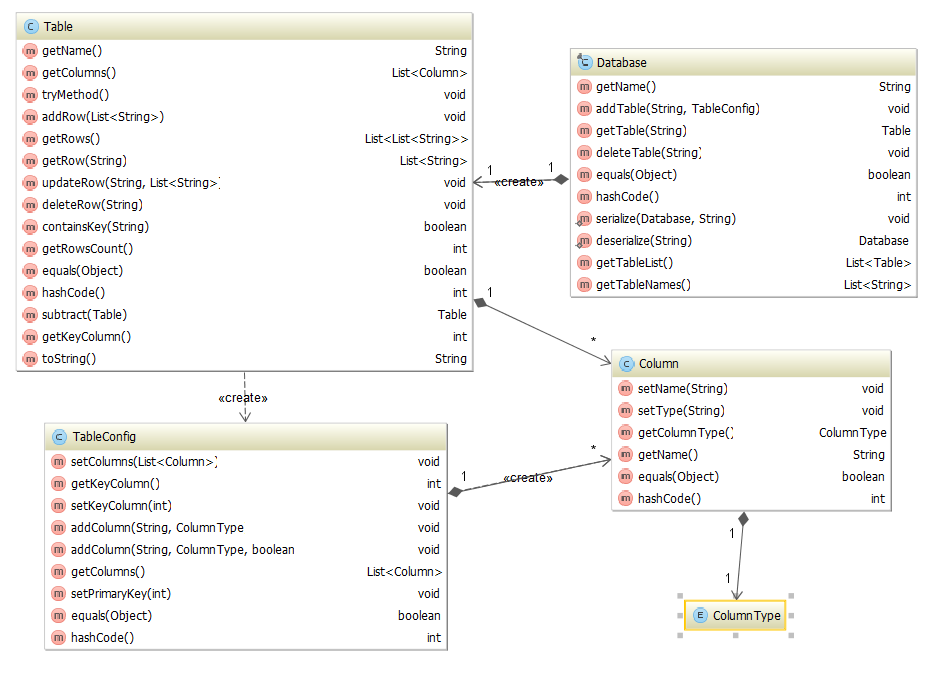
Розробка локальної (нерозподіленої) версії СУТБД (із власною реалізацією класів "Таблиця" та "База").

Обов'язкові при виконанні пункти:

* Розробка власних класів для понять "Таблиця", "База" та, можливо, деяких інших класів, спряжених із поняттям "Таблиця" (наприклад, "Схема таблиці", "Атрибут", "Рядок таблиці" тощо).
* Створення UML-діаграми класів (з наявними між класами відношеннями).
* Проведення unit-тестування. Надати 3..\* тести, один з яких має бути призначеним для тестуванням “індивідуальної” (варіантної) операції з розділу 2.
* Забезпечення дружнього інтерфейсу системи.

*Виконання:*

* реалізовано на Java, розроблено власні класи Database, Table, TableConfig, Column, ColumnType
* UML-діаграма класів



* Розроблено Unit-тести на функціональність «Знайти різницю між таблицями»

@Test  
**public void** subtractTablesNonEmptyResult() {  
 TableConfig config = **new** TableConfig();  
 config.addColumn(**"id"**, ColumnType.***IntegerType***, **true**);  
 config.addColumn(**"Char column"**, ColumnType.***CharType***, **false**);  
 config.addColumn(**"Picture column"**, ColumnType.***PictureType***, **false**);  
 Table table1 = **new** Table(**"Table1"**, config);  
 Table table2 = **new** Table(**"Table2"**, config);  
 List<String> row1 = Arrays.*asList*(**"1"**, **"a"**, **"http://worldartsme.com/images/123-clipart-1.jpg"**);  
 List<String> row2 = Arrays.*asList*(**"2"**, **"b"**,  
 **"https://upload.wikimedia.org/wikipedia/commons/thumb/e/e1/NY-234.svg/750px-NY-234.svg.png"**);  
 List<String> row3 = Arrays.*asList*(**"3"**, **"c"**,  
 **"http://worldartsme.com/images/123-clipart-1.jpg"**);  
 table1.addRow(row1);  
 table1.addRow(row2);  
 table2.addRow(row1);  
 table2.addRow(row3);  
  
 Table table3 = table1.subtract(table2);  
  
 *assertEquals*(1, table3.getRowsCount());  
 *assertEquals*(row2, table3.getRows().get(0));  
  
 Table table4 = table2.subtract(table1);  
  
 *assertEquals*(1, table4.getRowsCount());  
 *assertEquals*(row3, table4.getRows().get(0));  
}

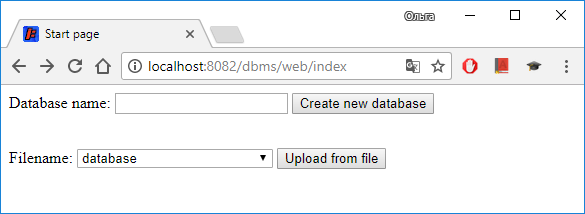
* Використано фреймворк Spring для створення об'єктів типу Database, Table, Column

***Третій етап***

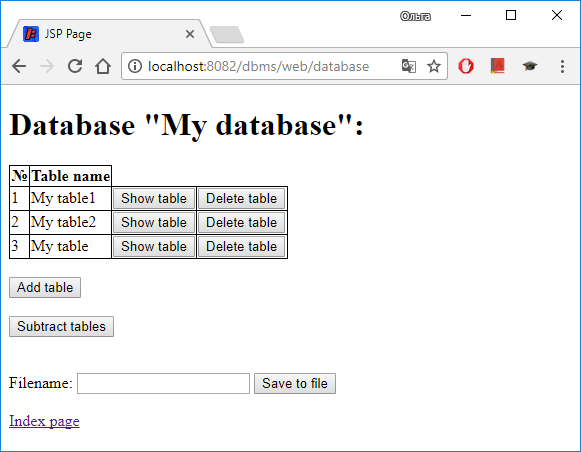
Web-проект. (Технології на вибір: ASP .NET, ASP .NET MVC, WPF, JSP, JavaServlet та інші, у тому числі на основі фреймворків Spring, Struts, Struts 2, JSF, Tapestry, Wicket, GWT тощо).

*Виконання:* реалізовано за допомогою фреймворка SpringMVC. Використовуються JSP сторінки.

*Початкова сторінка:*



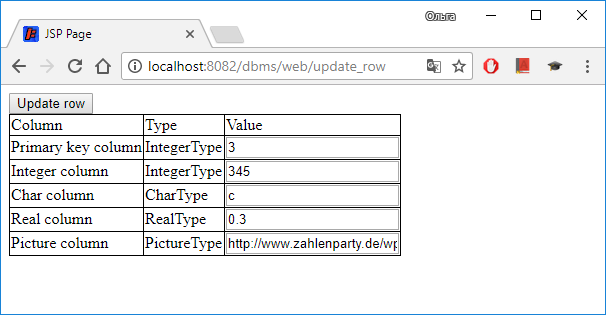
*Представлення бази:*

**

*Представлення таблиці:*



*Можливість змінити рядок:*

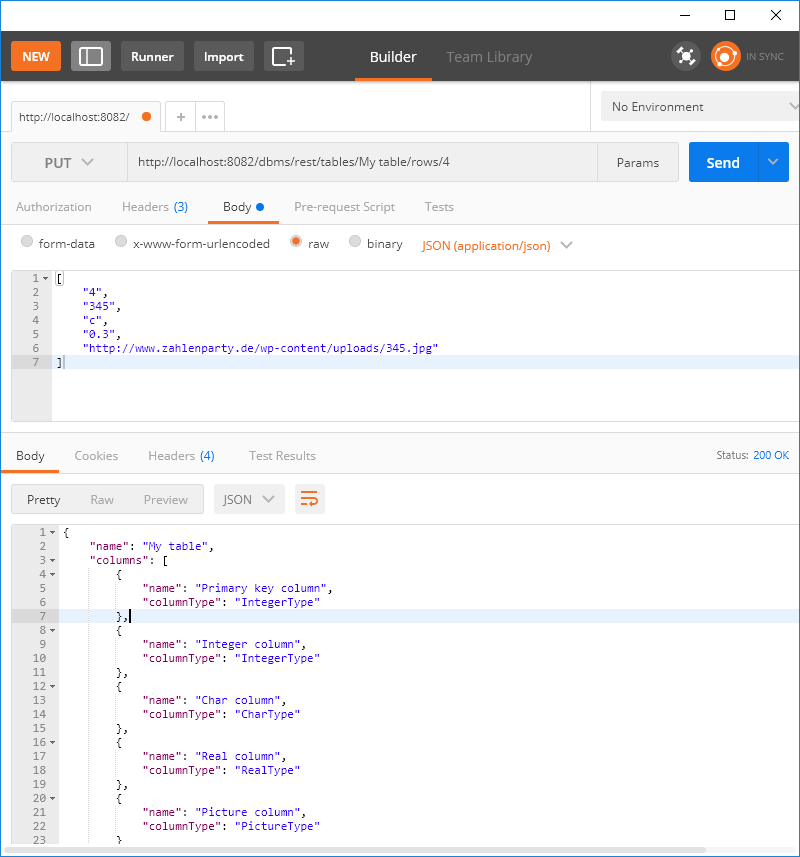


*Відображення picture-файлів:*

***Четвертий етап:***

REST web-сервіси. Реалізація СУТБД на основі технології REST web-сервісів (сервер, набір тестових клієнтських запитів).

*Виконання:* існує можливість доступу до бази, таблиць, рядків (перегляд, створення, модифікація, видалення). Також підтримуються можливість віднімати таблиці з однаковою структурою. Комунікація з web-сервісом відбувається за допомогою JSON-файлів.



***П’ятий та шостий етапи:***

Два варіанти розподілених версій системи (із реалізацією програм-клієнтів та програм-серверів), використовуючи за власним вибором будь-які дві з наступних технологій: Java RMI/JRMP, Java RMI/IIOP, Net Remoting, WCF, IIOP Net, EJB тощо.

*Виконання:*

Розроблено інтерфейс RmiDatabase, що наслідує інтерфейс Remote

**public interface** RmiDatabase **extends** Remote {  
 String getName() **throws** RemoteException;  
 List<String> getTableNames() **throws** RemoteException;  
 Table getTable(String tableName) **throws** RemoteException;  
 Table subtract(String tableName1, String tableName2) **throws** RemoteException;  
}

1. Java RMI/JRMP – зв’язування відбувається за допомогою rmirgistry
   1. Сервер:

**public class** RmiServer **extends** RemoteObject **implements** RmiDatabase{

// implementation here...  
  
 **public static void** main(String[] args) {  
 **try** {  
 String name = **"Database"**;  
 RmiDatabase database = **new** RmiServer();  
 RmiDatabase stub = (RmiDatabase)  
 UnicastRemoteObject.*exportObject*(database,0);  
 Registry registry = LocateRegistry.*getRegistry*();  
 registry.rebind(name, stub);  
 System.***out***.println(**"Working"**);  
 } **catch** (RemoteException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
}

* 1. Клієнт:

**public class** RmiClient {  
 **public static void** main(String[] args) {  
 **try** {  
 String name = **"Database"**;  
 Registry registry = LocateRegistry.*getRegistry*();  
 RmiDatabase database = (RmiDatabase) registry.lookup(name);

// using obtained database...

} **catch** (NotBoundException | RemoteException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
}

1. Java RMI/IIOP – зв’язування відбувається за допомогою orbd
   1. Сервер

**public class** RmiServer **extends** PortableRemoteObject **implements** RmiDatabase{  
  
 // implementation here...  
  
 **public static void** main(String[] args) {  
 **try** {  
 String name = **"Database"**;  
 RmiDatabase database = **new** RmiServer();  
 Context context = **new** InitialContext();  
 context.rebind(name, database);  
 System.***out***.println(**"Working"**);  
 } **catch** (Exception e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
}

* 1. Клієнт:

**public class** RmiClient {  
  
 **public static void** main(String[] args) {  
 String name = **"Database"**;  
 **try** {  
 Context context = **new** InitialContext();  
 Object obj = context.lookup(name);  
 RmiDatabase database = (RmiDatabase) PortableRemoteObject.*narrow*(obj, RmiDatabase.**class**);  
  
 //using obtained database...

} **catch** (Exception e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
}

***Сьомий етап:***

Варіант розподіленої версії з використанням COM або CORBA (сервер, клієнт).

*Виконання:*

За допомогою rmic компілятора стрворено файл RmiDatabase.idl, за допомогою idlj компілятора з нього згенеровано допоміжні стаби та скелетони:

\_RmiDatabaseStub

RmiDatabase

RmiDatabaseHelper

RmiDatabaseHolder

RmiDatabaseOperations

RmiDatabasePOA

1. Сервер:

**public class** CorbaServer **extends** RmiDatabasePOA {  
  
 **public static void** main(String[] args) {  
 **try** {  
 ORB orb = ORB.*init*(args, **null**);  
 POA rootPoa = POAHelper.*narrow*(  
 orb.resolve\_initial\_references(**"RootPOA"**));  
 rootPoa.the\_POAManager().activate();  
 CorbaServer database = **new** CorbaServer();  
 Object ref = rootPoa.servant\_to\_reference(database);  
 RmiDatabase href = RmiDatabaseHelper.*narrow*(ref);  
 Object objRef = orb.resolve\_initial\_references(**"NameService"**);  
 NamingContextExt ncRef = NamingContextExtHelper.*narrow*(objRef);  
 String name = **"Database"**;  
 NameComponent[] path = ncRef.to\_name(name);  
 ncRef.rebind(path, href);  
 System.***out***.println(**"Working"**);  
 orb.run();  
 } **catch** (Exception e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
}

1. Клієнт:

**public class** CorbaClient {  
 **public static void** main(String[] args) {  
 **try** {  
 ORB orb = ORB.*init*(args, **null**);  
 Object objRef = orb.resolve\_initial\_references(**"NameService"**);  
 NamingContextExt ncRef = NamingContextExtHelper.*narrow*(objRef);  
 String name = **"Database"**;  
 RmiDatabase database = RmiDatabaseHelper.*narrow*(ncRef.resolve\_str(name));  
  
 //using obtained database...

} **catch** (Exception e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
}

***Восьмий етап:***

Сумісність технологій. Серед можливих варіантів пропонуються такі: ASMX web-сервіси - WCF, Java RMI/IIOP - CORBA). Треба, наприклад, для ASMX web-сервісу (етап 6) розробити WCF-клієнт або, навпаки, для WCF-сервісу - ASMX-клієнт.

*Виконання:*

Для RMI сервера розроблено СORBA клієнт

Створено інтерфейс Database, за допомогою компіляторів rmic та idlj згенеровано необхідні для RMI та CORBA допоміжні файли.

1. Сервер:

**public class** RmiServer **extends** PortableRemoteObject **implements** Database {

// implementation here…  
  
 **public static void** main(String[] args) {  
 **try** {  
 String name = **"Database"**;  
 Database database = **new** RmiServer();  
 Context context = **new** InitialContext();  
 context.rebind(name, database);  
 System.***out***.println(**"Working"**);  
 } **catch** (Exception e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
}

1. Клієнт:

**public class** CorbaClient {  
 **public static void** main(String[] args) {  
 **try** {  
 ORB orb = ORB.*init*(args, **null**);  
 Object objRef = orb.resolve\_initial\_references(**"NameService"**);  
 NamingContextExt ncRef = NamingContextExtHelper.*narrow*(objRef);  
 String name = **"Database"**;  
 RmiDatabase database = RmiDatabaseHelper.*narrow*(ncRef.resolve\_str(name));  
  
 //using obtained database...

} **catch** (Exception e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
}

***Дев'ятий етап:***

Web-проект із використанням AJAX.

*Виконання:*

Конструювання сторінки у веб-додатку відбувається з використанням технології AJAX.

$(**function** () {  
  
 **var** $columns = $(**'#columns'**);  
 **var** $columnName = $(**'#columnName'**);  
 **var** $columnType = $(**'#columnType'**);  
  
 $.ajax({  
 **type**: **'PUT'**,  
 **url**: **'../rest/config'**,  
 success: **function** () {  
 **console**.log(**'initialized new tableConfig'**);  
 }  
 })  
 $(**'#addButton'**).on(**'click'**, **function** () {  
 $.ajax({  
 **type**: **'POST'**,  
 **url**: **'../rest/config?columnName='** + $columnName.val() +  
 **'&columnType='** + $columnType.val(),  
 success: **function** () {  
 **console**.log(**'wrote to the db'**);  
 }  
 })  
 $columns.append(  
 **'<tr><td>'** + $columnName.val() + **'</td><td>'** + $columnType.val() + **'</td></tr>'** )  
 $columnName.val(**''**);  
 $columnType.val(**'IntegerType'**);  
 })  
});

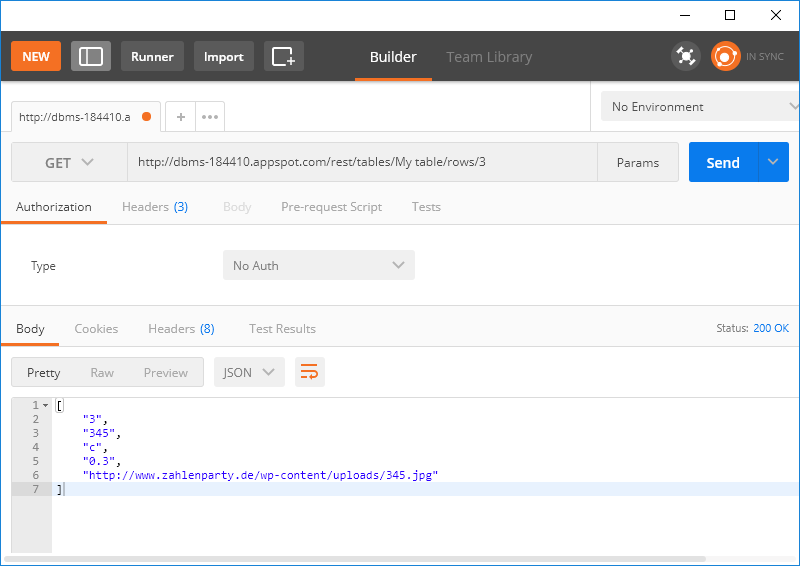
***Десятий етап:***

Варіант проекту із використанням хмарних технологій (Microsoft Azure, Google App Engine, Amazon Web Services тощо).

*Виконання:*

Розгорнуто REST web-сервіс використовуючи Google App Engine.

Доступ до працюючого сервісу : <http://dbms-184410.appspot.com/rest/>



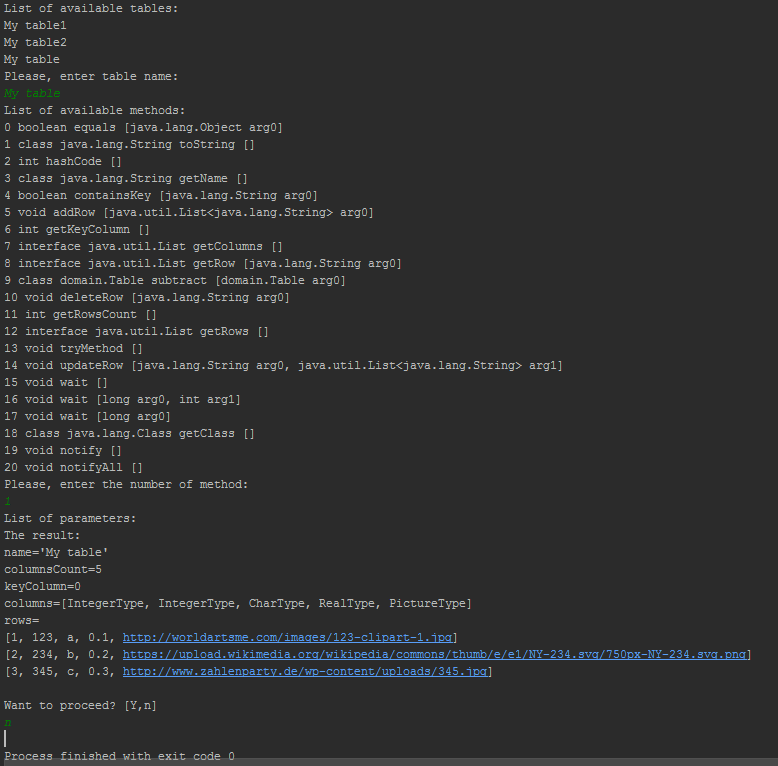
***Додаткове завдання:***

Рефлексія (reflection). Реалізація "динамічних викликів" на прикладі одного з об'єктів клієнтської частини розподіленої версії.

*Виконання:*

Реалізована можливість викликів методів над об’єктами класу Table. Послідовність дій:

1. Убрати таблицю для виконання виклику методу
2. Обрати метод серед існуючих ( у консоль виводиться список усіх методів)
3. Задати необхідні параметри
4. Отримуємо результат обчислень

Приклад роботи програми: