Київський національний університет імені Тараса Шевченка

Факультет кібернетики

Звіт до лабораторної роботи на тему:

**«Фрагментарна реалізація системи управління табличними базами даних»**

Студентів I курсу

ОКР «Магістр»

Спеціальності «Соціальна інформатика»

Савицької Катерини

Чернова Олега

Київ – 2013

**Зміст**

1. Постановка задачі
2. Діаграма прецедентів
3. Діаграми класів, взаємодії, компонентів, розгортання, VPOC - діаграми
4. Локальна версія програми
5. Рефлексія
6. Розподілена версія програми : RMI
7. Веб-сервіс
8. Допомога користувачу

**Постановка задачі**

В даній роботі необхідно було розробити програмний продукт для роботи з базами даних. З допомогою даного продукту користувач повинен мати змогу не лише відкривати та переглядати бази даних, що розміщені в нього на компьютері, а також створювати, редагувати та видаляти бази даних (окремі рядки або ж цілі таблиці).

В кожній базі даних має бути реалізована підтримка наступних типів полів:

* integer;
* real;
* char;
* перелiчуваний тип (множину значень складає набiр рядкiв).

Також користовач повинен мати змогу перейменовувати колонки, шукати рядки у таблиці, вказавши певний пошуковий шаблон, а також переглядати отримані результати пошуку.

Дана робота бута виконана в кілька етапів:

1) Підготовчий етап. Функціональна специфікація системи управління табличними базами даних (СУТБД) у вигляді діаграм прецедентів.

2) Проектування програми з використанням UML-діаграм (12 штук).

* діаграми прецедентів
* діаграми класів
* діаграми діяльності
* діаграми взаємодії
* діаграми станів
* діаграми компонентів
* діаграми розгортання

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

4) Розподілена версія системи (з реалізацією програм-клієнтів та програм-серверів) з використанням технології Java RMI/JRMP

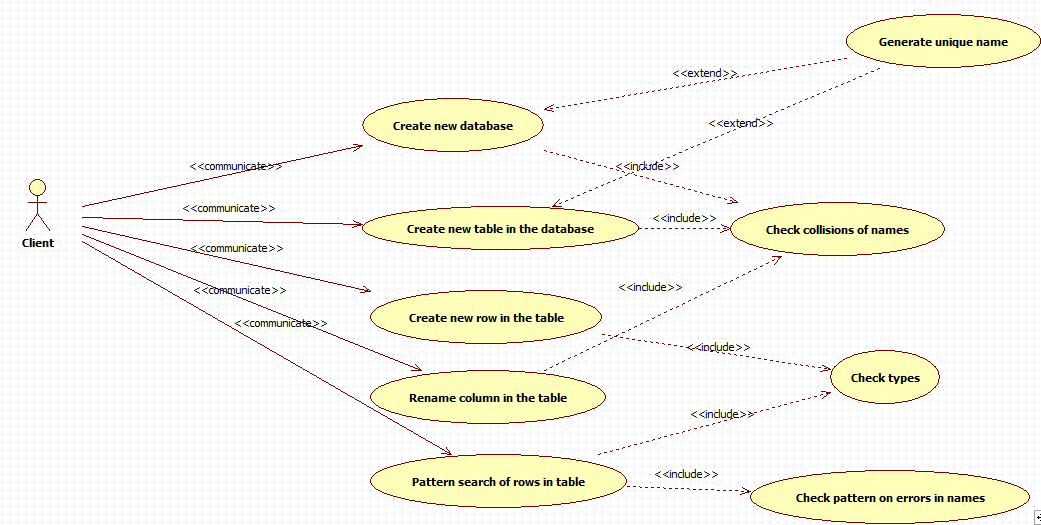
5) Розподілена версія системи (з реалізацією програм-клієнтів та програм-серверів)  з використанням технології Java RMI/IIOP.

6) Реалізація рефлексії ("динамічні виклики" на прикладі одного з об'єктів клієнтської частини розподіленої версії).

7) Розробка Web-сервісу. Реалізація СУТБД на основі технології web-сервісів (сервер, клієнт).

**Діаграма прецедентів**

Для аналізу вимог до програми згідно з варіантами лабораторної роботи була розроблена наступна діаграма прецедентів:



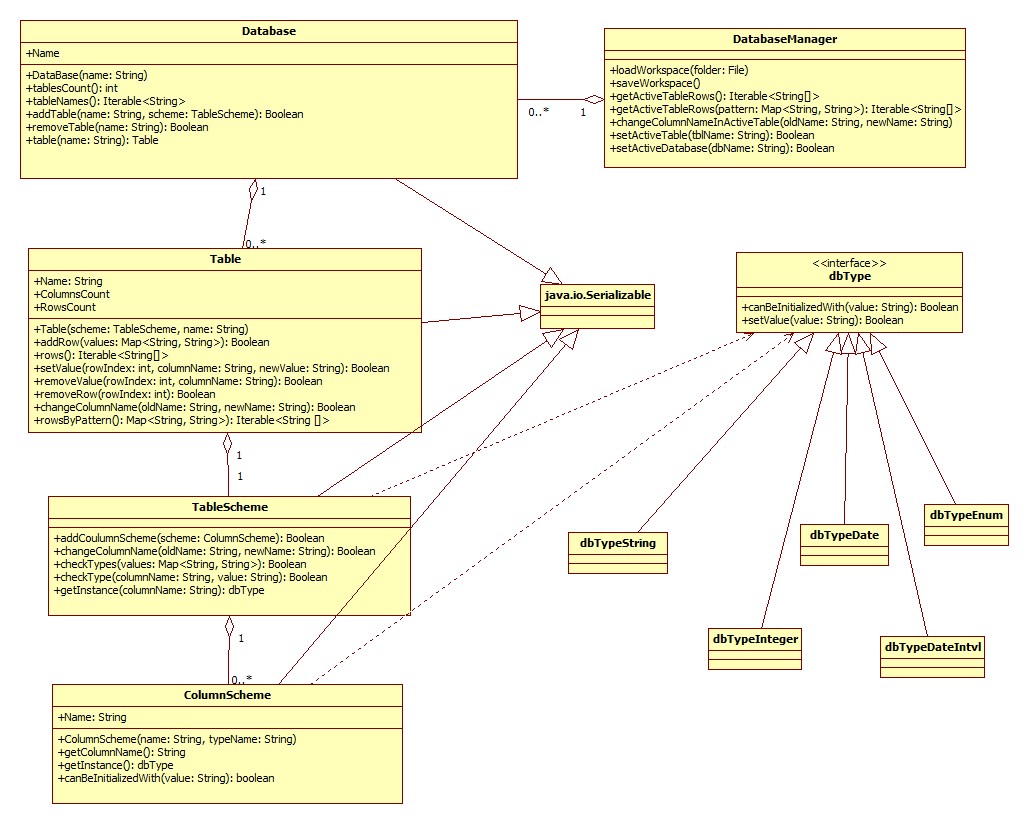
Тут основна увага звертається на такі операції програми як:

1) Перейменування колонки однієї з таблиць

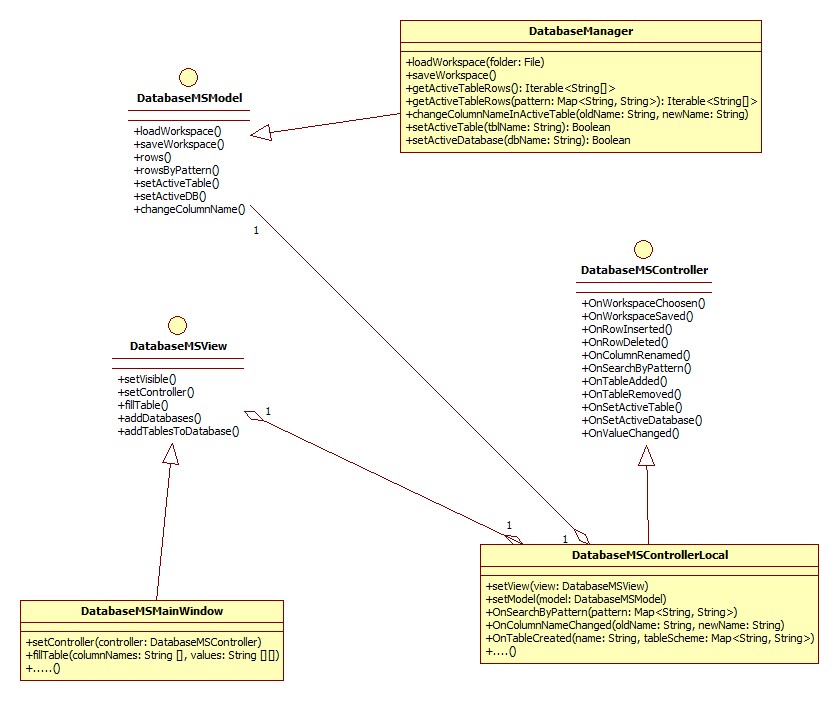
2) Пошук рядків таблиці, що задовольняють вимогам користувача (патерну, що задається користувачем)

**Інші діаграми**

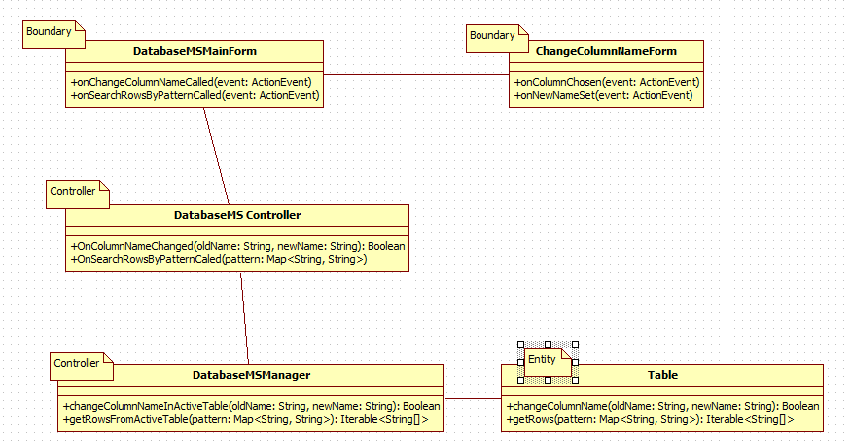
*1. Діаграма класів ядра (логіка бази даних)*

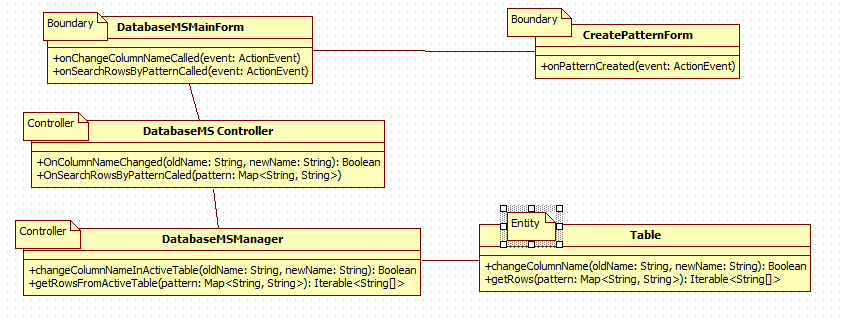


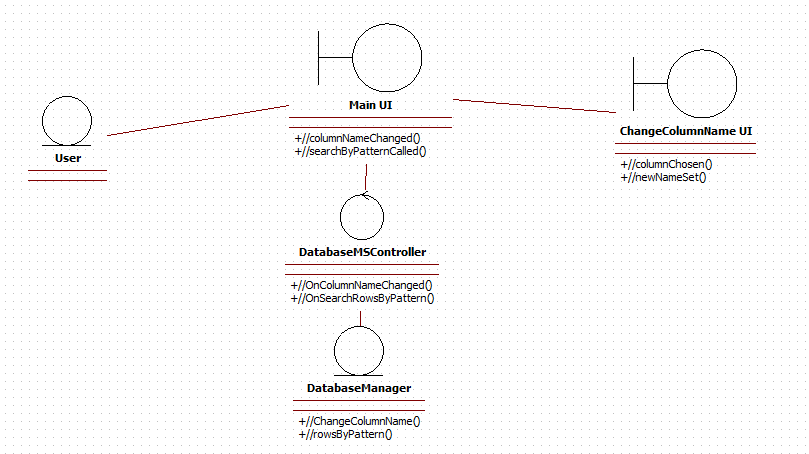
*2. Діаграма MVC*



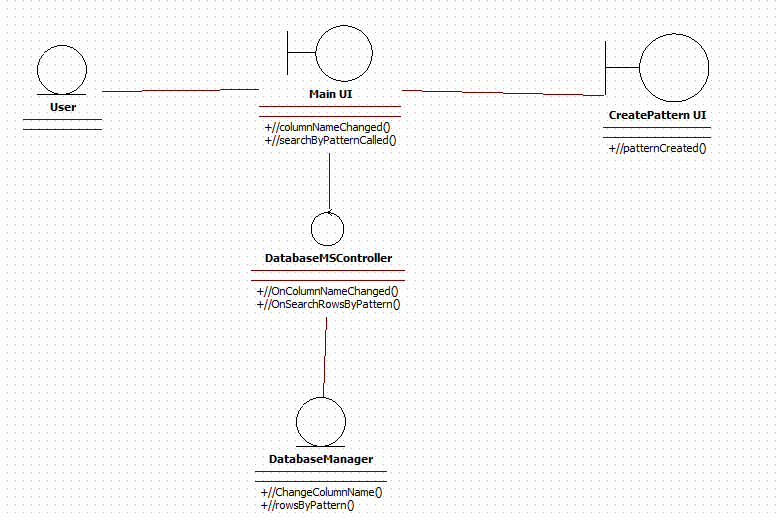
*3. VOPC діаграма для перейменування колонки однієї з таблиць*



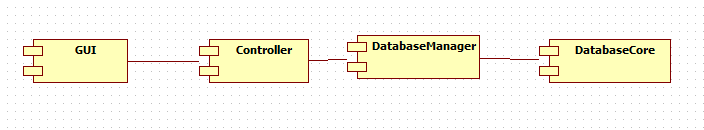
*4. VOPC діаграма для пошуку рядків за патерном*  
  
  
*5. Діаграма взаємодії 1*



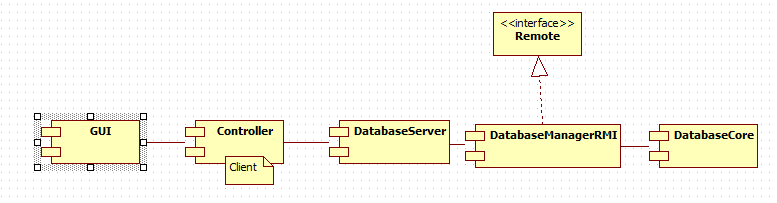
*6. Діаграма взаємодії 2*



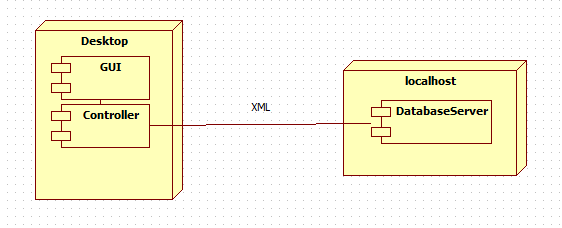
*7. Діаграма компонентів для локальної версії*



*8. Діаграма компонентів росподіленої версії*



*9. Діаграма розгортання*



**Локальна версія програми**

Локальна версія програми є базою для всіх інших версій. Архітектура локальної версії програми побудована на патерні проектування Модель - Вид - Контролер, де:

***Модель*** відповідає за логіку бази даних і складається з таких класів, як

*dbTypeString...dbTypeEnum* - класи, що представляють тип колонок       таблиці та надають такі конвертація до типу строкового значення

*ColumnScheme* - клас, що відповідає за назву колонки та її тип

*TableScheme* - клас, що відповідає за типи всіх колонок таблиці

*Table* - клас, що представляє таблицю

*Database* - клас, що представляє базу даних(набір таблиць)

*DatabaseManager* - клас, який зібрав в собі всі операції, які можна здійснити над моделлю (набір баз даних та операції над ними)

***Вид*** - це основна та допоміжні форми, які складають user interface програми

***Контролер*** - клас, що відповідає за сповіщення моделі про зміни, які були здійснені користувачем через інтерфейс та обновлення виду.

Шаблон MVC був використаний для того, щоб відділити дані від їхнього представлення користувачу. Оскільки цей шаблон передбачає, що дані та інтерфейс не взаємодіють безпосередньо один з одним, а тільки через контролер,то реалізація росподілених версій програми в подальшому потребувала мінімуму зусиль.

Далі подані лістинги коду, що відповідає за функції варіантів лабораторної роботи.

**Перейменування колонки**

*Вид*

**private** ActionListener renameColumnActionListener = **new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

**if**(tableView.getColumnCount() <= 0)

**return**;

String [] columnNames = **new** String[tableView.getColumnCount()];

**for**(**int** i = 0;i < columnNames.length; ++i)

columnNames[i] = tableView.getColumnName(i);

DatabaseMSChangeColumnNameWindow dialog = **new** DatabaseMSChangeColumnNameWindow(frmDbmanager, columnNames);

dialog.show(**new** DatabaseMSChangeColumnNameWindow.ColumnNameChangedListener() {

@Override

**public** **void** columnNameChanged(String oldName,String newName){

msController.OnColumnNameChanged(oldName, newName);

}

});

}

};

*Контролер*

**public** Boolean OnColumnNameChanged(String oldName, String newName) {

**if**(!dbManager.isActiveTableSet())

**return** **false**;

**if**(!dbManager.changeColumnNameInActiveTable(oldName, newName))

**return** **false**;

fillTable();

**return** **true**;

}

*Модель*

**public** Boolean changeColumnNameInActiveTable(String oldName, String newName) {

**return** activeTable().changeColumnName(oldName, newName);

}  
  
  
  
**Пошук рядків за патерном**  
  
*Вид*

**private** ActionListener patternSearchListener = **new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

**if**(tableView.getColumnCount() <= 0)

**return**;

String [] columnNames = **new** String[tableView.getColumnCount()];

**for**(**int** i = 0;i < columnNames.length; ++i)

columnNames[i] = tableView.getColumnName(i);

DatabaseMSPatternWindow dialog = **new** DatabaseMSPatternWindow(frmDbmanager, columnNames);

dialog.setController(msController);

dialog.show();

}

};

*Контролер*

**public** Boolean OnSearchByPattern(Map<String, String> pattern) {

**if**(pattern.size() == 0)

**return** **true**;

ArrayList<Integer> foundRows = **new** ArrayList<Integer>();

**for**(Integer i : dbManager.getActiveTableRows(pattern))

foundRows.add(i);

fillTable(foundRows);

**return** **true**;

}

*Модель*

**public** ArrayList<Integer> getActiveTableRows(Map<String, String> pattern) {

**return** activeTable().rows(pattern);

}

**Рефлексія**

Рефлексія при реалізації програми була використана наступним чином.

Для кожної колонки таблиці нам необхідно створювати об’єкти відповідного типу. Так от ці об’єкти створюються класом-фабрикою, що приймає назву класу об’єкта і повертає об’єкт необхідного типу. Патерн фабрика може бути реалізований на мові програмування Java за допомогою рефлексі. Лістинг коду:

Class<? **extends** dbType> type = **null**;

**try** {

type = (Class<? **extends** dbType>) Class.*forName*("DatabaseMSCore." + simpleTypeName);

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**try** {

**return** type.newInstance();

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

// **TODO** Auto-generated catch block

e.printStackTrace();

}  
  
  
  
  
  
  
  
  
  
**Розподілена версія програми: RMI**

Фактично, реалізації обох розподілених версій RMI\JRMP та RMI\IIOP є майже однаковими. Методи класу *DatabaseManager* були виділені в окремий інтерфейс віддаленої взаємодії *DatabaseSModelRMI* (наслідується від Remote).

**public** **interface** DatabaseMSModelRMI **extends** Remote{

......

**public** Iterable<String[]> getActiveTableRows() **throws** RemoteException;

**public** Iterable<Integer> getActiveTableRows(Map<String, String> pattern) **throws** RemoteException;

**public** Boolean changeColumnNameInActiveTable(String oldName, String newName) **throws** RemoteException;

.........

}

Клас *DatabaseMSManagerRMIImpl* наслідує інтерфейс віддаленої взаємодії, що дозволяє створити об’єкт цього класу на серверній стороні та викликати його методи на стороні клієнта. Тепер клас контролер не створює новий об’єкт модель, а користується об’єктом, що був розміщений на сервері.

Різниця між двома реалізаціями полягая в тому, як об’єкт-модель розміщується на сервері та як до нього доступається клієнт.

**RMI\JRMP**

*Серверний код:*

**try** {

DatabaseMSModelJRMPImpl model = **new** DatabaseMSModelJRMPImpl();

DatabaseMSModelRMI stub = (DatabaseMSModelRMI) UnicastRemoteObject.*exportObject*(model, 0);

// Bind the remote object's stub in the registry

Registry registry = LocateRegistry.*createRegistry*(1099);

registry.rebind("DatabaseMSModelJRMP", stub);

System.*err*.println("Server ready");

} **catch** (Exception e) {

System.*err*.println("Server exception: " + e.toString());

e.printStackTrace();

}

*Клієнтський код:*

**this**.dbManager = **null**;

**try** {

Registry registry = LocateRegistry.*getRegistry*();

dbManager = (DatabaseMSModelRMI) registry.lookup("DatabaseMSModelJRMP");

} **catch** (Exception e) {

System.*err*.println("Client exception: " + e.toString());

e.printStackTrace();

}

**this**.msView = msView;

**RMI\IIOP**

*Серверний код*

**try** {

// Step 1: Instantiate the Hello servant

DatabaseMSModelIIOPImpl model = **new** DatabaseMSModelIIOPImpl();

// Step 2: Publish the reference in the Naming Service

// using JNDI API

Context initialNamingContext = **new** InitialContext();

initialNamingContext.rebind("DatabaseMSModelIIOPImpl", model );

System.*out*.println("DatabaseMS Server: Ready...");

} **catch** (Exception e) {

e.printStackTrace();

}

*Клієнтський код*

**try** {

// Step 1: Instantiate the Hello servant

DatabaseMSModelIIOPImpl model = **new** DatabaseMSModelIIOPImpl();

// Step 2: Publish the reference in the Naming Service

// using JNDI API

Context initialNamingContext = **new** InitialContext();

initialNamingContext.rebind("DatabaseMSModelIIOPImpl", model );

System.*out*.println("DatabaseMS Server: Ready...");

} **catch** (Exception e) {

e.printStackTrace();

}

**Веб-сервіс**

Ще один варіант розподіленої версіі програми був реалізований на основі технології JAX-WS.Тут клас, що представляє модель наслідує service endpoint інтерфейс

@WebService

@SOAPBinding(style=Style.*DOCUMENT*)

**public** **interface** IDatabaseMSWebService {

...

**public** ArrayList<Integer> getActiveTableRowsByPattern(Map<String, String> pattern);

@WebMethod

**public** Boolean changeColumnNameInActiveTable(String oldName, String newName);

...

}

@WebService(endpointInterface = "DatabaseMSWebService.IDatabaseMSWebService")

@SOAPBinding(style=Style.*DOCUMENT*)

**public** **class** DatabaseMSWebServiceImpl **implements** IDatabaseMSWebService {

...

}

На стороні сервера створюється об’єкт модель, який розміщується на localhost, за допомогою класа Endpoint.

**public** **class** Publisher {

**public** **static** **void** main(String[] args) {

Endpoint.*publish*("http://localhost:8080/database\_ms", **new** DatabaseMSWebServiceImpl());

System.*out*.println("Ready!");

}

}

Клієнтський код отримує посилання на об’єкт, що знаходиться на сервері та передає його в контролер, що працює працює з об’єктом-моделлю, як і в локальній версії програми.

URL url = **null**;

**try** {

url = **new** URL("http://localhost:8080/database\_ms?wsdl");

} **catch** (MalformedURLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

QName name = **new** QName("http://DatabaseMSWebService/", "DatabaseMSWebServiceImplService");

Service service = Service.*create*(url, name);

IDatabaseMSWebService dbService = service.getPort(IDatabaseMSWebService.**class**);

DatabaseMSView dbView = **new** DatabaseMSMainWindow();

DatabaseMSWebServiceController dbController = **new** DatabaseMSWebServiceController(dbView, dbService);

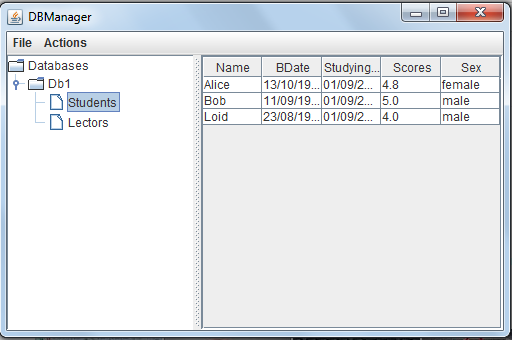
dbView.setController(dbController);

Вихідний код програмної реалізації є відкритим та доступний у git-репозиторії за адресою: <https://github.com/chernovoleh/DatabaseMS.git>

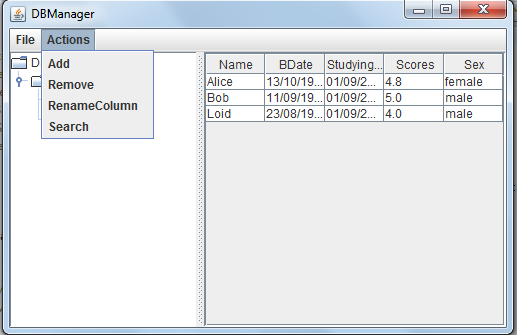
**User Guide**

Для початку роботи з програмою оберіть в меню пункт File->Open Workspace та  оберіть шлях, за  яким розташована ваша база даних. Зліва ви побачите список доступних до перегляду баз. Щоб побачити, які таблиці належать обраній базі даних - просто натисніть на її ім’я та список таблиць баз даних буде завантажений.

Для роботи з таблицею оберіть у лівій частині вікна її їім’я та натисніть.



Доступні операції з таблицею ви можете подивитися в меню Actions. Таким чином, користувач може додавати та видаляти рядки в таблиці, перейменовувати колонки, а також шукати потрібну йому інформацію.



Зупинимось на кожній дії окремо. Для того, аби додати до активної таблиці рядок,  оберіть в меню Actions-> Add. В таблиці буде додано новий порожній рядок, до якого ви можете занести всю необхідну інформацію. Зверніть увагу на корректність вводу ваших даних, адже програма валідує їх:

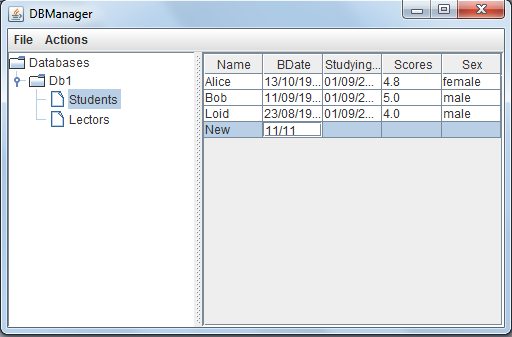
в поле типу dbString  можна ввести будь-яке слово, або їх комбінацію;

в поле типу dbInteger можна ввести будь-яке число;

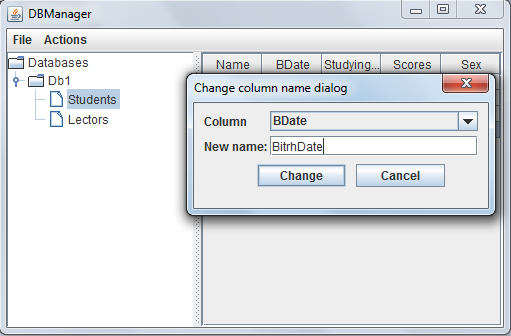
поле типу dbDate представлений в наступному форматі: mm/dd/yyyy;

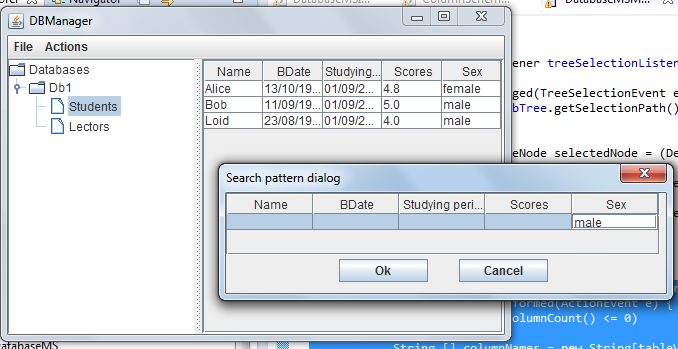
поле типу dbDateInterval користувач має задати наступним чином: mm/dd/yyyy-mm/dd/yyyy (поєднання двох дат, з'єднаних знаком дефіс);

в поле типу dbEnum користувач може ввести лише ті значення, що входять в множину даного    enumу. Наприклад, якщо enum "Колір" має значення зелений, червоний та синій, то значення "жовтий" користувачеві буде недоступне.



Наступна дія, доступна користувачеві, видалення рядку з таблиці. Для виконання цієї дії необхідно обрати рядок, натиснути на нього, а потім вибрати в меню Actions-> Remove.

Для того, аби перейменувати назву якоїсь колонки оберіть в меню Actions->RenameColumn. Після цього буде відкрите нове вікно, в якому користувач має змогу обрати поточне ім'я колонки та ввести нове ім'я у відповідне віконце.

Ще одна дія, доступна користувачеві-пошук інформації за шаблоном. Для цього необхідно вибрати в меню Actions -> Search. Після цього буде відкрито нове вікно,  в якому буде відкрито пустий рядок таблиці. Користувач може ввести в будь яке поле необхідне значення (або у кілька полей), таким чином задавши пошуковий паттерн. Далі, після натискання Ok  у вікні будуть відображені результати пошуку по таблиці (усі рядки таблиці, що влаштовують пошуковий запит). У випадку, якщо не знайдено жодного рядка, влаштовуючого пошуковий паттерн, вікно залишиться порожнім.

Слід зазначити, що операції додавання та видалення доступні не лише для рядка, а й для таблиці вцілому. Тобто, якщо на поточний час у вікні програми активною буде таблиця (саме її ім'я буде підсвічено синім кольором), то при виборі в меню пункту Add  до бази даних буде додана нова таблиця. Те саме справджується і для дії видалення.