**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА**

**ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА КОМП’ЮТЕРНИХ ТЕХНОЛОГІЙ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(повне найменування назва факультету)

**КАФЕДРА СИСТЕМНОГО ПРОЕКТУВАННЯ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(повна назва кафедри)

**Пояснювальна записка**

до дипломної (кваліфікаційної) роботи

**БАКАЛАВР**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(освітньо-кваліфікаційний рівень)

на тему

**«Створення додатків для роботи з СУБД Oracle за допомогою JAVA»**

Виконав: студент **ІV** курсу, групи **ФеІ-42**

напряму підготовки

6.050101 Комп’ютерні науки \_

(шифр і назва напряму підготовки)

**Шайнога O.В** \_

(прізвище та ініціали)

Керівник **Балицький О.О\_\_\_\_ \_\_** \_

(прізвище та ініціали)

Рецензент **Фургала Ю.М** \_

(прізвище та ініціали)

Львів - 2017

**Анотація**

У дипломній роботі було створено додатки для роботи з СУБД ORACLE за допомогою мови програмування Java SE8. Використання JavaFX та Swing для створення користувацького інтерфейсу. Шаблону проектування MVS для відокремлення розробки графічного інтерфейсу від розробки бізнес логіки. Використання JDBC інтерфейсу для доступу до бази даних ORACLE. У практичній частині роботі продемонстровано розробка програми з використанням всіх перелічених вище технологій.

**Summary**

In the thesis work was created application to work with ORACLE database using the programming language Java SE8. Using JavaFX and Swing user interface to create. MVS design pattern for separating GUI development from the development of business logic. Using the JDBC interface to access database ORACLE. In the practical part of the work demonstrated development programs using all of the above technologies.

ЗМІСТ

**ВСТУП**

Java – кросплатформна  [об'єктно-орієнтована](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)  мова програмування. Використовується для створення як мобільних додатків так і великих ентерпрайз проектів . Мова значно запозичила синтаксис із [C](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) і [C++](https://uk.wikipedia.org/wiki/C%2B%2B). Зокрема, взято за основу об'єктну модель С++, проте її модифіковано.Java надає корисні бібліотеи для створення багатопоточних програм та програм з віконим інтерфейсом.

В сучасному світі все більше і більше інформації. Щоб зберігати та аналізувати її використовують системи управління базами даних . СУБД — комплекс [програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F), що надає можливості створення, збереження, оновлення та пошуку інформації в [базах даних](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) з контролем доступу до даних.

Але для того щоб керувати даними у базі даних потрібно знати SQL- мову структурованих запитів. Навіть для найпростішої операції потрібно написати SQL запит. Через те що без допоміжних засобів з базою даних можна взаємодіяти тільки через командну стрічку такі прості задачі як перегляд даних або перегляд структури бази даних забирають багато часу.

Метою роботи є дослідження СУБД Oracle та платформи Java їх використання для створення додатків. Об’єктом дослідження є процес програмування. Предметом дослідження є взаємодія мови програмування Java з СУБД Oracle . Мова Java володіє потужною бібліотекою класів але найголовніше є те що Java має Java DataBase Connectivity прикладний програмний інтерфейс Java, який визначає методи, з допомогою яких програмне забезпечення на Java здійснює доступ до бази даних.

Також на основі досліджень платформи, було сформовано завдання реалізувати додаток для роботи з СУБД Oracle. Користувацький інтерфейс буде реалізовано за підтримки мови розмітки FXML та патерну проектування MVC.

1.**Мова програмування Java**

Java - [об'єктно-орієнтована](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [мова програмування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), випущена 1995 року компанією [«Sun Microsystems»](https://uk.wikipedia.org/wiki/Sun_Microsystems) як основний компонент платформи Java. З 2009 року мовою займається компанія [«Oracle»](https://uk.wikipedia.org/wiki/Oracle_Corporation), яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми [компілюються](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80) у [байт-код](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4), який при виконанні інтерпретується [віртуальною машиною](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_Java) для конкретної платформи.

«Oracle» надає компілятор Java та [віртуальну машину Java](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_Java), які задовольняють специфікації [Java Community Process](https://uk.wikipedia.org/wiki/Java_Community_Process), під ліцензією [GNU General Public License](https://uk.wikipedia.org/wiki/GNU_General_Public_License).

Мова значно запозичила синтаксис із [C](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) і [C++](https://uk.wikipedia.org/wiki/C%2B%2B). Зокрема, взято за основу об'єктну модель С++, проте її модифіковано. Усунуто можливість` появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в С/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java вплинула на розвиток [J++](https://uk.wikipedia.org/w/index.php?title=J%2B%2B&action=edit&redlink=1), що розроблялась компанією [«Microsoft»](https://uk.wikipedia.org/wiki/Microsoft). Роботу над J++ було зупинено через судовий позов «Sun Microsystems», оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі «Microsoft» [.NET](https://uk.wikipedia.org/wiki/.NET_Framework) випустили [J#](https://uk.wikipedia.org/wiki/Visual_J_Sharp), щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування [С#](https://uk.wikipedia.org/wiki/C_Sharp) стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію [Microsoft Visual Studio](https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio) 2005. Мова сценаріїв [JavaScript](https://uk.wikipedia.org/wiki/JavaScript) має схожу із Java назву і синтаксис, але не пов'язана із Java.

### 1.1 Платформа Java

### Під «незалежністю від архітектури» мається на увазі те, що програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у [початковому коді](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%87%D0%B0%D1%82%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4) та [перекомпіляції](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80).

Цього можна досягти, компілюючи початковий Java код у [байт-код](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4), який є спрощеними машинними командами. Потім програму можна виконати на будь-якій платформі, що має встановлену [віртуальну машину](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) Java, яка інтерпретує байткод у код, пристосований до специфіки конкретної [операційної системи](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) і [процесора](https://uk.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80). Зараз віртуальні машини Java існують для більшості процесорів і операційних систем.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка [графіки](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%BA%D0%B0), [багатопотоковість](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%BD%D0%B8%D1%82%D0%B5%D0%B2%D1%96%D1%81%D1%82%D1%8C) та роботу з [мережами](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0). У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми.

Основна перевага використання [байт-коду](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4) — це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної» мови. Проте, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Одним із таких методів є [just-in-time компіляція](https://uk.wikipedia.org/wiki/JIT-%D0%BA%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%86%D1%96%D1%8F) (JIT, що перетворює байт-код Java у машинний під час першого запуску програми, а потім кешує його. У результаті така програма запускається і виконується швидше, ніж простий інтерпретований код, але ціною додаткових витрат на компіляцію під час виконання. Складніші віртуальні машини також використовують динамічну рекомпіляцію, яка полягає в тому, що віртуальна машина аналізує поведінку запущеної програми й вибірково рекомпілює та оптимізує певні її частини. З використанням динамічної рекомпіляції можна досягти більшого рівня оптимізації, ніж за статичної компіляції, оскільки динамічний компілятор може робити оптимізації на базі знань про довкілля періоду виконання та про [завантажені](https://uk.wikipedia.org/wiki/Java_Classloader) класи. До того ж він може виявляти так звані гарячі точки ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) hot spots) — частини програми, найчастіше внутрішні цикли, які займають найбільше часу при виконанні. JIT-компіляція та динамічна рекомпіляція збільшує швидкість Java-програм, не втрачаючи при цьому портативності.

Існує ще одна технологія оптимізації байткоду, широко відома як статична компіляція, або компіляція ahead-of-time (AOT). Цей метод передбачає, як і традиційні компілятори, безпосередню компіляцію у машинний код. Це забезпечує хороші показники в порівнянні з інтерпретацією, але за рахунок втрати переносності: скомпільовану таким способом програму можна запустити тільки на одній, цільовій платформі.

Швидкість офіційної віртуальної машини Java значно покращилася з моменту випуску ранніх версій, до того ж, деякі випробування показали, що продуктивність JIT-компіляторів у порівнянні зі звичайними компіляторами у машинний код майже однакова. Проте ефективність компіляторів не завжди свідчить про швидкість виконання скомпільованого коду, тільки ретельне тестування може виявити справжню ефективність у даній системі.

### 1.2 Об'єктність

На противагу C++, Java об'єктно-орієнтованіша. Всі дані і дії групуються в класи [об'єктів](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Виключенням з повної об'єктності (як скажімо в [Smalltalk](https://uk.wikipedia.org/wiki/Smalltalk)) є примітивні типи (int, float тощо). Це було свідомим рішенням проектувальників мови задля збільшення швидкості. Через це Java не вважається повністю об'єктно-орієнтовною мовою.

У Java всі об'єкти є похідними від головного об'єкта (він називається просто Object), з якого вони успадковують базову поведінку і властивості.

Хоча у [C++](https://uk.wikipedia.org/wiki/C%2B%2B) вперше стало доступне множинне [успадкування](https://uk.wikipedia.org/wiki/%D0%A3%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), але у Java можливе тільки одинарне успадкування, завдяки чому виключається можливість конфліктів між членами класу ([методи](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) і [змінні](https://uk.wikipedia.org/wiki/%D0%97%D0%BC%D1%96%D0%BD%D0%BD%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))), які успадковуються від базових класів.

### 1.3 Безпека

У намірах проектувальників Java мала замінити [C++](https://uk.wikipedia.org/wiki/C%2B%2B) — об'єктного наступника мови [C](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Проектувальники почали з аналізу властивостей C++, які є причиною найбільшого числа помилок, щоби створити просту, безпечну і безвідмовну мову програмування.

В Java існує система винятків або ситуацій, коли програма зустрічається з неочікуваними труднощами, наприклад:

* операції над елементом масиву поза його межами або над порожнім елементом
* читання з недоступного каталогу або неправильної адреси URL
* ввід недопустимих даних користувачем

Одна з особливостей концепції [віртуальної машини](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) полягає в тому, що помилки (виключення) не призводять до повного краху системи. Крім того, існують інструменти, які «приєднуються» до середовища періоду виконання і кожен раз, коли сталося певне виключення, записують інформацію з пам'яті для відлагодження програми. Ці інструменти [автоматизованої обробки виключень](https://uk.wikipedia.org/w/index.php?title=%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%BD%D0%B0_%D0%BE%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%B2%D0%B8%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D1%8C&action=edit&redlink=1) надають основну інформацію щодо виключень в програмах на Java.

Проте мову програмування Java не рекомендується використовувати в системах, збій в роботі яких може призвести до смерті, травм чи значних фізичних ушкоджень (наприклад, програмне забезпечення для керування атомними електростанціями, польотами, систем життєзабезпечення чи систем озброєння) через ненадійність програм, написаних на мові програмування Java.

### 1.4 Автоматичне керування пам'яттю

Java використовує автоматичний збирач сміття для керування пам'яттю під час життєвого циклу об'єкта. Програміст вирішує, коли створювати об'єкти, а віртуальна машина відповідальна за звільнення пам'яті після того, як об'єкт стає непотрібним. Коли до певного об'єкта вже не залишається посилань, збирач сміття може автоматично прибирати його із пам'яті. Проте, витік пам'яті все ж може статися, якщо код, написаний програмістом, має посилання на вже непотрібні об'єкти, наприклад на об'єкти, що зберігаються у діючих контейнерах.

Збирання сміття дозволене у будь-який час. В ідеалі воно відбувається під час бездіяльності програми. Збірка сміття автоматично форсується при нестачі вільної пам'яті в купі для розміщення нового об'єкта, що може призводити до кількасекундного зависання. Тому існують реалізації віртуальної машини Java з прибиральником сміття, спеціально створеним для програмування [систем реального часу](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D1%87%D0%B0%D1%81%D1%83).

Java не має підтримки вказівників у стилі C/C++. Це зроблено задля безпеки й надійності, аби дозволити збирачу сміття переміщувати вказівникові об'єкти.

1.5 Структура джава програми

Програми на Java утворені з визначень [класів](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) та [інтерфейсів](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81). Класи містять [змінні](https://uk.wikipedia.org/wiki/%D0%97%D0%BC%D1%96%D0%BD%D0%BD%D0%B0) та [константи](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D0%B0), які утримують дані, [методи](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4), які виконують дії, та [конструктори](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), які створюють екземпляри класів — [об'єкти](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Дані можуть мати простий [тип](https://uk.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) (наприклад [байт](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82), [ціле число](https://uk.wikipedia.org/wiki/%D0%A6%D1%96%D0%BB%D1%96_%D1%87%D0%B8%D1%81%D0%BB%D0%B0), [символ](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB)) або бути [посиланням](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D0%B8%D0%BB%D0%B0%D0%BD%D0%BD%D1%8F) на об'єкт. Мова Java є [статично типізованою](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%97).

### Лексична структура

Java-програми записуються в [Юнікоді](https://uk.wikipedia.org/wiki/%D0%AE%D0%BD%D1%96%D0%BA%D0%BE%D0%B4), також надається лексичне перетворення, яке дозволяє записувати символи Юнікоду керівними кодами Unicode за допомогою лише множини символів [ASCII](https://uk.wikipedia.org/wiki/ASCII). Мова Java представляє текст послідовностями 16-бітних кодових одиниць, використовуючи кодування [UTF-16](https://uk.wikipedia.org/wiki/UTF-16). За винятком коментарів, ідентифікаторів та вмісту символьних та рядкових літералів, всі вхідні елементи програми на Java складаються із символів ASCII або відповідних їм керівних кодів Unicode.

### 1.6 Типи даних

Java є [суворо типізованою](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%97) мовою, кожна змінна та вираз має [тип](https://uk.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), відомий на етапі [компіляції](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80).

Типи даних Java належать до двох категорій: прості (primitive) та [вказівникові](https://uk.wikipedia.org/wiki/%D0%92%D0%BA%D0%B0%D0%B7%D1%96%D0%B2%D0%BD%D0%B8%D0%BA) (reference). До простих типів належить булевий(логічний) тип, числові типи та символьний тип.

Числові типи складаються із цілих типів byte, short, int, long та дійсних типів float, double. Символьний тип представлений типом char. Вказівникові типи складаються із класів, інтерфейсів, масивів. Значенням вказівникового типу є вказівник на об'єкт — екземпляр класу чи [масиву](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D0%B8%D0%B2_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)). [Рядки](https://uk.wikipedia.org/wiki/%D0%A0%D1%8F%D0%B4%D0%BE%D0%BA_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Рядок (програмування)) є об'єктами класу String.

|  |  |  |
| --- | --- | --- |
| **Тип** | **Довжина (в байтах)** | **Діапазон або набір значень** |
| boolean | не визначено | true, false |
| byte | 1 | −128..127 |
| char | 2 | 0..216−1, або 0..65535 |
| short | 2 | −215..215−1, або −32768..32767 |
| int | 4 | −231..231−1, або −2147483648..2147483647 |
| long | 8 | −263..263−1, або приблизно −9.2·1018..9.2·1018 |
| float | 4 | -(2-2−23)·2127..(2-2−23)·2127, або приблизно −3.4·1038..3.4·1038, а також -∞{\displaystyle -\infty }, ∞{\displaystyle \infty }, NaN |
| double | 8 | -(2-2−52)·21023..(2-2−52)·21023, або приблизно −1.8·10308..1.8·10308, а також -∞{\displaystyle -\infty }, ∞{\displaystyle \infty }, NaN |

1.7 JavaFX

JavaFX — платформа та набір інструментів для створення [насичених інтернет-застосунків](https://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) Rich Internet Applications, RIA) з можливістю підвантаження медіа та змісту. Вперше продемонстровано [Sun Microsystems](https://uk.wikipedia.org/wiki/Sun_Microsystems) на Міжнародній конференції [Java](https://uk.wikipedia.org/wiki/Java)-розробників JavaOne у травні [2007](https://uk.wikipedia.org/wiki/2007). JavaFX включає в себе набір утиліт, за допомогою яких веб-розробники та дизайнери можуть швидко створювати та надавати розвинуті інтернет-[застосунки](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BA%D0%B8) для десктопів, мобільних пристроїв, телебачення та інших платформ.

JavaFX складається з JavaFX Script і JavaFX Mobile. Починаючи з випуску JavaFX 2.0 забезпечено можливість створення JavaFX-застосунків, написаних цілком мовою [Java](https://uk.wikipedia.org/wiki/Java). Для розробки [застосунків](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) доступний багатий графічний і мультимедійний [API](https://uk.wikipedia.org/wiki/API), що спрощує створення візуальних програм.

Нижче наведені деякі з найбільш важливих особливостей JavaFX -

• FXML - JavaFX є мова, відома як FXML, декларативна мову розмітки. Єдина мета цієї мови для визначення призначеного для користувача інтерфейсу.

• Scene Builder - JavaFX надає додаток під назвою Scene Builder. Інтегруючи це додаток в середовищі IDE, такі як Eclipse, і IntelijIdea, користувачі можуть отримати доступ до дизайну інтерфейсу, який використовується для розробки додатків FXML (так само, як свінг Drag & Drop і DreamWeaver додатків).

• Вбудовані елементи керування для користувача інтерфейсу - бібліотека JavaFX обслуговує елементи управління користувальницького інтерфейсу, за допомогою якого ми можемо розробити повнофункціональні додатки.

• CSS, як Styling - JavaFX забезпечує CSS як стиль. За допомогою цього, ви можете поліпшити дизайн вашого застосування за допомогою простого знання CSS.

• Багатий набір API, - бібліотека JavaFX забезпечує багатий набір API, для розробки додатків з графічним інтерфейсом, 2D і 3D графіки. Цей набір API також включає можливості платформи Java. Тому, використовуючи цей API, ви можете отримати доступ до функції Java мов, такі як дженеріки, анотації, багатопоточність і лямбда-вирази.

• Інтегрована графічна бібліотека - JavaFX надає класи для 2D і 3D графіки.

• Графічний конвеєр - JavaFX підтримує графіку на основі апаратного прискорення графічного конвеєра, відомий як Prism. При використанні з підтримуваної графічною картою або GPU пропонує гладкі графіки.

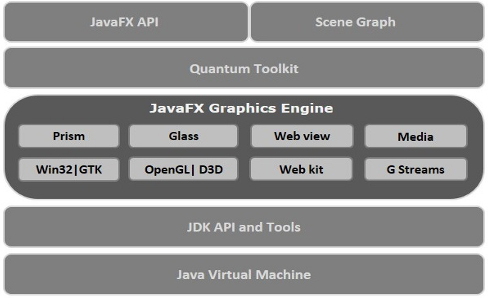
• На наведеному нижче малюнку показана архітектура JavaFX API. Тут ви можете побачити компоненти, які підтримують JavaFX API. 

Рисунок 1. - архітектура JavaFX API

1.8 Java DataBase Connectivity

JDBC — [прикладний програмний інтерфейс](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) [Java](https://uk.wikipedia.org/wiki/Java), який визначає методи, з допомогою яких програмне забезпечення на Java здійснює доступ до [бази даних](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85). JDBC — це платформо-незалежний промисловий стандарт взаємодії Java-[застосунків](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) з різноманітними [СУБД](https://uk.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94), реалізований у вигляді пакета java.sql, що входить до складу [Java SE](https://uk.wikipedia.org/w/index.php?title=Java_SE&action=edit&redlink=1).

В основі JDBC лежить концепція так званих драйверів, що дозволяють отримувати з'єднання з базою даних по спеціально описаному [URL](https://uk.wikipedia.org/wiki/URL). Драйвери можуть завантажуватись динамічно (під час роботи програми). Завантажившись, драйвер сам реєструє себе й викликається автоматично, коли програма вимагає [URL](https://uk.wikipedia.org/wiki/URL), що містить протокол, за який драйвер «відповідає».

Інтерфейси

JDBC [API](https://uk.wikipedia.org/wiki/API) містить два основні типи інтерфейсів: перший — для розробників [застосунків](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) і другий (нижчого рівня) — для розробників драйверів.

З'єднання з базою даних описується класом, що реалізує інтерфейс java.sql.Connection.

Маючи з'єднання з базою даних, можна створювати об'єкти типу Statement, використовувані для здійснення запитів до бази даних на мові [SQL](https://uk.wikipedia.org/wiki/SQL).

Існують такі види типів Statement, що відрізняються своїм призначенням:

* java.sql.Statement — Statement загального призначення;
* java.sql.PreparedStatement — Statement, що служить для здійснення запитів, котрі містять підставні параметри (позначаються символом '?' у тілі запиту);
* java.sql.CallableStatement — Statement, призначений для виклику збережених процедур.

Клас java.sql.ResultSet дозволяє легко обробляти результати запитів.

Переваги

Перевагами JDBC вважаються:

* Легкість розробки: розробник може не знати специфіки бази даних, з якою працює;
* Код не змінюється, якщо компанія переходить на іншу базу даних;
* Не треба встановлювати громіздку клієнтську програму;
* До будь-якої бази можна під'єднатись через легко описуваний [URL](https://uk.wikipedia.org/wiki/URL).

**2. ООП**

В основі об'єктно-орієнтованої мови програмування лежать два основних поняття: клас та об'єкт. Об'єкт – це базове поняття в ООП, це конкретна реалізація, екземпляр класу. Об'єкт складається з трьох частин: стан (змінні стану), методи (операції), ім'я об'єкта. Клас - це група даних і методів або функцій для роботи з цими даними, це шаблон. Об'єкти однаковими наборами змінних стану і методів, утворюють клас. Якщо об'єкти мають реалізацію з конкретного світу, то класи є абстракціями. Трохи більш складні об'єкти можуть взагалі не містити даних, а представляти процес і містити тільки функції, які реалізують цей процес. Для формування реального об'єкта необхідно мати шаблон, по прикладу якого і будується даний об'єкт.

Поняттю "об'єкт" зіставляють ряд доповнюючих один одного визначень:

· це сукупність змінних стану і пов'язаних з ними методів (операцій). Ці методи визначають як об'єкт взаємодіє з оточуючим його світом;

· це конкретна реалізація, екземпляр класу. У програмуванні відносини об'єкта і класу можна порівняти з описом змінної, де сама змінна (об'єкт) є екземпляром якого-небудь типу даних (класу);

· це відчутна реальність, що характеризується чітко визначеною поведінкою;

· особливий упізнаваний предмет, блок або сутність (реальний чи абстрактний), що має важливе функціональне призначення в даній предметній області.

Є й інша класифікація методів об'єкта, коли виділяють функції управління, реалізації, доступу та допоміжні функції. Індивідуальність об'єкта - це властивість об'єкта, що відрізняє цей об'єкт від усіх інших об'єктів. Об'єкти можуть перебувати в певних відносинах один до одного, відносини можуть бути ієрархічними. Основні ієрархічні відносини називають відносини використання та включення.

Основні властивості об'єктів:

Зазвичай вважається, що без інкапсуляції неможливо уявити собі об’єктно-орієнтоване програмування. Інкапсуляція це здатність об'єктів скривати деякі способи обробки даних (методи) та самі дані від навколишнього цифрового середовища. Історія розвитку методологій програмування спонукувана боротьбою зі складністю розробки програмного забезпечення. Складність великих програмних систем, у створенні яких бере участь відразу велика кількість розробників, зменшується, якщо на верхньому рівні не видно деталей реалізації нижніх рівнів. Власне, процедурний підхід був першим кроком на цьому шляху. Під інкапсуляцією (encapsulation, що можна перекласти по-різному, але на потрібні асоціації добре наводить слово "обволікання") розуміється приховування інформації про внутрішній устрій об'єкта, при якому робота з об'єктом може вестися тільки через його загальнодоступний (public) інтерфейс. Або це механізм програмування, об'єднуючий разом код і дані, якими він маніпулює, виключаючи як втручання ззовні, так і неправильне використання даних. В об'єктно-орієнтованій мові програмівання дані і код можуть бути об'єднані в абсолютно автономний чорний ящик. Усередині такого ящика знаходяться всі необхідні дані і код. Коли код і дані зв'язуються разом подібним чином, створюється об'єкт. Іншими словами, об'єкт - це елемент, що підтримує інкапсуляцію.

Поліморфізм - також одна з трьох основних парадигм ООП. Якщо говорити коротко, поліморфізм - це здатність об'єкта використовувати методи похідного класу, який не існує на момент створення базового. Слово «поліморфізм» можна перекласти як «багато форм». Тобто це можливість використання одного і того ж імені операції або методу до об'єктів різних класів, при цьому дії, що здійснюються з об'єктами, можуть істотно різнитися. Тому можна сказати, що в одного і того ж слова багато форм.

І остання парадигма спадкування (inheritance) - стосується здатності мови дозволяти будувати нові визначення класів на основі визначень існуючих класів. По суті, спадкування дозволяє розширювати поведінку базового класу, наслідуючи основну функціональність у похідному підкласі.Спадкування являє собою процес, в ході якого один об'єкт набуває властивостей іншого об'єкта. Це дуже важливий процес, оскільки він забезпечує принцип ієрархічної класифікації. Якщо вдуматися, то велика частина знань піддається систематизації завдяки ієрархічної класифікації по низхідній.

**3. MVC**

Враховуючи мету зменшення трудовитрат на розробку складного програмного забезпечення, припустимо, що необхідно використовувати готові уніфіковані рішення. Адже шаблонність дій полегшує комунікацію між розробниками, дозволяє посилатися на відомі конструкції, знижує кількість помилок.   
За словами [Вікіпедії](http://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), патерн (англ. design pattern) — повторимость архітектурна конструкція, що представляє собою рішення проблеми проектування в рамках деякого часто виникає контексту.    
Почнемо з першого головного — Model-View-Controller. MVC — це фундаментальний патерн, який знайшов застосування в багатьох технологіях, дав розвиток нових технологій і кожен день полегшує життя розробникам.    
Вперше патерн MVC з'явився в мові SmallTalk. Розробники повинні були придумати архітектурне рішення, яке дозволяло б відокремити графічний інтерфейс від бізнес логіки, а бізнес логіку від даних. Таким чином, в класичному варіанті, MVC складається з трьох частин, які й дали йому назву. Розглянемо їх:

###### Модель

Під Моделлю, зазвичай розуміється частина містить в собі функціональну бізнес-логіку програми. Модель повинна бути повністю незалежна від інших частин продукту. Модельний шар нічого не повинен знати про елементи дизайну, і яким чином він буде відображатися. Досягається результат, що дозволяє міняти подання даних, то як вони відображаються, не чіпаючи саму Модель.    
Модель володіє наступними ознаками:

* Модель — це бізнес-логіка додатка;
* Модель володіє знаннями про себе самій і не знає про контролери і уявленнях;
* Для деяких проектів модель — це просто шар даних (DAO, база даних, XML-файл);
* Для інших проектів модель — це менеджер бази даних, набір об'єктів або просто логіка додатка;

###### Представлення (View)

В обов'язки Уявлення входить відображення даних отриманих від Моделі. Однак, подання не може безпосередньо впливати на модель. Можна говорити, що подання має доступом «тільки на читання» до даних.    
Подання володіє наступними ознаками:

* У поданні реалізується відображення даних, які виходять від моделі будь-яким способом;
* У деяких випадках, подання може мати код, який реалізує деяку бізнес-логіку.

Приклади вистави: HTML-сторінка, WPF форма, Windows Form.

Основна ідея цього патерну в тому, що і контролер і уявлення залежать від моделі, але модель ніяк не залежить від цих двох компонент.    
 Ознаки контролера

* Контролер визначає, які уявлення має бути відображено в даний момент;
* Події уявлення можуть вплинути тільки на контроллер.контроллер може вплинути на модель і визначити інше уявлення.
* Можливо кілька подань тільки для одного контролера;   
   Реалізація:   
  Контролер перехоплює подія ззовні і відповідно до закладеної в нього логікою, реагує на цю подію змінюючи Mодель, за допомогою виклику відповідного методу. Після зміни Модель використовує подія про те що вона змінилася, і всі підписані на це події Уявлення, отримавши його, звертаються до Моделі за оновленими даними, після чого їх і відображають.

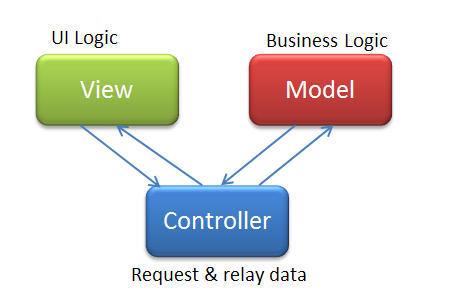
 

Рисунок 2.- Схема шаблону MVC

**4 Maven**

Maven **-**  це інструмент для збирання Java проекту: компіляції , створення jar, створення дистрибутиву програми , генерації документації . Прості проекти можна зібрати в командному рядку. Якщо збирати великі проекти з командного рядка , то команда для складання буде дуже довгою , тому її іноді записують у bat / sh скрипт . Але такі скрипти залежать від платформи. Для того щоб позбутися цієї залежності і спростити написання скрипта використовують інструменти для збірки проекту .  
Для платформи Java існують два основні інструменти для збирання: Ant і Maven .

Основні переваги Maven:  
    1) Незалежність від OS. Збірка проекту відбувається в будь-якій операційній системі . Файл проекту один і той же.  
    2) Управління залежностями . Рідко які проекти пишуться без використання сторонніх бібліотек ( залежностей ) . Ці сторонні бібліотеки часто теж в свою чергу використовують бібліотеки різних версій. Maven дозволяє управляти такими складними залежностями . Що дозволяє вирішувати конфлікти версій і в разі потреби легко переходити на нові версії бібліотек .  
    3) Можливість збирання з командного рядка. Таке часто необхідно для автоматичного складання проекту на сервері ( Continuous Integration ) .  
   4) Хороша інтеграція з середовищами розробки. Основні середовища розробки на java легко відкривають проекти, які збираються з допомогою maven . При цьому найчастіше проект налаштовувати не потрібно - він відразу готовий до подальшої розробки .  
    5) Як наслідок - якщо з проектом працюють в різних середовищах розробки, то maven зручний спосіб зберігання налаштувань . Настроювальний файл середовища розробки і для складання один і той же - менше дублювання даних і відповідно помилок.  
    6) Декларативний опис проекту .

INTELIJIDEA

http://o7planning.org/en/10167/java-jdbc-tutorial#a12645

http://docs.oracle.com/cd/E17781\_01/appdev.112/e18147/tdddg\_intro.htm#TDDDG12000

SCENBYILDER- https://docs.oracle.com/javase/8/scene-builder-2/installation-guide/jfxsb-installation\_2\_0.htm

**Oracle daabase**

Про реляційних базах даних

Кожна організація має інформацію, яку вона повинна зберігати і управляти ниею для задоволення своїх потреб. Наприклад, корпорація повинна збирати і вести облік людських ресурсів для своїх співробітників.

Інформаційна система являє собою формальну систему для зберігання і обробки інформації. Інформаційна система може представляти собою набір картонних коробок, що містять папки манільської разом з правилами, як зберігати та видавати папки. Тим не менше, більшість компаній сьогодні використовують бази даних для автоматизації інформаційних систем. База даних являє собою організований збір інформації розглядається як єдине ціле. Метою бази даних є для збору, зберігання та вилучення інформації, пов'язаної для використання додатків баз даних.

Система управління базами даних (СУБД)

Система управління базами даних (СУБД) є програмне забезпечення, яке управляє зберігання, організації та вилучення даних.

Як правило, СУБД має наступні елементи:

• Код ядра

Цей код управляє пам'яттю і для зберігання СУБД.

• Сховище метаданих

Це сховище зазвичай називається словником даних.

• Мова запитів

Ця мова дозволяє додаткам отримувати доступ до даних.

Додаток бази даних являє собою програмне забезпечення, яке взаємодіє з базою даних для доступу та управління даними.

реляційна модель

У своєму основоположному статті 1970 « A Relational Model of Data for Large Shared Data Banks» Е. Ф. Кодда визначив реляційну модель, засновану на математичній теорії множин. Реляційна база даних являє собою базу даних, яка відповідає реляційної моделі. Реляційна модель має наступні основні аспекти:

• структури

Чіткі об'єкти магазин або отримати доступ до даних в базі даних.

• Операції

Чітко визначені дії дозволяють додаткам обробляти дані і структуру бази даних.

• правила цілісності

правила цілісності регулюють операції над даними і структурами даних.

Реляційна база даних зберігає дані в наборі простих відносин. Ставлення являє собою набір кортежів. Кортеж є невпорядкований набір значень атрибутів.

Таблиця являє собою співвідношення у вигляді рядків (кортежів) і стовпців (атрибутів). Кожен рядок в таблиці має один і той же набір стовпців. Реляційна база даних являє собою базу даних, яка зберігає дані в стосунках (таблиці). Наприклад, реляційна база даних може зберігати інформацію про співробітників компанії в таблиці службовців, таблиці відділу і таблиці заробітної плати.

Система управління базами даних (СУБД)

Реляційна модель є основою для реляційної системи управління базами даних (СУБД). СУБД копіює дані в базу даних, зберігають дані, і витягає їх, так що додатки могли маніпулювати.

RDBMS розрізняє такі види операцій:

• Логічні операції

У цьому випадку додаток визначає, що потрібно зміст. Наприклад, програма запитує ім'я співробітника або додає запис співробітника до столу.

• Фізичні операції

В цьому випадку СУБД визначає, як все повинно бути зроблено, і виконує операцію. Наприклад, після того, як додаток запитує таблицю, база даних може використовувати індекс для пошуку необхідних рядків, читати дані в пам'ять, а також виконувати безліч інших кроків, перш ніж повертати результат користувачеві.

Oracle Database є СУБД. СУБД, яка реалізує об'єктно-орієнтовані функції, такі як визначені користувачем типи, успадкування і поліморфізм називається об'єктно-реляційна система управління базами даних (ОРСУБД). Oracle Database розширила реляційну модель для об'єктно-реляційної моделі, що дозволяє зберігати складні бізнес-модель в реляційної базі даних.

Structured Query Language (SQL)

SQL є декларативна мова, який надає інтерфейс до СУБД. На відміну від процедурних мов, таких як C, які описують, як все повинно бути зроблено, SQL є непроцедурного і описує те, що повинно бути зроблено.

SQL є стандартом мови ANSI для реляційних баз даних. Всі операції з даними в базі даних Oracle виконуються за допомогою SQL-операторів. Наприклад, можна використовувати SQL для створення таблиць і запитів і зміни даних в таблицях.

Оператор SQL можна розглядати як дуже просту, але потужну, комп'ютерну програму або інструкцію.

SQL дозволяє виконувати наступні завдання:

• Запит даних

• Вставка, оновлення та видалення рядків в таблиці

• Створювати, замінювати, змінювати і видаляти об'єкти

• Контроль доступу до бази даних і її об'єктів

• узгодженість і цілісність бази даних

SQL об'єднує попередні завдання в одному послідовному мовою. Oracle SQL є реалізацією стандарту ANSI. Oracle SQL підтримує велику кількість функцій, які виходять за рамки стандартного SQL.Transaction

Transactions

Під транзакцією розуміється неподільна з точки зору впливу на БД послідовність операторів маніпулювання даними (читання, видалення, вставки, модифікації) така, що або результати всіх операторів, які входять в транзакцію, відображаються в БД, або вплив усіх цих операторів повністю відсутній.   
Операція вважається транзакцією, якщо вона задовольняє вимогам ACID-тесту (Atomicity - атомарність, Consistency - узгодженість, Isolation - ізольованість, Durability - довговічність):

* атомарність - гасло транзакції – «Все або нічого»: при завершенні транзакції оператором COMMIT результати гарантовано фіксуються у зовнішній пам'яті (сенс слова commit – «зафіксувати» результати транзакції); при завершенні транзакції оператором ROLLBACK результати гарантовано відсутні в зовнішній пам'яті (сенс слова rollback - ліквідувати результати транзакції);
* узгодженість - по закінченню транзакції цілісність повинна бути встановлена, але в ході виконання транзакції цілісність може тимчасово порушуватися;
* ізольованість означає, що транзакції, які паралельно виконуються, не заважають одна одній;
* довговічність означає, що всі підтверджені в ході виконання зміни повинні бути внесені в базу даних навіть у разі збоїв системи.

Підтримання механізму транзакцій - показник рівня розвиненості СКБД. Коректний механізм підтримки транзакцій одночасно є основою забезпечення цілісності баз даних і також є базисом ізольованості користувачів в багатокористувацьких системах. Часто ці два аспекти розглядаються окремо, але насправді вони взаємопов'язані, що і буде показано нижче.

Зауважимо, що хоча з точки зору забезпечення цілісності баз даних механізм транзакцій слід було б підтримувати в персональних СКБД, на практиці це зазвичай не виконується. Тому при переході від персональних до багатокористувацьких СКБД користувачі стикаються з необхідністю чіткого розуміння природи транзакцій.

##### **Транзакції та цілісність баз даних**

Поняття транзакції має безпосередній зв'язок з поняттям цілісності БД. Дуже часто БД може мати такі обмеження цілісності, які просто неможливо не порушити, виконуючи лише один оператор зміни БД. Наприклад, нехай в базі даних є дві таблиці ПРАЦІВНИКИ і ВІДДІЛИ. Природним обмеженням цілісності є збіг значення атрибуту ВІДДІЛ\_РОЗМІР в рядку таблиці ВІДДІЛИ, що описує даний відділ, з числом рядків таблиці СПІВРОБІТНИКИ даного відділу. Як в цьому випадку прийняти на роботу у відділ нового співробітника? Незалежно від того, яка операція буде виконана першою (вставка нового рядку в таблицю СПІВРОБІТНИКИ або модифікація існуючого рядку таблиці ВІДДІЛИ) після виконання операції база даних залишиться в нецілісному стані.

Тому для підтримки подібних обмежень цілісності допускається їх порушення в транзакції з тією умовою, щоб до моменту завершення транзакції умови цілісності були дотримані. У системах з розвиненими засобами обмеження та контролю цілісності кожна транзакція починається при цілісному стані БД і повинна залишити цей стан цілісними після свого завершення. Недотримання цієї умови призводить до того, що замість фіксації результатів транзакції відбувається її відкат (тобто замість оператора COMMIT виконується оператор ROLLBACK), і БД залишається в такому стані, в якому перебувала до моменту початку транзакції, тобто в цілісному стані.

**Цілісність даних** – узгодженість даних в БД. **Обмежуюча умова** – це правило, яке обмежує значення даних в БД. В реляційній моделі є декілька обмежуючих умов, що використовуються для перевірки даних в БД та надання сенсу структурі даних. Прийнято виділяти такі обмеження.

Категорійна цілісність. Рядки реляційної таблиці представляють в БД елементи певних об’єктів реального світу або категорій. Ключ реляційної таблиці однозначно визначає кожний рядок і, як наслідок, кожен елемент категорії. Таким чином, коли користувач вилучає дані певного рядку або маніпулює ними, повинні знати значення ключа цього рядка. Відповідно недопустимим є порожнє значення ключа або частини ключа.

Правило категорійної цілісності: ніякий ключовий атрибут рядка не може бути порожнім.

Цілісність на рівні посилання. При побудові реляційних таблиць для зв’язування рядків однієї таблиці з рядками іншої використовуються зовнішні ключі. БД, в якій всі не порожні зовнішні ключі посилаються на поточні значення ключів іншої таблиці, володіє цілісністю на рівні посилання.

Правило цілісності на рівні посилання: значення не порожнього зовнішнього ключа повинно бути рівним одному з поточних значень ключа другої таблиці.

Oracle Database Architecture

Сервер бази даних є ключем до управління інформацієюю

Загалом, сервер надійно керує великою кількістю даних в багатокористувацькому середовищі, так щоб користувачі можуть одночасно отримувати доступ до тих самих даних. Сервер бази даних також запобігає несанкціонованому доступу і забезпечує ефективні рішення для відновлення після збою.

Database and Instance

Сервер бази даних Oracle складається з бази даних і, щонайменше, одного екземаляру бази даних. Так як екземпляр і база даних настільки тісно пов'язані, термін база даних Oracle іноді використовується для позначення як екземпляру і бази даних.

У строгому сенсі терміни мають таке значення:

• База даних

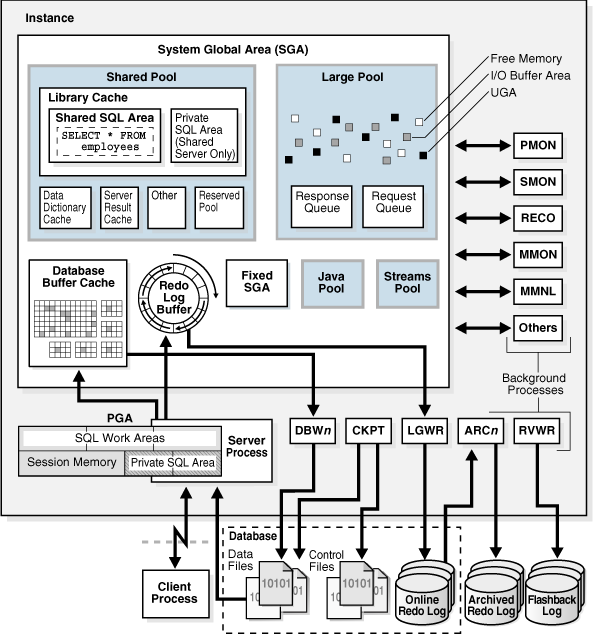
База даних являє собою набір файлів, розташованих на диску, які зберігають дані. Ці файли можуть існувати незалежно від екземпляра бази даних.

* Екземпляр бази даних
* Екземпляр являє собою набір структур пам'яті, які керують файлами бази даних. Примірник складається із загальної області пам'яті, називається системою глобальної області (SGA), а також набір фонових процесів. Примірник може існувати незалежно від файлів бази даних.

[Figure 1-1](http://docs.oracle.com/database/122/CNCPT/introduction-to-oracle-database.htm#GUID-8F2EEEC8-0372-4419-88FF-7D77A9C0FCAD__CEGDIFEF) shows a database and its instance. For each user connection to the instance, a [**client process**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-6F3BBCE8-F471-41E8-8C98-F4C62A85862B) runs the application. Each client process is associated with its own [**server process**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-E660AC1C-B704-4DC1-A35A-DB49EFB34F4A). The server process has its own private session memory, known as the [**program global area (PGA)**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-76021F69-AC7A-4D8D-A308-A7B8AC072EEF).

На малюнку 1-1 показана база даних і її екземпляр. Для кожного з'єднання користувача, процес клієнта запускає додаток. Кожен клієнтський процес пов'язаний зі своїм власним серверним процесом. Серверний процес має свою власну приватну пам'ять, відому як глобальну область програм (PGA).

***Figure 1-1 Oracle Instance and Database***

  
[Description of "Figure 1-1 Oracle Instance and Database"](http://docs.oracle.com/database/122/CNCPT/img_text/GUID-A30AF38A-7D55-4987-A0FB-02BAAF079AC5-print.htm)

База даних може розглядатися як з фізичної та логічної точки зору. Фізичні дані доступні для перегляду на рівні операційної системи. Наприклад, службові програми операційної системи, такі як lsand пс Linux можуть переглянути список файлів і процесів бази даних. Логічні дані, такі як таблиці, маєють сенс тільки для бази даних. SQL-оператор може показати список таблиць в базі даних Oracle, але утиліта операційної системи не може.

База даних мають фізичні і логічні структури. Оскільки фізичні і логічні структури розділені, ви можете управляти фізичним зберіганням даних, не впливаючи на доступ до логічних структур зберігання. Наприклад, перейменування фізичного файлу бази даних не перейменовує таблиці, дані якиі зберігаються в цьому файлі.

**Структури зберігання бази даних**

Основне завдання реляційної бази даних є зберігання даних.Коротко опишемо фізичні і логічні структури зберігання даних, які використовують Oracle Database.

Фізичні структури зберігання

Фізичні структури баз даних є файлами, в яких зберігаються дані.

When you execute a CREATE DATABASE statement, the following files are created:

При виконанні наступної команди CREATE DATABASE, створюються такі файли:

• Файли даних

Кожна база даних Oracle має один або кілька фізичних файлів даних, які містять всі дані бази даних. Дані про логічні структури бази даних, наприклад таблиці і індекси.

• Файли контралю

Кожна база даних Oracle має файл управління. Контрольний файл містить метадані, що визначають фізичну структуру бази даних, включаючи ім'я бази даних і імена і розташування файлів бази даних.

Many other files are important for the functioning of an Oracle database server. These include parameter files and networking files. Backup files and archived redo log files are offline files important for backup and recovery.

Багато інших файлів мають важливе значення для функціонування сервера бази даних Oracle. До них відносяться файли параметрів та мережеві файли. Резервне копіювання файлів і архіви журнальних файлів є важливими для резервного копіювання та відновлення.

Логічна структура

Логічні структури зберігання дозволяють Oracle Database мати контроль над простором використання пам’яті.

Логічні структури зберігання:

• Блоки даних

На найменшому рівні деталізації, Oracle Database дані зберігаються в блоках даних. Один блок даних відповідає певній кількості байт на диску.

* Екстент

Екстент це конкретне число логічно суміжних блоків даних, в одному розподілі, використовується для зберігання певного типу інформації.

* Сегменти

Сегмент являє собою набір екстентів, виділених для призначеного для користувача об'єкта (наприклад, таблиця або індекс), скасовані дані або тимчасові дані.

* Табличний простір

База даних розділена на логічні одиниці зберігання під назвою табличний простір. Табличний простір є логічним контейнером для сегмента. Кожен табличний простір складається з щонайменше одного файлу даних.

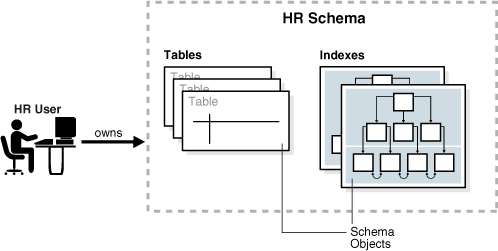
Введення Об'єкти схеми

A [database user](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-2D2B8165-D2CE-4D08-82D6-A7E1D988471B) account has a password and specific database privileges. Each user account owns a single schema, which has the same name as the user. The schema contains the data for the user owning the schema. For example, the hr user account owns the hr schema, which contains schema objects such as the employees table. In a production database, the schema owner usually represents a database application rather than a person.

Обліковий запис користувача бази даних має пароль і певні привілеї бази даних. Кожен обліковий запис користувача має одну схему. Схема містить дані для користувача, що володіє схемою. Наприклад, обліковий запис користувача володіє схемою, яка містить таблиці службовців. У виробничій базі даних, власник схеми, як правило, являє собою базу даних програми, а не людина.

У схемі, кожен об'єкт схеми конкретного типу має унікальне ім'я. Наприклад, hr.employees відноситься до працівників таблиці в hrschema. На малюнку 2-1 зображений власник схеми з ім'ям Hr і схемами об'єктів в схемі годин.

***Figure 2-1 HR Schema***



Типи схем об'єктів

Oracle SQL дозволяє створювати і управляти багатьма іншими типами об'єктів схеми.

Основні типи об'єктів схеми є:

• Таблиці

A Таблиця зберігає дані в рядках. Таблиці є найбільш важливими об'єктами схеми в реляційної базі даних.

• Індекси

Індекси є об'єктами схеми, які містять запис для кожної індексованої рядки таблиці або таблиці кластера і забезпечують прямий, швидкий доступ до рядків. Oracle Database підтримує кілька типів індексів.

• Пратішини

Партиішини є частинами великих таблиць і індексів. Кожен розділ має своє власне ім'я і необов'язково може мати свої власні характеристики зберігання.

• Віртуальні таблиці

Віртуальні таблиці -презентації даних в одній або декількох таблиць. Ви можете думати про них, як збережені запити. Віртуальні таблиці насправді не містять даних.

• Послідовність

Послідовність є створеним користувачем об'єктом, який може спільно використовуватися декількома користувачами, щоб генерувати цілі числа. Як правило, можна використовувати послідовності для генерації значень первинного ключа.

### Інші типи об'єктів також зберігаються в базі даних і можуть бути створені і управляються за допомогою SQL, але не містяться в схемі. Ці об'єкти включають в себе обліковий запис бази даних користувачів, ролі, контексти і об'єкти словника.

Схема зберігання об'єктів

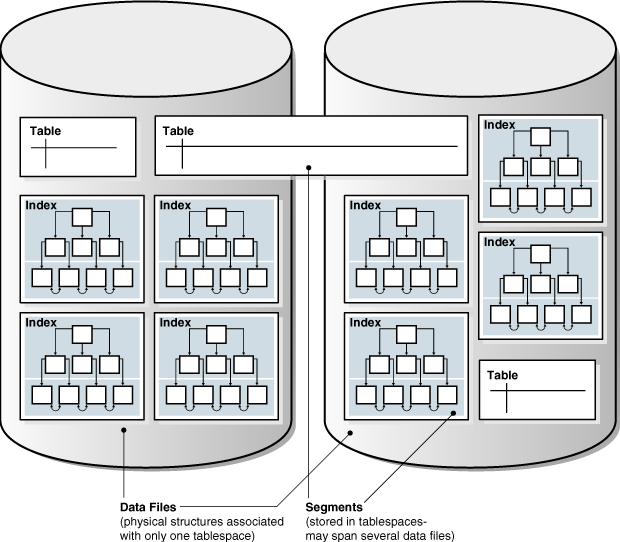
Деякі об'єкти схеми зберігають в типі логічної структури зберігання даних яка називається сегментом.

Інші об'єкти схеми, такі як Віртуальні таблиці і послідовності, складаються тільки з метаданих.

Oracle Database зберігає об'єкт схеми логічно в табличному просторі. Там немає ніякого зв'язку між схемами і табличноми просторами: табличний можуть містити об'єкти з різних схем, а також об'єкти для схеми можуть міститися в різних табличних просторах. Дані кожного об'єкта фізично містяться в одному або декількох файлах даних.

На наведеному нижче малюнку показана можлива конфігурація таблиць і індексів сегментів, табличними та файли даних. Сегмент даних для однієї таблиці охоплює два файли даних, які є частиною одного і того ж табличного простору. Сегмент не може охоплювати кілька табличних просторів

.***Figure 2-2 Segments, Tablespaces, and Data Files***



### SYS and SYSTEM Schemas

All Oracle databases include default administrative accounts. Administrative accounts are highly privileged and are intended only for DBAs authorized to perform tasks such as starting and stopping the database, managing memory and storage, creating and managing database users, and so on.

The SYS administrative account is automatically created when a database is created. This account can perform all database administrative functions. The SYS schema stores the base tables and views for the [**data dictionary**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-D6A3934F-BA6D-464F-9612-C683E20514A4). These base tables and views are critical for the operation of Oracle Database. Tables in the SYS schema are manipulated only by the database and must never be modified by any user.

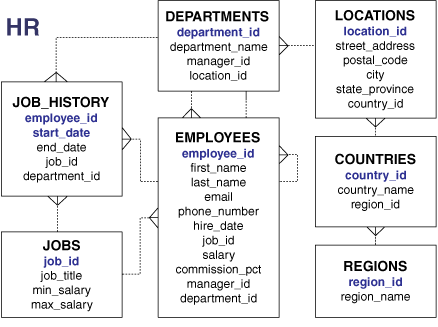
The SYSTEM administrative account is also automatically created when a database is created. The SYSTEM schema stores additional tables and views that display administrative information, and internal tables and views used by various Oracle Database options and tools. Never use the SYSTEMschema to store tables of interest to nonadministrative users.

### Sample Schemas

An Oracle database may include **sample schemas**, which are a set of interlinked schemas that enable Oracle documentation and Oracle instructional materials to illustrate common database tasks.

The hr sample schema contains information about employees, departments and locations, work histories, and so on. The following illustration depicts an entity-relationship diagram of the tables in hr. Most examples in this manual use objects from this schema.

***Figure 2-3 HR Schema***



## Overview of Tables

A **table** is the basic unit of data organization in an Oracle database.

A table describes an **entity**, which is something of significance about which information must be recorded. For example, an employee could be an entity.

Oracle Database tables fall into the following basic categories:

* Relational tables

Relational tables have simple columns and are the most common table type. [Example 2-1](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm" \l "GUID-B0DFC5A7-E482-4E17-A6F5-FF476A92DC73__CBBEHJCH) shows a CREATE TABLE statement for a relational table.

* Object tables

The columns correspond to the top-level attributes of an [**object type**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-33C40B41-75EF-4B5A-AE9F-45CB91700218). See "[Overview of Object Tables](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#GUID-EBB02ED3-525E-4D7F-B984-E8AC3BF039F7)".

You can create a relational table with the following organizational characteristics:

* A [**heap-organized table**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-861D8FCE-B86C-46B3-AA01-35066D24F4CF) does not store rows in any particular order. The CREATE TABLE statement creates a heap-organized table by default.
* An [**index-organized table**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-FD628CF1-3A67-4171-92BB-D4A74119ACD9) orders rows according to the primary key values. For some applications, index-organized tables enhance performance and use disk space more efficiently. See "[Overview of Index-Organized Tables](http://docs.oracle.com/database/122/CNCPT/indexes-and-index-organized-tables.htm#GUID-DAEC075B-C16D-4A57-898C-70EBCB364F0C)".
* An [**external table**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-1B18B978-29DD-4961-BD39-86838CB5F7D0) is a read-only table whose metadata is stored in the database but whose data is stored outside the database. See "[Overview of External Tables](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#GUID-97FC9DFF-A6CF-46CD-9F6F-D88A37C0E79C)".

A table is either permanent or temporary. A permanent table definition and data persist across sessions. A [**temporary table**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-41501962-A37B-4D37-BAF2-59030048FE08) definition persists in the same way as a permanent table definition, but the data exists only for the duration of a [**transaction**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-212D8EA1-D704-4D7B-A72D-72001965CE45) or [**session**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-0F44C072-9841-4E2E-B846-FB16A2E54139). Temporary tables are useful in applications where a result set must be held temporarily, perhaps because the result is constructed by running multiple operations.

### Columns

A table definition includes a table name and set of columns.

A [**column**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-F1D9AA5F-C66A-4D5F-A84C-8D7360DE3BE5) identifies an attribute of the entity described by the table. For example, the column employee\_id in the employees table refers to the employee ID attribute of an employee entity.

In general, you give each column a column name, a [**data type**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-3873B26C-657D-4508-B13D-9155F1D5D8F4), and a width when you create a table. For example, the data type for employee\_id is NUMBER(6), indicating that this column can only contain numeric data up to 6 digits in width. The width can be predetermined by the data type, as with DATE.

#### Virtual Columns

A table can contain a **virtual column**, which unlike a nonvirtual column does not consume disk space.

The database derives the values in a virtual column on demand by computing a set of user-specified expressions or functions. For example, the virtual column income could be a function of the salary and commission\_pct columns.

#### Invisible Columns

An **invisible column** is a user-specified column whose values are only visible when the column is explicitly specified by name. You can add an invisible column to a table without affecting existing applications, and make the column visible if necessary.

In general, invisible columns help migrate and evolve online applications. A use case might be an application that queries a three-column table with a SELECT \* statement. Adding a fourth column to the table would break the application, which expects three columns of data. Adding a fourth invisible column makes the application function normally. A developer can then alter the application to handle a fourth column, and make the column visible when the application goes live.

The following example creates a table products with an invisible column count, and then makes the invisible column visible:

CREATE TABLE products ( prod\_id INT, count INT INVISIBLE );

ALTER TABLE products MODIFY ( count VISIBLE );

### Oracle Data Types

Each column has a **data type**, which is associated with a specific storage format, constraints, and valid range of values. The data type of a value associates a fixed set of properties with the value.

These properties cause Oracle Database to treat values of one data type differently from values of another. For example, you can multiply values of the NUMBER data type, but not values of the RAW data type.

When you create a table, you must specify a data type for each of its columns. Each value subsequently inserted in a column assumes the column data type.

Oracle Database provides several built-in data types. The most commonly used data types fall into the following categories:

* [Character Data Types](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#GUID-5D05C023-C309-46DA-A7EC-2BED1CA512A7)
* [Numeric Data Types](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#GUID-AB604FF0-D321-44CC-956E-62A7791EE2FA)
* [Datetime Data Types](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#GUID-F5FBE34D-911C-4818-9D25-64B2F1613A9B)
* [Rowid Data Types](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#GUID-0258C4C2-2BF2-445F-B1E1-F282A57A6859)
* [Format Models and Data Types](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#GUID-A126BA2F-8249-4316-BC18-F12B02004550)

Other important categories of built-in types include raw, large objects (LOBs), and collections. PL/SQL has data types for constants and variables, which include BOOLEAN, reference types, composite types (records), and user-defined types.

**See Also:**

* "[Overview of LOBs](http://docs.oracle.com/database/122/CNCPT/concepts-for-database-developers.htm#GUID-E918382E-0E0A-444C-ABD3-BCC6EFC0AB38)"
* [Oracle Database SQL Language Reference](http://docs.oracle.com/database/122/SQLRF/Data-Types.htm#SQLRF0021) to learn about built-in SQL data types
* [Oracle Database PL/SQL Packages and Types Reference](http://docs.oracle.com/database/122/LNPLS/plsql-data-types.htm#LNPLS003) to learn about PL/SQL data types
* [Oracle Database Development Guide](http://docs.oracle.com/database/122/ADFNS/sql-data-types.htm#ADFNS0002) to learn how to use the built-in data types

#### Character Data Types

Character data types store alphanumeric data in strings. The most common character data type is VARCHAR2, which is the most efficient option for storing character data.

The byte values correspond to the [**character encoding**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-770C0F15-8AA7-446B-9DF8-455B8B406A4D) scheme, generally called a [**character set**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-9EDC68FA-88D3-4F15-85C4-2B603C3240CE). The database character set is established at database creation. Examples of character sets are 7-bit ASCII, EBCDIC, and Unicode UTF-8.

The length semantics of character data types are measurable in bytes or characters. The treatment of strings as a sequence of bytes is called [**byte semantics**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-BBEA62B4-5767-40DF-B2D6-BEC1F267B22F). This is the default for character data types. The treatment of strings as a sequence of characters is called [**character semantics**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-46D094F4-88B8-4939-AFBD-971EA3AABBDA). A character is a code point of the database character set.

**See Also:**

* "[Character Sets](http://docs.oracle.com/database/122/CNCPT/concepts-for-database-developers.htm#GUID-57374356-61C4-45B1-8A05-3353FD034202)"
* [Oracle Database 2 Day Developer's Guide](http://docs.oracle.com/database/122/TDDDG/creating-managing-schema-objects.htm#TDDDG31000) and [Oracle Database Development Guide](http://docs.oracle.com/database/122/ADFNS/sql-data-types.htm#ADFNS00302) and to learn how to select a character data type

##### **VARCHAR2 and CHAR Data Types**

The VARCHAR2 data type stores variable-length character literals. A **literal** is a fixed data value.

For example, 'LILA', 'St. George Island', and '101' are all character literals; 5001 is a numeric literal. Character literals are enclosed in single quotation marks so that the database can distinguish them from schema object names.

**Note:**

This manual uses the terms *text literal*, *character literal*, and *string* interchangeably.

When you create a table with a VARCHAR2 column, you specify a maximum string length. In [Example 2-1](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#GUID-B0DFC5A7-E482-4E17-A6F5-FF476A92DC73__CBBEHJCH), the last\_name column has a data type of VARCHAR2(25), which means that any name stored in the column has a maximum of 25 bytes.

For each row, Oracle Database stores each value in the column as a variable-length field unless a value exceeds the maximum length, in which case the database returns an error. For example, in a single-byte character set, if you enter 10 characters for the last\_name column value in a row, then the column in the row piece stores only 10 characters (10 bytes), not 25. Using VARCHAR2 reduces space consumption.

In contrast to VARCHAR2, CHAR stores fixed-length character strings. When you create a table with a CHAR column, the column requires a string length. The default is 1 byte. The database uses blanks to pad the value to the specified length.

Oracle Database compares VARCHAR2 values using nonpadded comparison semantics and compares CHAR values using blank-padded comparison semantics.

**See Also:**

[Oracle Database SQL Language Reference](http://docs.oracle.com/database/122/SQLRF/Data-Type-Comparison-Rules.htm#SQLRF51040) for details about blank-padded and nonpadded comparison semantics

##### **NCHAR and NVARCHAR2 Data Types**

The NCHAR and NVARCHAR2 data types store Unicode character data.

**Unicode** is a universal encoded character set that can store information in any language using a single character set. NCHAR stores fixed-length character strings that correspond to the national character set, whereas NVARCHAR2 stores variable length character strings.

You specify a national character set when creating a database. The character set of NCHAR and NVARCHAR2 data types must be either AL16UTF16 or UTF8. Both character sets use Unicode encoding.

When you create a table with an NCHAR or NVARCHAR2 column, the maximum size is always in character length semantics. Character length semantics is the default and only length semantics for NCHAR or NVARCHAR2.

**See Also:**

[Oracle Database Globalization Support Guide](http://docs.oracle.com/database/122/NLSPG/programming-with-unicode.htm#NLSPG0071) for information about Oracle's globalization support feature

#### Numeric Data Types

The Oracle Database numeric data types store fixed and floating-point numbers, zero, and infinity. Some numeric types also store values that are the undefined result of an operation, which is known as "not a number" or NaN.

Oracle Database stores numeric data in variable-length format. Each value is stored in scientific notation, with 1 byte used to store the exponent. The database uses up to 20 bytes to store the [**mantissa**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-0912AA8C-6F9E-4382-97A3-2B43832BEBE6), which is the part of a floating-point number that contains its significant digits. Oracle Database does not store leading and trailing zeros.

##### **NUMBER Data Type**

The NUMBER data type stores fixed and floating-point numbers. The database can store numbers of virtually any magnitude. This data is guaranteed to be portable among different operating systems running Oracle Database. The NUMBER data type is recommended for most cases in which you must store numeric data.

You specify a fixed-point number in the form NUMBER(*p*,*s*), where *p* and *s* refer to the following characteristics:

* Precision

The [**precision**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-84EF5156-C5F0-49CA-BFAD-AAEC3C5F9289) specifies the total number of digits. If a precision is not specified, then the column stores the values exactly as provided by the application without any rounding.

* Scale

The [**scale**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-7CA2255F-476D-461C-967A-6FABC5DAA8CE) specifies the number of digits from the decimal point to the least significant digit. Positive scale counts digits to the right of the decimal point up to and including the least significant digit. Negative scale counts digits to the left of the decimal point up to but not including the least significant digit. If you specify a precision without a scale, as in NUMBER(6), then the scale is 0.

In [Example 2-1](http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#GUID-B0DFC5A7-E482-4E17-A6F5-FF476A92DC73__CBBEHJCH), the salary column is type NUMBER(8,2), so the precision is 8 and the scale is 2. Thus, the database stores a salary of 100,000 as 100000.00.

##### **Floating-Point Numbers**

Oracle Database provides two numeric data types exclusively for floating-point numbers: BINARY\_FLOAT and BINARY\_DOUBLE.

These types support all of the basic functionality provided by the NUMBER data type. However, whereas NUMBER uses decimal precision, BINARY\_FLOATand BINARY\_DOUBLE use binary precision, which enables faster arithmetic calculations and usually reduces storage requirements.

BINARY\_FLOAT and BINARY\_DOUBLE are approximate numeric data types. They store approximate representations of decimal values, rather than exact representations. For example, the value 0.1 cannot be exactly represented by either BINARY\_DOUBLE or BINARY\_FLOAT. They are frequently used for scientific computations. Their behavior is similar to the data types FLOAT and DOUBLE in Java and XMLSchema.

**See Also:**

[Oracle Database SQL Language Reference](http://docs.oracle.com/database/122/SQLRF/Data-Types.htm#SQLRF00222) to learn about precision, scale, and other characteristics of numeric types

#### Datetime Data Types

The **datetime** data types are DATE and TIMESTAMP. Oracle Database provides comprehensive time zone support for time stamps.

##### **DATE Data Type**

The DATE data type stores date and time. Although datetimes can be represented in character or number data types, DATE has special associated properties.

The database stores dates internally as numbers. Dates are stored in fixed-length fields of 7 bytes each, corresponding to century, year, month, day, hour, minute, and second.

**Note:**

Dates fully support arithmetic operations, so you add to and subtract from dates just as you can with numbers. See [Oracle Database Development Guide](http://docs.oracle.com/database/122/ADFNS/sql-data-types.htm#ADFNS202).

The database displays dates according to the specified [**format model**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-F97AA700-3D17-428F-B9E0-52259D19C037). A format model is a character literal that describes the format of a datetime in a character string. The standard date format is DD-MON-RR, which displays dates in the form 01-JAN-11.

RR is similar to YY (the last two digits of the year), but the century of the return value varies according to the specified two-digit year and the last two digits of the current year. Assume that in 1999 the database displays 01-JAN-11. If the date format uses RR, then 11 specifies 2011, whereas if the format uses YY, then 11 specifies 1911. You can change the default date format at both the database instance and session level.

Oracle Database stores time in 24-hour format—HH:MI:SS. If no time portion is entered, then by default the time in a date field is 00:00:00 A.M. In a time-only entry, the date portion defaults to the first day of the current month.

**See Also:**

* [Oracle Database Development Guide](http://docs.oracle.com/database/122/ADFNS/sql-data-types.htm#ADFNS182) for more information about centuries and date format masks
* [Oracle Database SQL Language Reference](http://docs.oracle.com/database/122/SQLRF/Format-Models.htm#SQLRF00212) for information about datetime format codes

##### **TIMESTAMP Data Type**

The TIMESTAMP data type is an extension of the DATE data type.

TIMESTAMP stores fractional seconds in addition to the information stored in the DATE data type. The TIMESTAMP data type is useful for storing precise time values, such as in applications that must track event order.

The DATETIME data types TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE are time-zone aware. When a user selects the data, the value is adjusted to the time zone of the user session. This data type is useful for collecting and evaluating date information across geographic regions.

**See Also:**

[Oracle Database SQL Language Reference](http://docs.oracle.com/database/122/SQLRF/Data-Types.htm#SQLRF00203) for details about the syntax of creating and entering data in time stamp columns

#### Rowid Data Types

Every row stored in the database has an address. Oracle Database uses a ROWID data type to store the address (rowid) of every row in the database.

Rowids fall into the following categories:

* Physical rowids store the addresses of rows in heap-organized tables, table clusters, and table and index partitions.
* Logical rowids store the addresses of rows in index-organized tables.
* Foreign rowids are identifiers in foreign tables, such as DB2 tables accessed through a gateway. They are not standard Oracle Database rowids.

A data type called the [**universal rowid**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-288B0853-9B77-4A7F-AEE9-8924EA7877A7), or urowid, supports all types of rowids.

##### **Use of Rowids**

Oracle Database uses rowids internally for the construction of indexes.

A [**B-tree index**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-8D6D0C64-6AC8-4B22-A9AF-1B62F61AE10B), which is the most common type, contains an ordered list of keys divided into ranges. Each key is associated with a rowid that points to the associated row's address for fast access.

End users and application developers can also use rowids for several important functions:

* Rowids are the fastest means of accessing particular rows.
* Rowids provide the ability to see how a table is organized.
* Rowids are unique identifiers for rows in a given table.

You can also create tables with columns defined using the ROWID data type. For example, you can define an exception table with a column of data type ROWID to store the rowids of rows that violate integrity constraints. Columns defined using the ROWID data type behave like other table columns: values can be updated, and so on.

##### **ROWID Pseudocolumn**

Every table in an Oracle database has a pseudocolumn named ROWID.

A [**pseudocolumn**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-175D4923-5C7E-4FF0-A69B-C4D8F3D93A3D) behaves like a table column, but is not actually stored in the table. You can select from pseudocolumns, but you cannot insert, update, or delete their values. A pseudocolumn is also similar to a SQL [**function**](http://docs.oracle.com/database/122/CNCPT/glossary.htm#GUID-5611828A-3621-4527-AEB1-12017A454E94) without arguments. Functions without arguments typically return the same value for every row in the result set, whereas pseudocolumns typically return a different value for each row.

Values of the ROWID pseudocolumn are strings representing the address of each row. These strings have the data type ROWID. This pseudocolumn is not evident when listing the structure of a table by executing SELECT or DESCRIBE, nor does the pseudocolumn consume space. However, the rowid of each row can be retrieved with a SQL query using the reserved word ROWID as a column name.

The following example queries the ROWID pseudocolumn to show the rowid of the row in the employees table for employee 100:

SQL> SELECT ROWID FROM employees WHERE employee\_id = 100;

ROWID

------------------

AAAPecAAFAAAABSAAA

http://docs.oracle.com/database/122/CNCPT/tables-and-table-clusters.htm#CNCPT010

Реалізація додатку

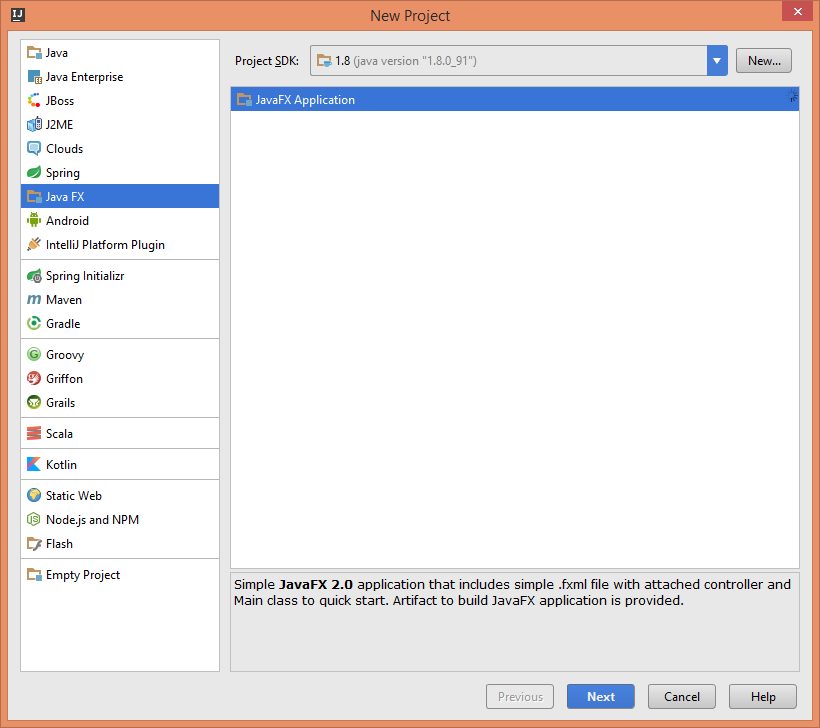
Реалізував кросплатформний додаток який дозволяє керувати СУБД ORACLE. Проект написаний використовуючи JAVA. Користувацький інтерфейс створено за підтримки технології Java FX та патерну проектування MVC (Model-View-Control).У проекті реалізовані базові функції CRUD представлення метаінформації,як окремих таблиць ,так і цілої бази даних. Створений конструктор таблиць з можливістю додавання зовнішніх ключів та індексів. Реалізована можливість створення SQL запитів самим користувачем.

Для створення GUI використовується технологія JavaFX .Для підключення до бази даних був обраний JDBC інтерфейс.

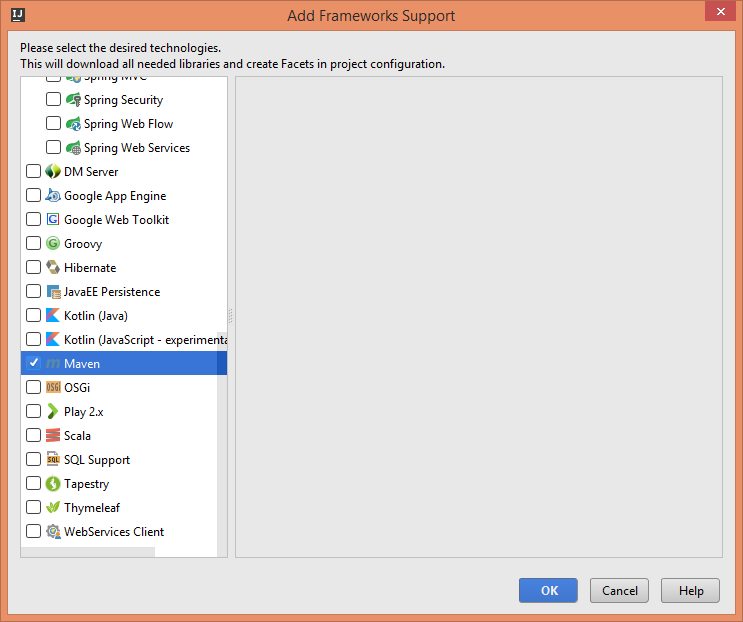
Створення і налаштування проекту

Для створення додаткиів з користувацьким інтерфейсом в мові Java є декілька платформ а саме JavaFX, Swing , Avt в цьому проекті я вирішив використовувати JavaFX так як ця платворма надає дуже широкий інтерфейс прикладного програмування та розширює можливості Swing платформи.

Створимо пустий додаток



Для  управління та складання будемо використовувати засіб автоматизації роботи з [програмними проектами](https://uk.wikipedia.org/w/index.php?title=%D0%A0%D0%9E%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F&action=edit&redlink=1) Maven. Спочатку додаємо фреймворк до проекту



За допомогою мавена додаємо потрібні бібліотеки для цього відкриваємо файл pom.xml який знаходиться в кореневій папці проекту додаємо теги

<dependencies></dependencies>

В тіло цього тегу ми і будемо додавати теги які автоматично додадуть в проект потрібні нам бібліотеки.

Додаємо

* JDBC драйвер

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc14</artifactId>

<version>10.2.0.4.0</version>

</dependency>

* JUNIT

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

Змінимо деякі налаштування проекту

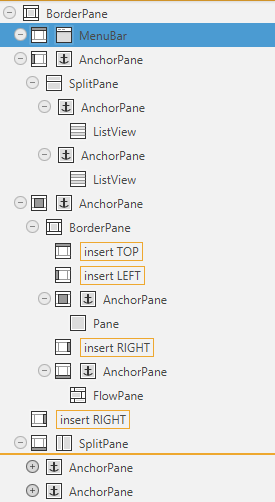
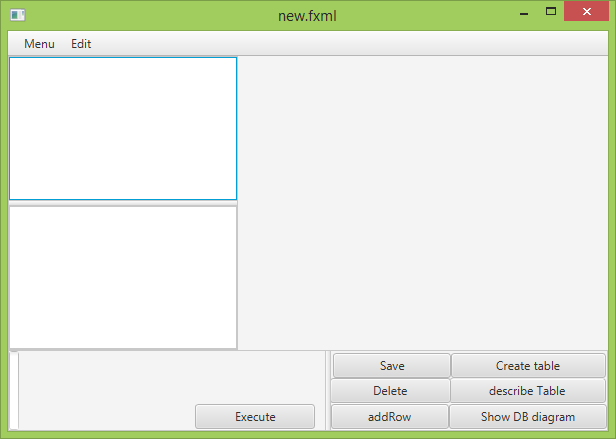
<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

За допомогою SceneByilder створимо початкове вікно в основі якого буде лежати BorderPane layout



**Зв'язок з базою даних**

Через те що щоб відкрити зв'язок з базою даних потрібно затратити багато ресурсів реалізуємо його за паттерном singleton.

Створимо статичний блок ініціалізації

**static**{**try** {  
 connection = DriverManager.getConnection(**"jdbc:oracle:thin:@localhost:1521:xe"**,login,pass);  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }

та матод який поертає проініціалізований звязок

**static public** Connection getConnect(){  
 **return** *connection*;  
}

через те що поле **private static** Connection *connection*;

Є статичне та при кожному зверненні метод буде повертати вже заздалегіть створене з'єднання.

**Відображення списку баз даних їх таблиць та вмісту таблиць**

Спочатку отримуємо список усіх баз даних як є

ObservableList<String> items = FXCollections.*observableArrayList*(**dbUtils**.getAllDatabases());  
**userList**.setItems(items);  
**tablePanel**.getChildren().add(**tableNode**);

Додаємо listener на userList для того щоб коли користувач обере базу даних

В іншому списку відобразились усі таблиці.

<**ListView fx:id="userList" onMouseClicked="#onChoseUser"...**

Код меоду який викликаєтся коли користувач обирає потрібного базу даних

**public void** onChoseDb() {  
 String name = **userList**.getSelectionModel().getSelectedItem();  
 ObservableList<String> items = FXCollections.*observableArrayList*(**dbUtils**.getDbTables(name));  
 **dbList**.setItems(items);  
}

приклад коду який безпосередньо дістає з бази список усіх таблиць для потрібного неймспейсу(бази даних).

**public** ArrayList<String> getUserTables(String name){  
 ArrayList<String> names = **new** ArrayList<String>();  
 **try** {  
 String qu = **"SELECT table\_name FROM dba\_tables where TABLESPACE\_NAME =?"**;  
 PreparedStatement preparedStatement = **connection**.prepareStatement(qu);  
 preparedStatement.setString(1, name);  
 ResultSet rs = preparedStatement.executeQuery();  
 names = GetLists.*getStringList*(rs);  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 } **return** names;  
}

Після того як користувач обере потрібну таблицю ActionListener викличе метод який відобразить дані в цій таблиці. Для цього створимо метод який з ResultSet створює готову компоненту JTable заповнену даними.

Щоб отримати назви колонок та їх кількість отрмаємо обєкт ResultSetMetaData з ResultSet.

ResultSetMetaData rsmd = rs.getMetaData();  
**int** numberOfColumns = rsmd.getColumnCount();  
ArrayList<ArrayList<String>> data = **new** ArrayList<>();  
 columnNames=**new** String[numberOfColumns];  
**for** (**int** i = 1; i <numberOfColumns+1 ; i++) {  
 columnNames[i-1]=rsmd.getColumnName(i);  
}

Отримуємо дані по рядкам

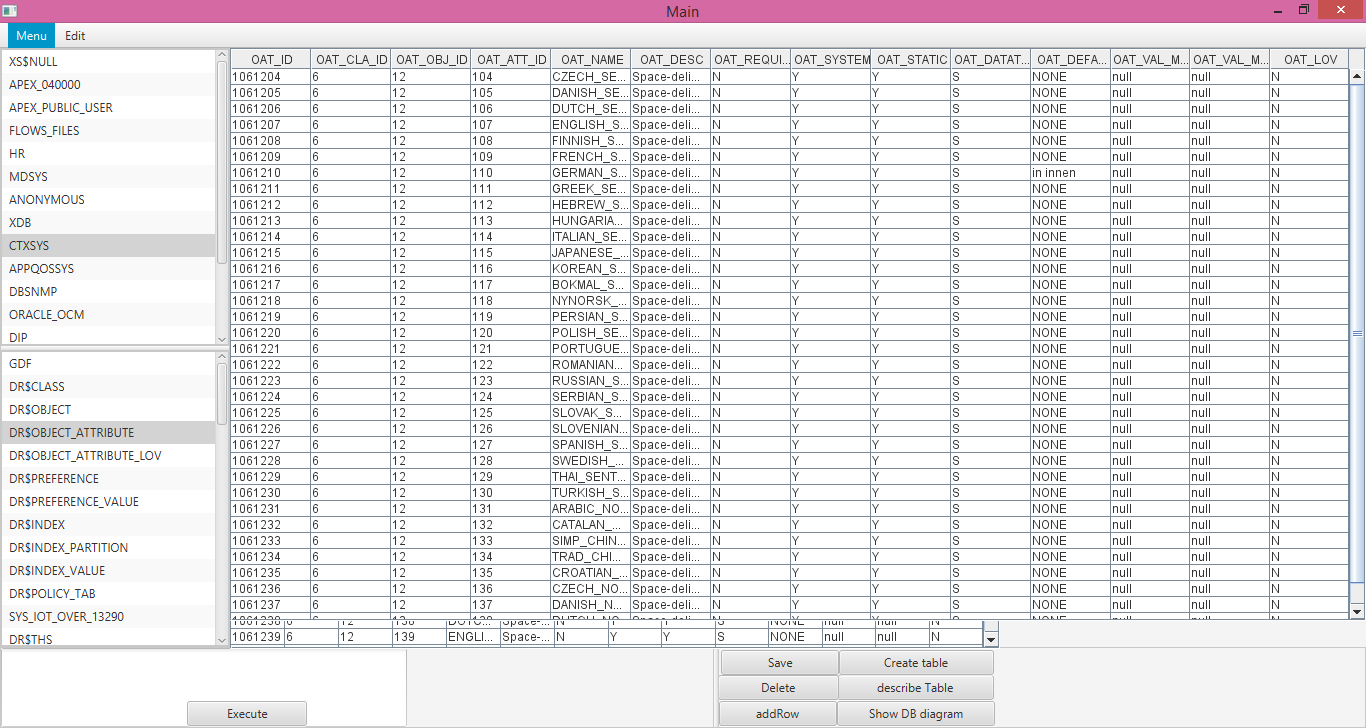
**while** (rs.next()) {  
 **for** (**int** i = 1; i < numberOfColumns+1; i++) {  
 data.get(i-1).add(String.*valueOf*(rs.getObject(i)));  
 }}

Повертаємо створену табліцю заповнену даними.

**return new JTable(dataArr, columnNames);**

Відображаємо отриману таблицю на формі.

String name = **userList**.getSelectionModel().getSelectedItem();  
String table = **dbList**.getSelectionModel().getSelectedItem();  
JTable table1 = **dbUtils**.getUserTable(name, table);  
setTable(table1);



**Створення слахача подій для компоненти JTable та Action класу.**

**Є потреба у створенні CRUD операцій для цього створимо** ActionListener який буде фіксувати зміни в поточній таблиці клас має відповідати таким вимогам

Фіксувати рядок та колонку в якій були здійснені зміни

Фіксувати нове та старе значення комірки

Реалізований клас типу Action якому будуть передаватися дані на основі яких буде генеруватися SQL команда для збаережання змін в базі даних

**public class** TableCellListener **implements** PropertyChangeListener{

Створюємо потрібні поля для збареання даних

**private** Action **action**;  
**private int row**;  
**private int column**;  
**private** Object **oldValue**;  
**private** Object **newValue**;

@Override  
**public void** propertyChange(PropertyChangeEvent e) {   
  
 **if** (**"tableCellEditor"**.equals(e.getPropertyName())) {  
 **if** (**table**.isEditing())  
 processEditingStarted();  
 **else** processEditingStopped();  
 }  
}

Після закінчення редагування викликаєтся метод який перевіряє чи змінилися дані якщо змінилися то викликаємо відповідний Action передаючи йому екземпляр ActionEvent з усією потрібною інформацією.

**private void** processEditingStopped() {  
 **newValue** = **table**.getModel().getValueAt(**row**, **column**);  
**if** (!**newValue**.equals(**oldValue**)) {  
TableCellListener tcl = **new** TableCellListener(  
 getTable(), getRow(), getColumn(), getOldValue(), getNewValue());  
 ActionEvent event = **new** ActionEvent(  
 tcl,  
 ActionEvent.***ACTION\_PERFORMED***,  
 **""**);  
 **action**.actionPerformed(event);  
 }}

Клас Action

**public class** changeCellListener **implements** Action {  
 **public void** actionPerformed(ActionEvent e) {  
 TableCellListener tcl = (TableCellListener) e.getSource(); String goodSQL =**"update "**;  
 goodSQL=goodSQL+Data.*getDbTable*().replace(**" "**,**"."**)+ **" SET "**+ Data.*getColumnNames*()[tcl.getColumn()]+**"='"**+tcl.getNewValue()+**"'"**;  
 goodSQL=goodSQL+**" where "** +Data.*getColumnNames*()[tcl.getColumn()] + **" ='"** + tcl.getOldValue() +**"' and "** + Data.*getColumnNames*()[0] + **" = '"** + Data.*getDataArr*()[tcl.getRow()][0]+ **"'"**;  
 **changeList**.addDBAction(goodSQL);

Метод actionPerformed генерує потрібну SQL команду і записує її в список змін.

Збереження змін як було зазначено вище при кожній зміні геренуєтся SQL команда

ArrayList<DBAction> all = ChangeList.*getActions*();  
Connection connect = DBConnection.*getConnect*();  
**try** {  
 Statement statement = connect.createStatement();  
 **for** (DBAction dba : all) {  
 statement.executeUpdate(dba.getQuery());  
 }  
 connect.commit();  
} **catch** (SQLException e) {  
 e.printStackTrace();  
}  
System.***out***.println(**"UPDATE READY"**);  
після успішного апдейту очищуємо список.  
ChangeList.*setActions*(**new** ArrayList<DBAction>());

Якщо користувач змінив поточну таблицю без збереження змін спосок змін таксамо очищаєтся

**Видалення рядка**

Додамо action на кнопку яка відповідає за видалення рядка

<**Button layoutX="-1.0" layoutY="27.0" mnemonicParsing="false" action="#deleteRow"**

Тепер при натисненні на цю кнопку JavaFX викличе метод deleteRow()

Перед сигнатурою метода додамо анотацію @FXML для того щоб фреймворк бачив привітні методи

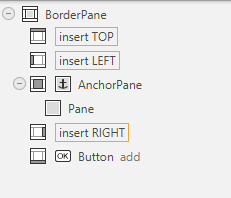
@FXML  
**private void** deleteRow() {  
 String delString = **"delete From "**;

Формуємо команду  
 **for** (**int** i = 0; i < data[1].**length**; i++) {  
 Double test = Double.*valueOf*(data[**table**.getSelectedRow()][i]);  
 delString += Data.*getDbTable*().replace(**" "**, **"."**) + **" where "** + columns[i] + **" = "** + data[**table**.getSelectedRow()][i] + **""**;  
 } **catch** (Throwable e) {  
 delString += Data.*getDbTable*().replace(**" "**, **"."**) + **" where "** + columns[i] + **" = '"** + data[**table**.getSelectedRow()][i] + **" '"**;  
 }

Отримуємо підключення до бази даних виконуємо команду та підтверджуємо транзакцію  
 Connection connection = DBConnection.*getConnect*();  
 Statement statement = **null**;  
 **try** {  
 statement = connection.createStatement();  
 statement.executeUpdate(delString);statement.execute(**"commit"**);  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 } }

**Додавання нового рядка**

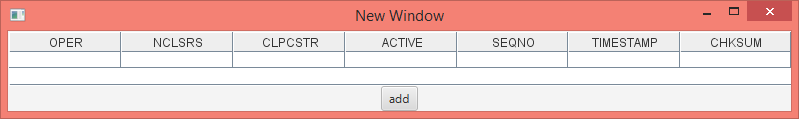
Створюю нове вікно за допомогою Scene Builder



Спочатку перевизначимо метод initialize() таким чином щоб створювалась нова таблиця з одним пустим рядком та ідентичними колонками з нашою поточною таблицею. Для цього створимо метод

**private** JTable curentEmptyTable() {  
 String[] columnNames = Data.*getColumnNames*();  
 String[][] data = **new** String[1][columnNames.**length**];  
 **for** (**int** i = 0; i < columnNames.**length**; i++) {  
 **for** (**int** j = 0; j < 1; j++) {  
 data[j][i] = **""**;  
 }  
 }  
 **table** = **new** JTable(data, columnNames);  
 setTable(**table**);  
 **return table**;  
}

Створює таблицю та відображає її у віконому інтерфейсі методом setTable(**table**);



Дороб адд.

Створення підпрограми для візуалізації структури бази даних

Щоб візуалізувати структуру бази даних спочатку створимо декілька допоміжних класів.Нас цікавлять імена таблиць,назви колонок в цих тблицях,типи даних їх розмір.Ми таксамо хочемо зобразити зовнішні ключі та первинний ключі .

Перший клас відповідає за колонку в базі даних.

public class DBColumn {

public String name;

public String type;

public String size;

public boolean primaryKey;

Інший клас відповідає за зв’язки в базі даних

public class DBRelation {

public String pk\_key;

public String pk\_table;

public String fk\_key;

public String fk\_table;

Найважливішим класом є клас DBMetaData який буде отримувати з бази даних інформацію про колонки , таблиці та зв’язки у бази даних .

Створимо потрібні поля та ініціалізуємо їх в статичних блоках .

static Connection connection = null;  
static DatabaseMetaData metadata = null;  
static { try {  
 connection = DBConnection.getConnection();  
 } catch (SQLException e) {  
 System.err.println("There was an error getting the connection: "  
 + e.getMessage());  
 } try {  
 metadata = connection.getMetaData();  
 } catch (SQLException e) {  
 System.err.println("There was an error getting the metadata: "  
 + e.getMessage()); }}

Створимо метод який буде отримувати список таблиць для цього з об’єкту метадати отримаємо список таблиць викликавшм метод getTables який має чотири параметри catalog, schemaPattern, tableNamePattern, types.Перший та третій параметр для нас неважливі.Другий параметр відповідає з якої схеми будуть таблиці тут ми вводимо назву нашої бази диних та в якості параметра вводимо

String table[] = {"TABLE"};

так як нам потрібна інформація тільки про таблиці.

String a[]=Data.getDbTable().split(" ");  
 rs = metadata.getTables(null, a[0], null, table);  
 while (rs.next()) {  
 tables.add(rs.getString("TABLE\_NAME"));  
 }  
 return tables;  
}

Отримання інформації про зв’язки для кожної таблиці для кожної таблиці яку ми отримали отримуємо метаінформацію по ключах.Метод getImportedKeys приймає три параметри каталог в базі даних,схема та назва тибліці.

for (String tableName : tableNames) {

ResultSet foreignKeys = metadata.getImportedKeys(connection.getCatalog(), null, tableName);

По всіх отриманих зовнішних ключах отримумо потрібну інформацію

while (foreignKeys.next()) {  
 String fkTableName = foreignKeys.getString("FKTABLE\_NAME");  
 String fkColumnName = foreignKeys.getString("FKCOLUMN\_NAME");  
 String pkTableName = foreignKeys.getString("PKTABLE\_NAME");  
 String pkColumnName = foreignKeys.getString("PKCOLUMN\_NAME");  
 DBRelation relation = new DBRelation(pkColumnName, pkTableName, fkColumnName, fkTableName);  
 result.add(relation);  
}

та створюємо екземпляр класу DBRelation.

Створюємо метод для отримання даних по колонках в таблицях,а саме назві колонки ,тип ,розмір поля та чи є колнока ключовим полем.

По-перше ми виклакаємо getPrimaryKeys () метод, передаючи 3 параметра до нього. Тільки TABLENAME є ненульовим.

rs = metadata.getPrimaryKeys(null, null, tableName);

ResultSet повернутого getPrimaryKeys () метда містить список стовпців, які складають первинний ключ цієї таблиці. Колонка з четвертим індексом містить ім'я стовпця.

Первинний ключ може складатися з декількох стовпців. Такий ключ називається складовим ключем. Якщо таблиці містять складові ключі, то ResultSet буде містити кілька рядків. Один рядок для кожного стовпця в ключі з'єднання.

ArrayList<String> primaryKeys = new ArrayList<String>();  
while (rs.next()) {  
 primaryKeys.add(rs.getString("COLUMN\_NAME"));}

Отримуємо список стовпців та проходимо по ньому.

ArrayList<DBColumn> result = new ArrayList<DBColumn>();  
rs = metadata.getColumns(null, null, tableName, null);

while (rs.next()) {

Створюємо обєкт DBColumn та заповнюємо його поля за допоогою конструктора потрібними значеннями.  
 DBColumn column = new DBColumn(rs.getString("COLUMN\_NAME"), rs.getString("TYPE\_NAME"),  
 rs.getString("COLUMN\_SIZE"));  
Перевіряємо чи поле є ключовим.  
 for(String primaryKey : primaryKeys) {  
 if (primaryKey.equals(column.name))  
 column.primaryKey = true; }  
 result.add(column);}

**Створення користувацького інтерфейсу**

У цьому підпроекті будемо використовувати Swing для побудови GUI

Ми будемо використовувати екземпляр класу Diagram. управління DiagramView, в JScrollPane для маштабування. Те, що ми хочемо показати на схемі, можна прокручуватися по горизонталі і по вертикалі і змінювати масштаб відповідними компонентами управління.

Ми створюємо діаграму

diagram = new Diagram();

Діаграма потребує DiagramView для відображення інформації. Ми створюємо і додамо її JScrollPane. Панель прокрутки забезпечує необхідні смуги прокрутки, якщо блок-схема стає занадто велика.

diagramView = new DiagramView(diagram);

diagramView.setVisible(true);

scrollPane = new JScrollPane(diagramView);

scrollPane.setVisible(true);

scrollPane.setAutoscrolls(true);

Бібліотека Java Diagram надає кілька допоміжних елементів управління і в нашому прикладі ми будемо використовувати елемент управління масштабуванням

zoomer = new ZoomControl();

zoomer.setView(diagramView);

zoomer.setPreferredSize(new Dimension(70, 50));

zoomer.setVisible(true);

Ми використовуємо метод setView(), щоб зв'язати елемент маштабування з DiagramView. Алгоритм BorderLayout допомагає нам організувати розташування компонент. По-перше, ми застосовуємо його на JFrame ContentPane, а потім додаємо компонент масштабування в праву частину BorderLayout . Потім, ми додаємо JScrollPane до центру, що дозволяє йому використовувати весь доступний простір.

getContentPane().setLayout(new BorderLayout());

this.add(zoomer, BorderLayout.EAST);

this.add(\_scrollPane, BorderLayout.CENTER);

**V. Creating the Tables**

Спочатку читаємо дані по колонокам в БД, а потім для кожної таблиці в базі даних, ми створюємо діаграму TableNode.

ArrayList tableData = DBMetaData.getColumnsMetadata (TABLENAME);

Ми використовуємо клас Factory з бібліотеки діаграм щоб отримати потрібну компоненту.

Dimension tableSize = new Dimension(50, 30);

TableNode \_table = diagram.getFactory().createTableNode(10, 10, 50, tableData.size() \* 8, 4, tableData.size());

table.setCaption("<b>" + tableName + "</b>");

table.setId(tableName);

Метод createTableNode (), які ми використовуємо приймає такі аргументи розташування таблиці, її ширина і висота і кількість рядків і стовпців. Ми використовуємо HTML форматування, щоб зробити заголовок таблиці жирним шрифтом. Для заголовка і ідентифікатора tableNode ми використовуємо ім'я таблиці.

Так як ми використовуємо HTML потрібно підключити цю функцію.

diagram.setEnableStyledText(true);

Коли Factory створює компоненту вона автоматично додається в колекцію контролю DiagramNodes.

Заповнюємо таблицю даними.

int rowIndex = 0;

for(DBColumn column: tableData)

{

table.getCell(1, rowIndex).setText("<b>" + column.name + "</b>");

table.getCell(2, rowIndex).setText(column.type);

table.getCell(3, rowIndex).setText(column.size);

if(column.isPrimaryKey())

{

try {

File pathToFile = new File("res/key.png");

Image image = ImageIO.read(pathToFile);

table.getCell(0,rowIndex).setImage(image);

} catch (IOException ex) {

ex.printStackTrace();

} } rowIndex++;}

Тут ми перебирати всі об'єкти DBColumn, які ми створили, використовуючи метадані бази даних. У кожному рядку в таблиці пишемо ім'я поля, тип даних і розмір даних. Починаємо з другої колонки (індекс 1), так як перший один зарезервований для зображення первинного або зовнішнього ключа.

**Зв'язки між таблицями**

Ми читаємо метадані відносини з допоміжним методом в класі DBMetaData:

ArrayList relations = DBMetaData.getRelationsMetadata(tables);

Метод повертає список наших об'єктів DBRelation, де кожен з них містить дані про дві зв’язані таблиці та їх поля.

TableNode source = (TableNode)diagram.findNodeById(relation.pk\_table);

TableNode destination = (TableNode)diagram.findNodeById(relation.fk\_table);

Тепер, коли ми визначили індекси полів, ми можемо поєднати їх.Як і в випадку вище ми використовуємо Factory щоб створити компоненту .

DiagramLink link = diagram.getFactory().createDiagramLink(source, pk\_index, destination, fk\_index);

Також зробими деякі візуальні налаштування, щоб зробити лінії зв'язків зрозумілішими.

link.setBaseShape(ArrowHeads.RevWithLine);

link.setBaseShapeSize(3f);

link.setHeadShapeSize(3f);

link.setShape(LinkShape.Cascading);

Для тих рядків, які є зовнішніми ключами, але не є первинними ключами додамо значок:

try {

File pathToFile = new File("res/fkey.png");

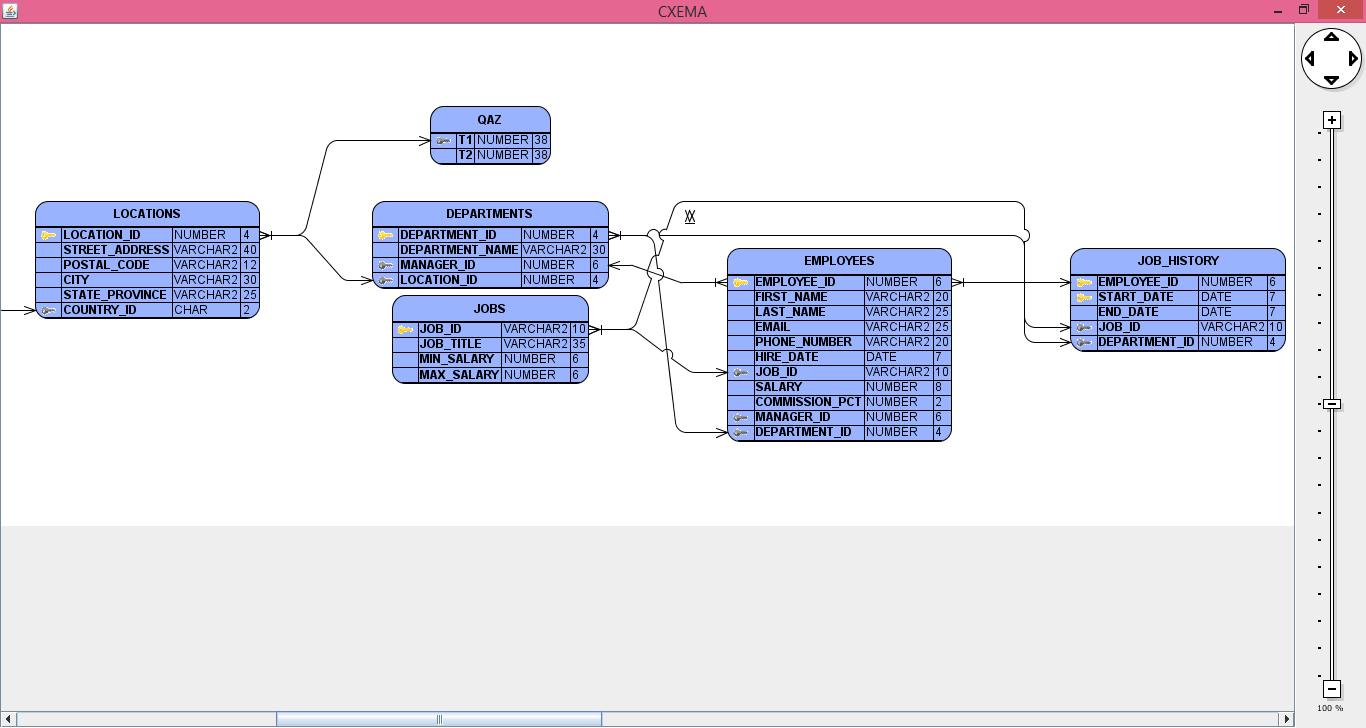
Image image = ImageIO.read(pathToFile);

if(fk\_index &gt; -1 &amp;&amp; destination.getCell(0,fk\_index).getImage() == null)

destination.getCell(0,fk\_index).setImage(image);

} catch (IOException ex) {

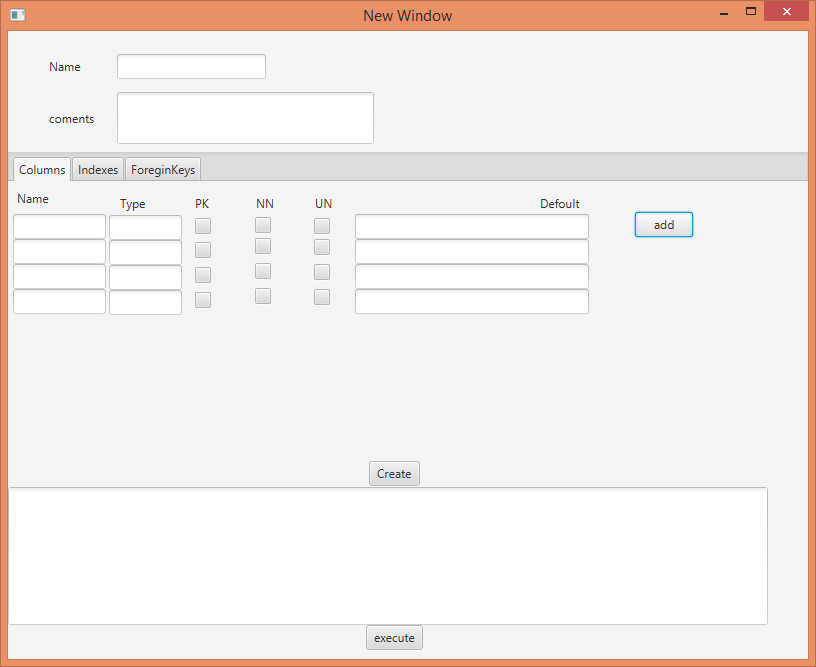
ex.printStackTrace();}



**Створення піддодатку “кноструктор таблиць”**

**Конструктор таблицць має виконувати три підзадачі**

* **створювати колонки**
* **зовнішні ключі**
* **індекси**
* **моджливісь редагування згенерованої SQL команди**



Задопомогою Scene Builder створимо нове вікно де головну роль буде виконувати TabPane .В TabPane додамо три вкладки які будуть відповідати за генерацію колонок , індексів,та зовнішніх ключів.

Генерація колонок кожна колонка меє мати назву,тип і необов’язкові параметри як ключове поле,не нульове поле,унікальне,та дефолтні значення.

Для диномічної генерації нових колонок будемо використовувати VBox .

Коли користувач тисне кнопку add викликаєтся метод який додає до VBox нові компоненти

**private void** add() {  
 **k**++;  
 **name**.getChildren().add(**new** TextField());  
 **type**.getChildren().add(**new** TextField());  
 **pk**.getChildren().add(get());  
 **nn**.getChildren().add(get());  
 **un**.getChildren().add(get());  
**defoult**.getChildren().add(**new** TextField());  
 }

Компоненту CheckBox будемо створювати таким чином щоб її висота була однакова з іншими компонентами.

**private** CheckBox get() {  
 CheckBox c = **new** CheckBox();  
 c.setMinSize(25, 25);

**return** c;

Генерування SQL проходить наступними чином проходимо по компонентах VBox

І формуємо команду для створення колонок.

String query = **"CREATE TABLE "** + q[0] + **"."** + tablename + **"("**;  
**for** (**int** i = 0; i < **k**; i++) {  
 query += **" "** + ((TextField) **name**.getChildren().get(i)).getText();  
 query += **" "** + ((TextField) **type**.getChildren().get(i)).getText();  
 CheckBox c1 = (CheckBox) **pk**.getChildren().get(i);  
 **if** (c1.isSelected())  
 query += **" PRIMARY KEY "**;  
 c1 = (CheckBox) **un**.getChildren().get(i);  
 **if** (c1.isSelected())  
 query += **" UNIQUE "**;  
 **if** (((TextField) **defoult**.getChildren().get(i)).getText() != **""** && ((TextField) **defoult**.getChildren().get(i)).getText().length() >= 1)  
 query += **" DEFAULT "** + ((TextField) **defoult**.getChildren().get(i)).getText();  
 c1 = (CheckBox) **nn**.getChildren().get(i);  
 **if** (c1.isSelected())  
 query += **" NOT NULL "**;  
 **if** (i != **k** - 1)  
 query += **","**;  
}

Генерація зовнішніх ключів

Коли користувач перехдить на вкладку створення зовнішніх ключів автоматично викликаєтся метод який ініціалізує ChoiseBox .

Всього на формі 5 ChoiseBox три генеруются спочатку.

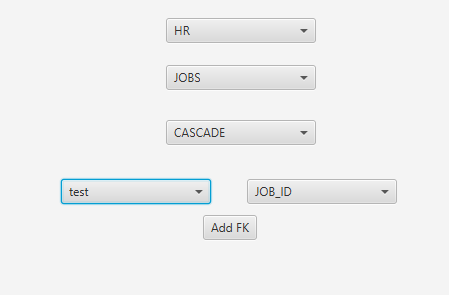
Список баз даних

ObservableList<String> items = FXCollections.*observableArrayList*(**dbUtils**.getAllDB());  
**choiseDB**.setItems(items);

Тип звязку пи зміні залежного рядка.  
ArrayList<String> cases = **new** ArrayList<>();  
cases.add(**"No action"**);  
cases.add(**"CASCADE"**);  
cases.add(**"SET NULL"**);

Список колонок в поточній таблиці.  
ObservableList<String> items2 = FXCollections.*observableArrayList*(myTab);  
**myTables**.setItems(items2);

Обирання зовнішньої таблиці виклакає метод який отримує список колонок в цій таблиці і заповнює список зовнішніх колонок.



Після цього користувач потрібно тільки обрати потрібні колонки і натиснути

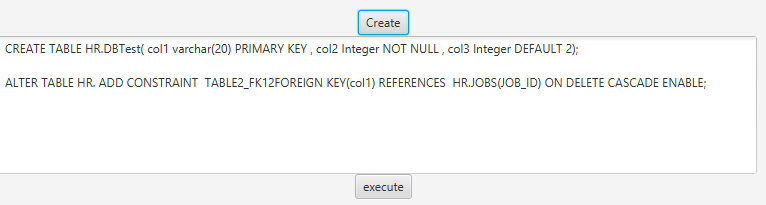
Add FK.

Створюєтся запит на зміну таблиці в якоми ми додаємо **CONSTRAINT з зовнішнім ключем .**  
String query = **"ALTER TABLE "** + q[0] + **"."** + tablename;  
query += **" ADD CONSTRAINT "** + **" TABLE2\_FK1"** + **TableKONSTRAINTS** + **"FOREIGN KEY("**;  
query += (String) **myTables**.getSelectionModel().getSelectedItem() + **") REFERENCES**

**Створення індексів таксамо як і в пеоередніх випадках користувачу надаются поля для введеня назви індекса,типу індекса(Unique BitMap Domain) таксамо динамічно додаются поля для додаваня колонок до певного індекса**

**public void** createINdex(){  
 String query=**"CREATE INDEX "**;  
 query+=IndexName+**"."**+**indexNAme**.getText();  
 *query+=****"ON"****+DbTAble;* query+=**"("**;  
 **for** (**int** i = 0; i <**r** ; i++) {  
  
 **qury** += **" "** + ((ChoiceBox) **indexColumn**.getChildren().get(i));  
*// String indexTyPE=((ChoiceBox) indexType.getChildren().get(i));* **if**(!indexTyPE.isEmpty()||!indexTyPE.equals(**null**))  
 **qury** += **" "** + ((TextField) **indexType**.getChildren().get(i)).getText();  
 **if**(i!=**r**-1){  
 query+=**","**;  
 }  
 }  
 query+=**");"**;  
 **IndexString**+=query;  
  
 }

Після того як користувач сформував обов’язково колонки ті індекси з зовнішніми ключами програма формує загальний SQL запит.



Виконання згенерованої SQL команди

**Відкриваємо підключення до бази даних**   
 DBConnection testConnection = **new** DBConnection();  
 Connection connection = testConnection.*getConnect*();  
 **try** {  
 String comands[] = **textAreaOUTPUT**.getText().split(**";"**);  
 Statement statement = connection.createStatement();

Почерзі виконуємо команди  
 **for** (String s : comands) {  
 System.***out***.println(s);  
 statement.execute(s);  
 }  
*якщо всі команди були успішно виконані підтвержуємо Транзакцію* statement.execute(**"commit"**);  
 } **catch** (Exception e) {

При помилці виконанння SQL запиту користувач побачить вікно з текстом помилки   
 Alert alert = **new** Alert(Alert.AlertType.***INFORMATION***);  
  
 alert.setTitle(**"Error"**);  
 alert.setHeaderText(**null**);  
 alert.setContentText(e.getMessage());  
  
 alert.showAndWait();  
 e.printStackTrace();  
 }  
 }



Вик література

http://it-ua.info/news/2014/03/17/patterni-dlya-novichkv-mvc-vs-mvp-vs-mvvm.html