**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет   
імені Петра Могили**

**Факультет комп’ютерних наук**

**Кафедра інженерії програмного забезпечення**

**КУРСОВА РОБОТА**

з дисципліни «Організація баз даних»

на тему: **«Створення бази даних по настольній рольовій грі Dungeons&Dragons»**

121 – КР.ПЗ.01 – 208.1710824

**Виконав: студент 2 курсу, групи 208**

**спеціальності 121 «Інженерія програмного забезпечення»**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Удовиченко О.А.

(підпис, ініціали та прізвище)

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

(дата)

**Керівник: ст. викладач кафедри ІПЗ,   
Фаленкова М. В.**

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_\_

Члени комісії

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

(підпис) (ініціали та прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (ініціали та прізвище)

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

(підпис) (ініціали та прізвище)

**м. Миколаїв – 2019 рік**

**Зміст**

[ВСТУП 3](#_Toc10150904)

[Опис предметної сфери та постановка задачі 4](#_Toc10150905)

[Проектування БД – логічна модель даних 5](#_Toc10150906)

[Проектування БД – фізична модель бази даних 10](#_Toc10150907)

[Опис клієнтського застосунку для роботи із даними ДБ 13](#_Toc10150908)

[Виснувок 24](#_Toc10150909)

[Список використаних джерел 25](#_Toc10150910)

[Додатки 26](#_Toc10150911)

[Анімація 26](#_Toc10150912)

[Sample 26](#_Toc10150913)

[Class 26](#_Toc10150914)

[Race 27](#_Toc10150915)

[Species 27](#_Toc10150916)

[Level 28](#_Toc10150917)

[Cantrips 29](#_Toc10150918)

[Spells 30](#_Toc10150919)

[Characters 31](#_Toc10150920)

[Перевірка на плагіат 32](#_Toc10150921)

# ВСТУП

Система управління базами данних (СУБД) – це комплекс мовних засобів і програм, необхідних для створення баз данних.

СУБД повинна включати в свій склад засоби, що забезпечують потреби користувачів різних категорій на всіх етапах життєвого циклу систем БД: проектування, створення, експлуатації.

Системи управління базою даних дозволяють вирішувати цілий комплекс завдань, серед яких найбільш важливими є:

1. зберігання інформації

2. швидкий пошук інформації за ознаками

3. систематизація інформації

4. обробка інформації

5. синтез нової інформації на підставі інформації бази.

База даних – сукупність даних, набір певної інформації, організованої відповідно до концепції, що в свою чергу описує характеристику цих даних і взаємозв’язки між елементами бази.

SQLite – це полегшена реліаційна система керування базами данних код якої може використовуватись безплатно та у будь яких цілях. Вона втілена у вигляді бібліотеки та у своїй більшості реалізована за стандартом SQL-92. Ця база данних широко застосовується у програмній розробці а також у розробці мобільних додатків.

У данному курсовому проекті за предметну область взято базу данних, що буде за допомогою програмного коду на мові Java зберігати інформацію про створенних персонідів настільної ролівої гри Dungeons&Dragons(D&D), а також містити у собі данні про іх можливі класси, раси, підраси, рівні, можливости та закляття.

# Опис предметної сфери та постановка задачі

**Задача**: розробка бази даних, на тему настільної рольової гри «Dungeons&Dragons» на основі реляційної моделі із використанням системи SQLite та створення користувацького графічного інтерфейсу на мові програмування Java та додатком JavaFX.

**Мета**: створення БД, за допомогою SQLite, що дозволяє зручно та швидко разробити персонажа настільної гри D&D 5-ї редакції, його классу, раси, підраси, рівня, можливостями та закляттями саме для обраних споживачем полів, для подальшого його запису та використання. Модель повинна давати початкове уявлення про створенного персонажа та його можливостей для спрощення подальшого створення персонажів у цій грі, а також запису нових ідей. Після визначення моделі буде створенно користувацький інтерфуйс за допомогою застосунку JavaFX та реалізованого за домомогою sql запитів на мові програмування Java.

**Предмет**:програмні засоби для створення користувацького застосунку для роботи з БД.

**Об’єкт**: створення бази даних для настільної ролевої гри «Dungeons&Dragons».

# Проектування БД – логічна модель даних

Деталогічна модель данних відображає логічні взаємозв’язки між елементами данних, їх змісту та організації.Для реалізації логічної структури бази данних потрібно розбити всю інформацію з файлів, а також визначити склад атрибудів для кожного з плагинів. Для цього реляційна база данних повинна бути нормалізованою.

Логічна модель - це загальний погляд на дані, модель даних в пе­вній предметній області без фізичних представлень. Саме такою моделлю і є модель типу «сутність-зв'язок».

Модель «сутність-зв'язок» (ER-модель) – модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. Існує ряд моделей для представлення знань, але одним з найзручніших інструментів уніфікованого представлення даних, незалежного від програмного забезпечення, що його реалізує, є модель «сутність-зв'язок».

Для того щоб побудувати модель данних необхідно для початку виділити основну суть бази, після чого встановити зв’язок.

На початку побудови діаграми сутність-зв'язок ми маємо наступний результат(рис.1)

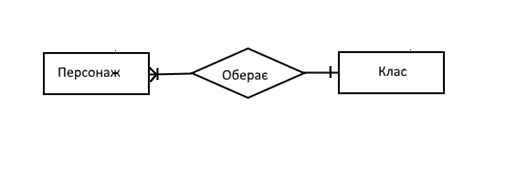


Рис.1. Початкова структура

Також необхідно додати сутності, які пвинен мати на початку кожен персонаж. Тобто расу та рівень(рис.2).

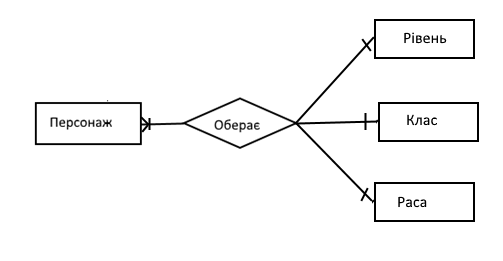


Рис.2. Розширена початкова структура

Тепер необхідно додати такі сутності як підраса, можливості накладання заклять стосовно класу та рівня, а також самі закляття які діляться на два типи: заговір та закляття(рис.3).

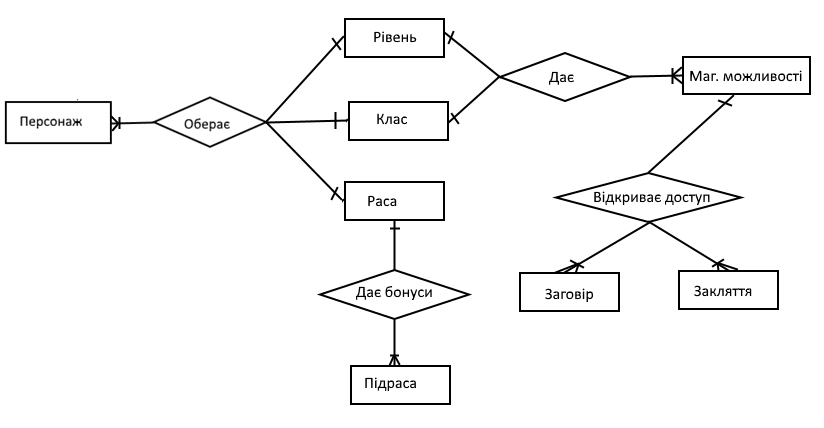


Рис.3. Структура без полів

Тепер необхідно додати в цю структуру поля. А саме у кожного персонажу наявні такі поля як і’мя, клас, раса, підраса, рівень, кількість заговорів яку він знає на його рівні, а також кількість заклять. Рівень має такі поля як сам рівень, кількість досвіду необхідного для цього рівня, а також професійний бонус, який характерний для цієї гри. У таблиці класу будуть знаходитись поля такі як назва класу, кубик його життя, його професійні навики, кидки збереження та найкраща раса для цього класу, яка в подальшому буде реалізована за допомогою запитів. У таблиці раси наявні такі поля як назва раси, бонус до характеристик характерний для неї, тривалість життя, розмір, швидкість, наявність зору у темряві, мови які вона знає а також особисті риси. Підраса в свою чергу також має особисті риси та бонус до характеристик, також в ній повині буди поля її назви та назва раси, від якої вона походить. Магічні можливості це допоміжна таблиця яка не буде відображатись користовачу, але вона існує для полагодження виборів та запитів, а також додавання інформації у інші таблиці. Вона має такі поля як назва класу, рівень, кількість заклять які персонаж може знати на обраному рівні з обраним класом, кількість заговорів та максемальний рівень заклять доступний для нього для подальших запитів. Таблиця заговорів має такі поля як назва та клас, якому вони доступні. Та заклятя назву, клас, та рівень закляття(рис.4).

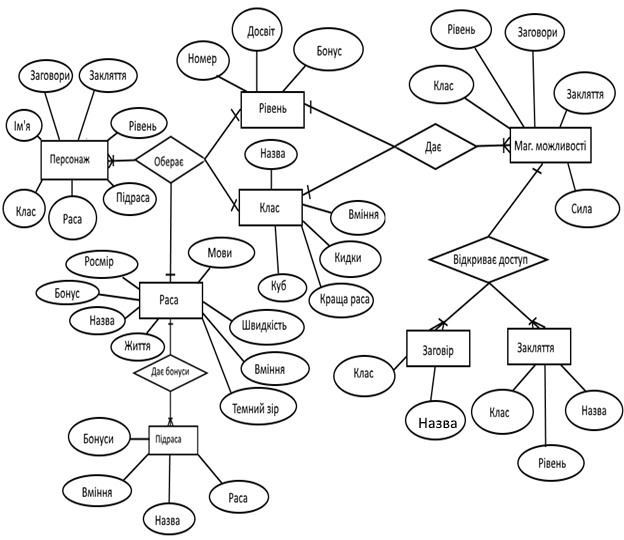


Рис.4. Кінцева IR-діаграма

Схема даних моделі виглядає наступним чином(рис.5).

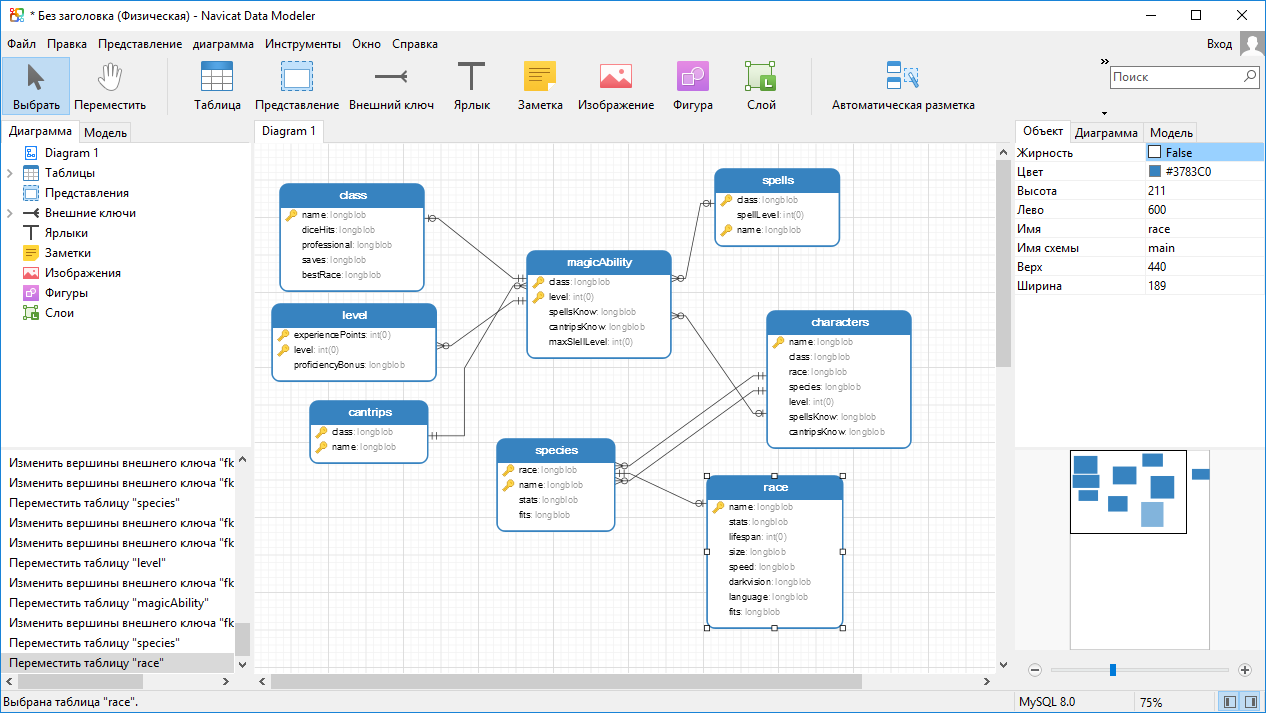


Рис.5. Схема данних

Склад бази данних:

cantrips(class, name);

characters(name, class, race, species, level, spellsKnow, cantripsKnow);

class(name, diceHits, professional, saves, bestRace);

level(experiencePoints, level, proficiencyBonus);

magicAbility(class, level, spellsKnow, cantripsKnow, maxSlellLevel);

race(name, stats, lifespan, size, speed, darkvision, language, fits);

species(race, name, stats, fits);

spells(class, spellLevel, nane);

# Проектування БД – фізична модель бази даних

Фізична модель данних це наступний етап проектування після логічної моделі. Фізична модель будується на основі логічної з врахуванням обмежень що накладаються можливостями обраної СУБД. Фізична модель має поля таблиць з їх обмеженнями, ключами та типами данних.

Отже фізична модель по моїй базі данних матиме такий вигляд (рис6,7,8,9,10,11,12).



Рис.6. Таблиця cantrips

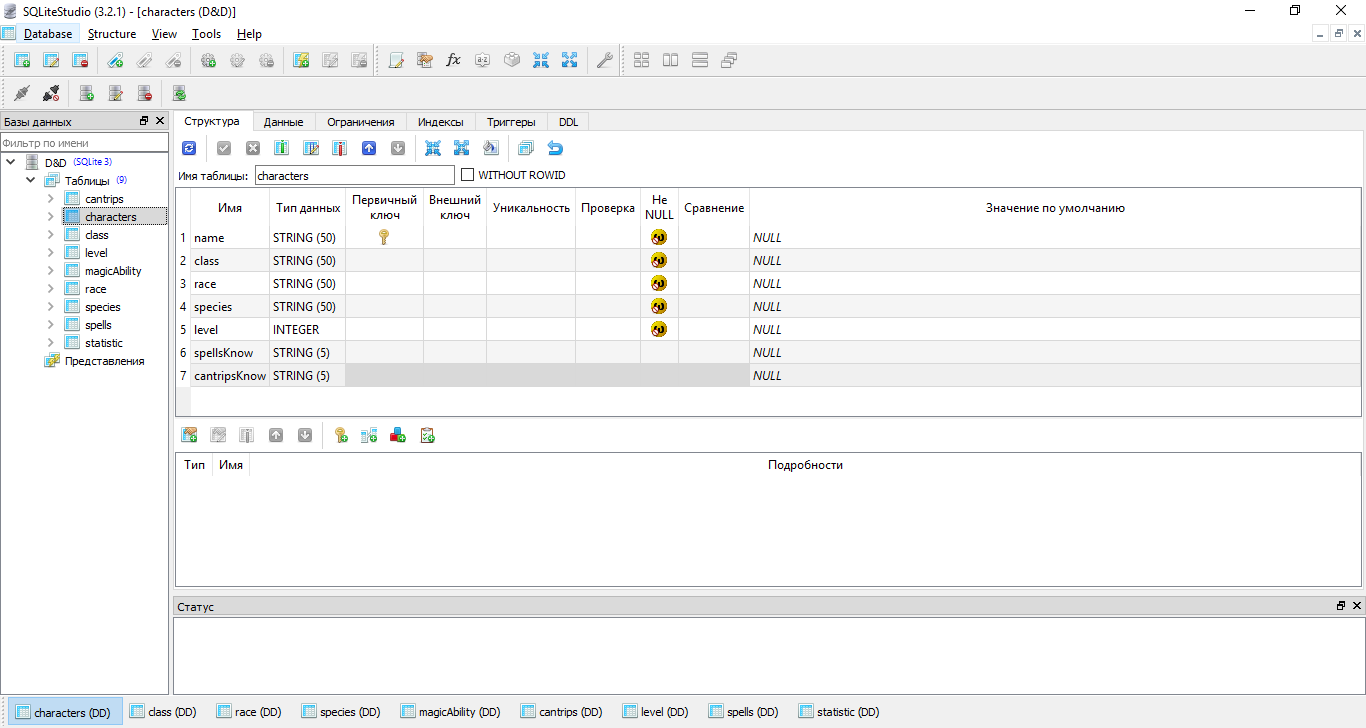


Рис.7. Таблиця characters

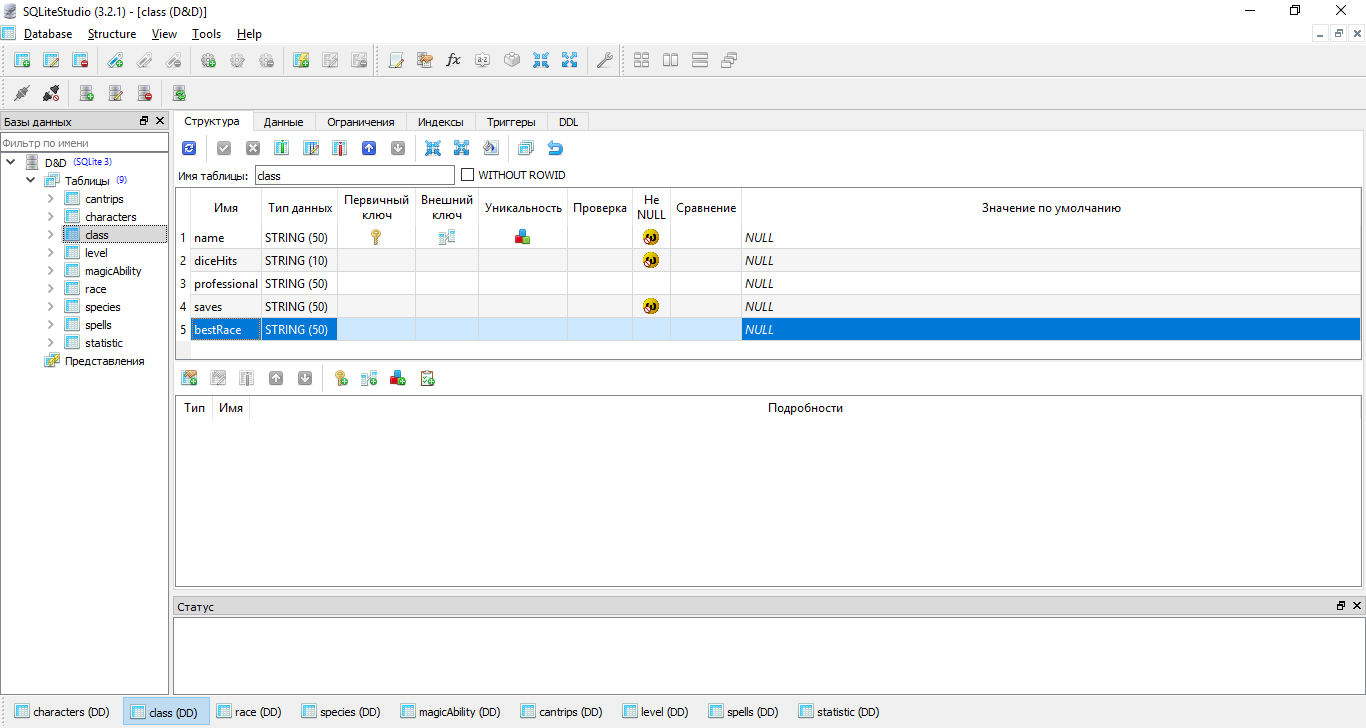


Рис.8. Таблиця class

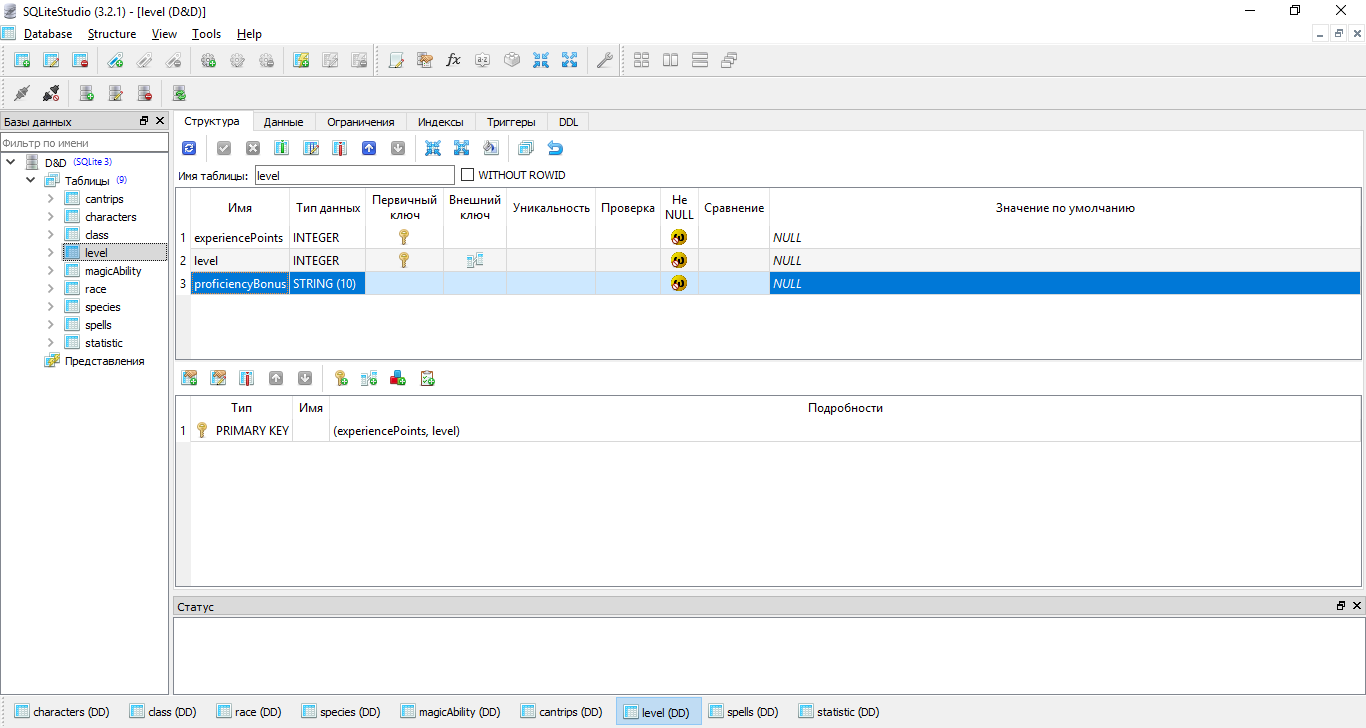


Рис.9. Таблиця level

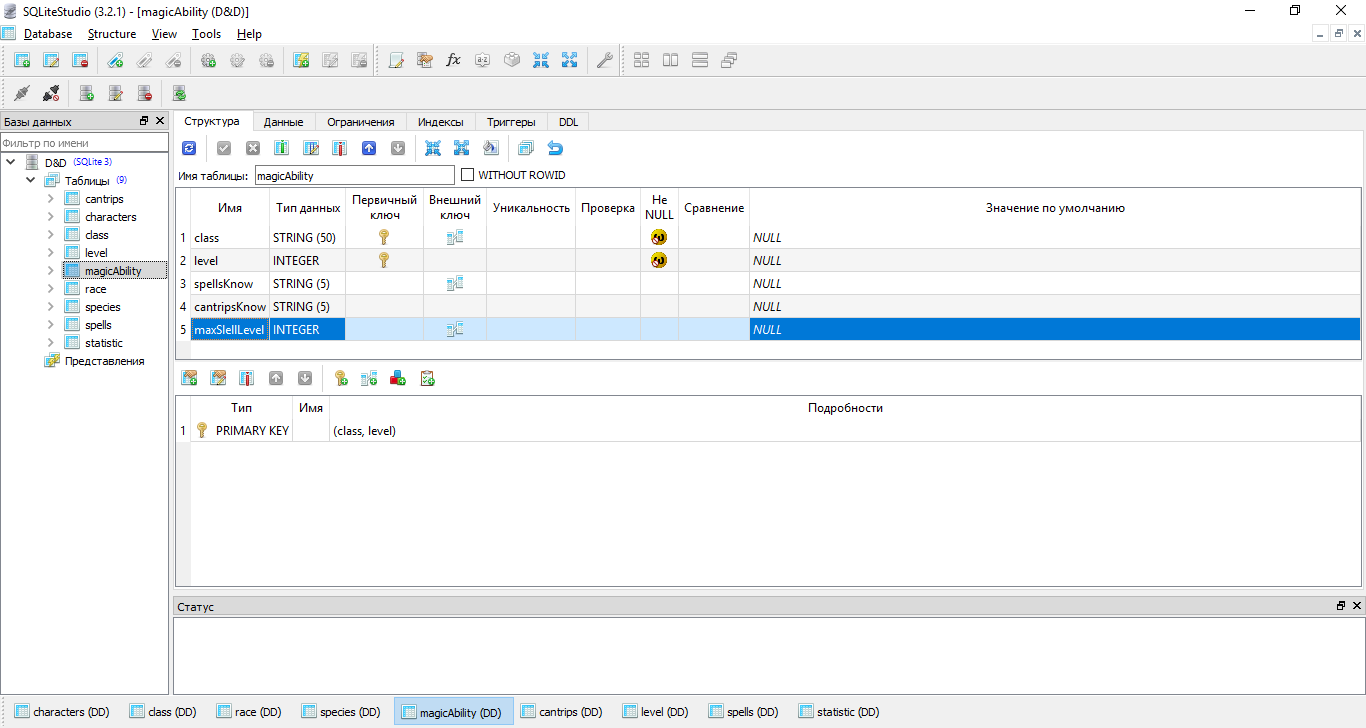


Рис.10. Таблиця magicAbility

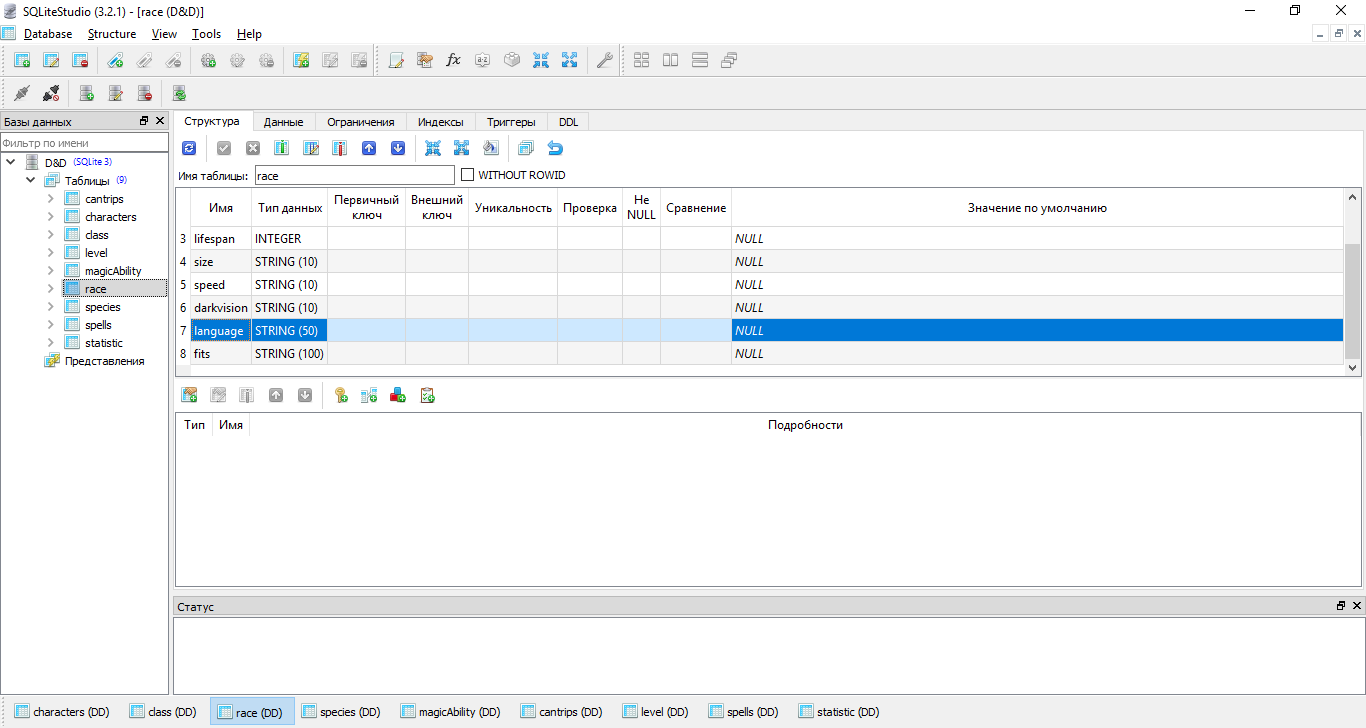


Рис.10. Таблиця race



Рис.11. Таблиця species

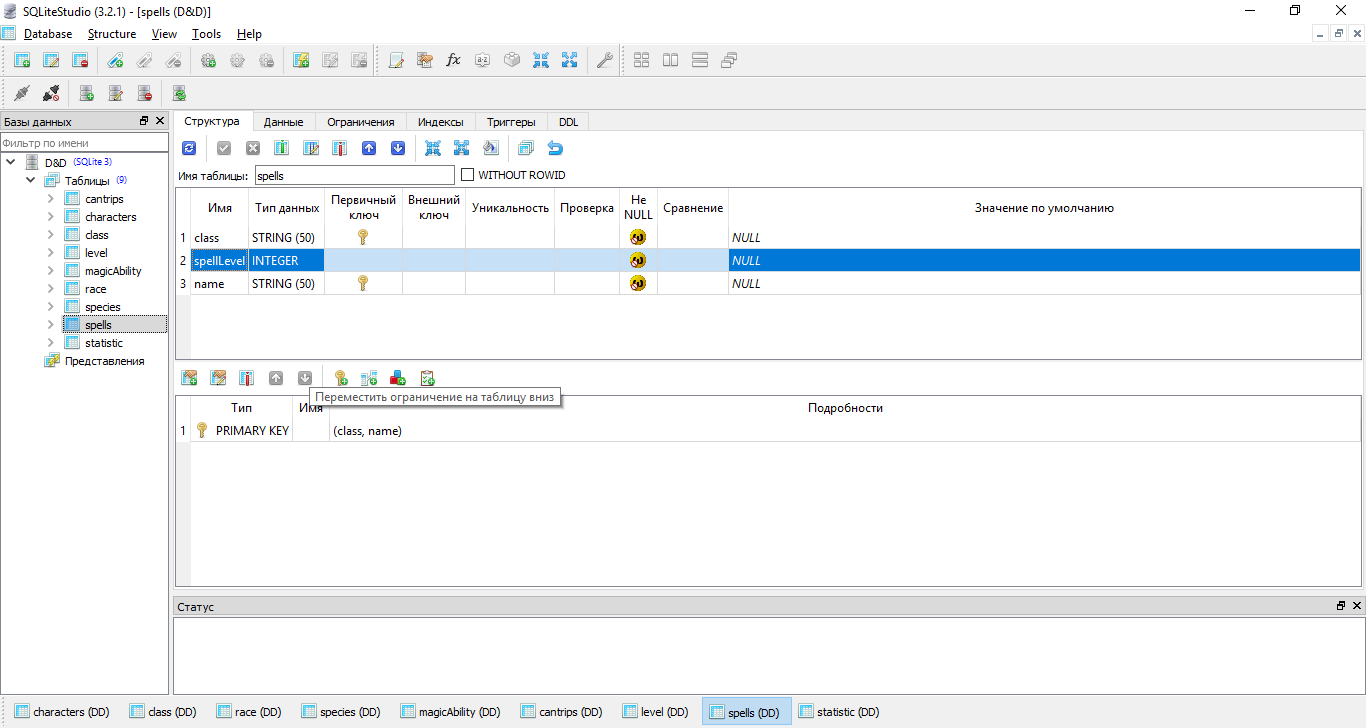


Рис.12. Таблиця spells

Також було додано таблицю statistic для майбутнього тригеру. Ця таблиця буде збирати данні про обрані класи(рис.13).

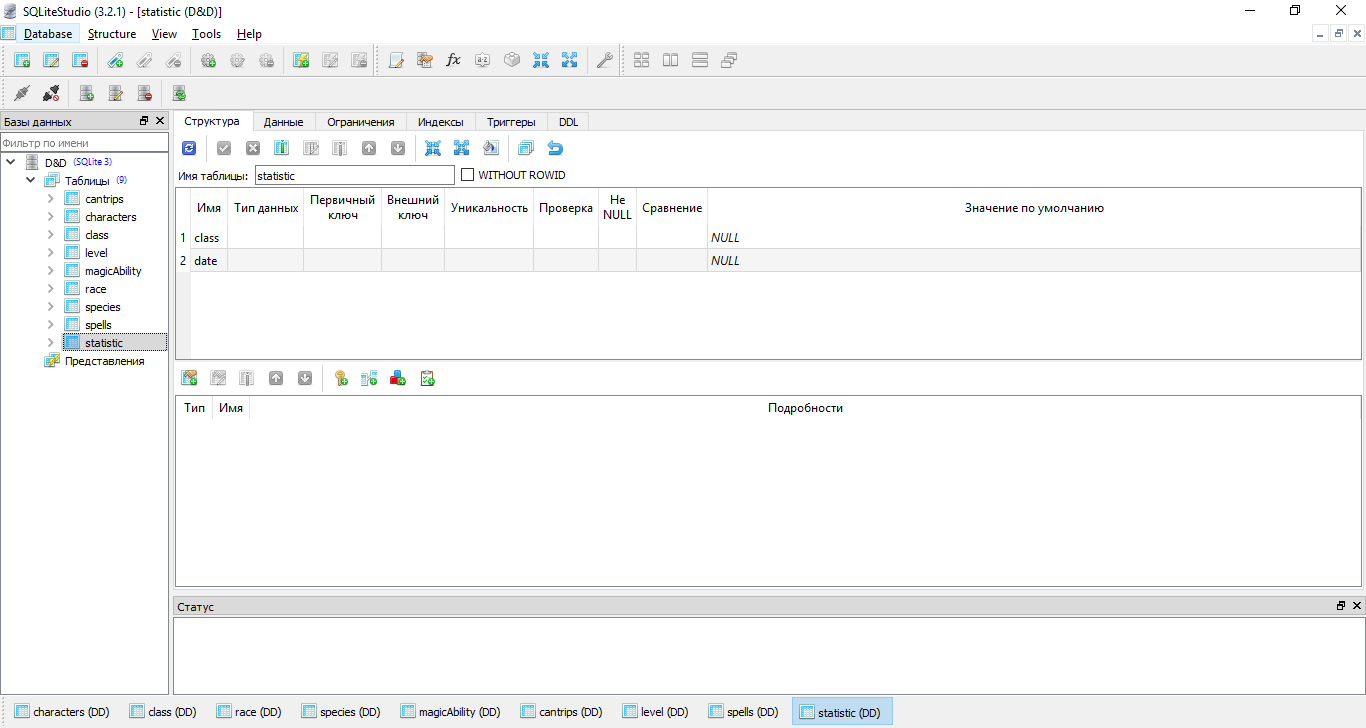


Рис.13. Таблиця statistic

# Опис клієнтського застосунку для роботи із даними ДБ

Клієнський застусунок було розроблено на мові программуваня Java у застосунку JavaFX.

Розробка застосунку складалась з декількох етапів, а саме:

1. Розроблення загального інтерфейсу;

2. З’єднання сцен сторінок послідовно між один одним;

3. Відображення таблиць з бази данних на екрані користувача;

4. Додавання sql запитів для більш зручної роботи з базою;

5. Створення загальної логіки програми.

Майже на кожну таблицю було зроблено окрему сцену за для забеспечення зручності у використанні додатку.

Суть програми полягає в тому щоб користувач міг додавати своїх нових персонажів у окрему таблицю та йому відображались таблиці з бази данних згідно його попередніх виборів. Всі пошуки та сортування автоматизовані залежно від того що саме вводить користувач. Тобто йому не доведеться шукати закляття доступні для нього адже вони відобразяться автоматично згідно його введенному класу, якщо у класа є така можливість.

Інтерфейс було зроблено за допомогою додатку SceneBuilder.

Першим що користувач бачить при запуску програми – вітальну сторінку, у якій одразуж пропонується ввести ім’я нового персонажу за для того щоб потім додати його у таблицю персонажів. А так ож кнопка переходу до списку вже готових персонажів(рис.14).

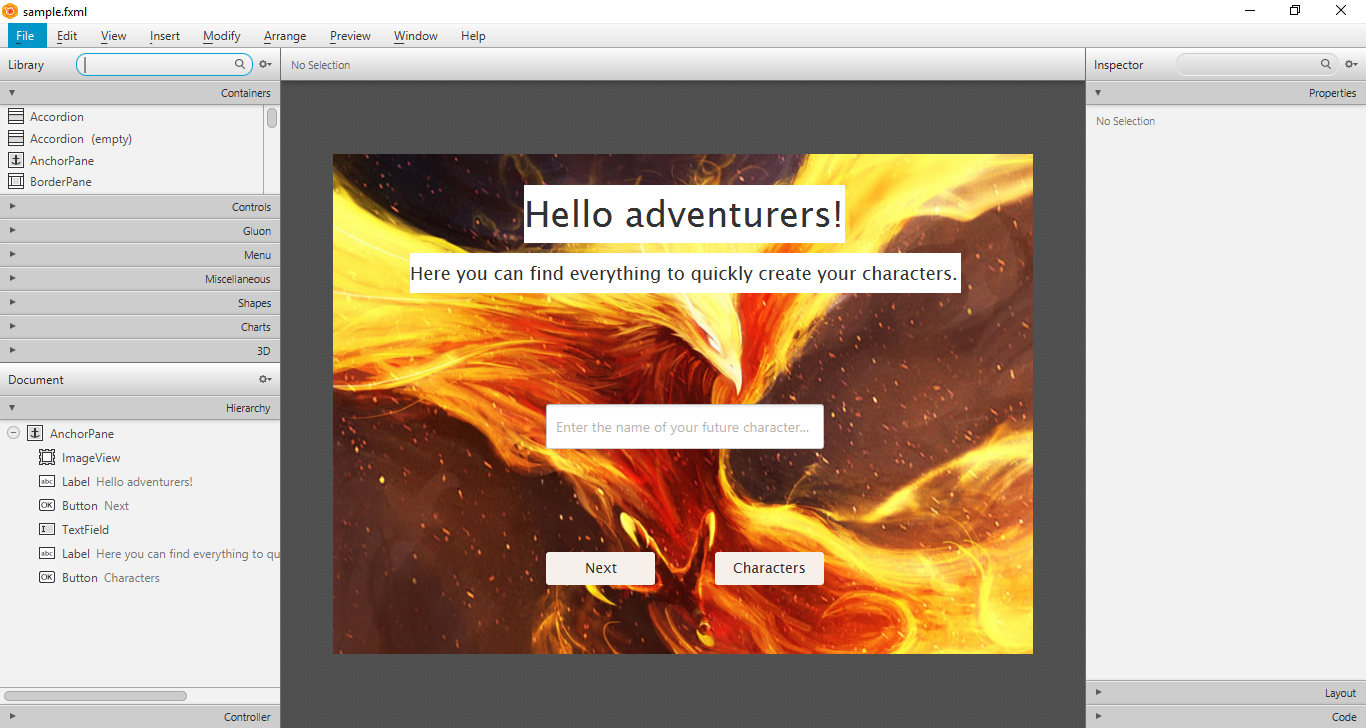


Рис.14. Вітальна сторінка

Після введення нового імені програма переміщує користувача на нову сцену, де йому пропонується обрати клас свого нового персонажу. На більшу частину сцени розміщенна таблиця з переліком можливих класів, з яких користувач може обрати будь-який довподоби. Окрім переліку классів є можливість побачити необхідну для вибору інформацію про кожного з них. Також є кнопка повернення назад, як що користувач сумніваеться у обраному імені, хоча потім його можно буде змінити.

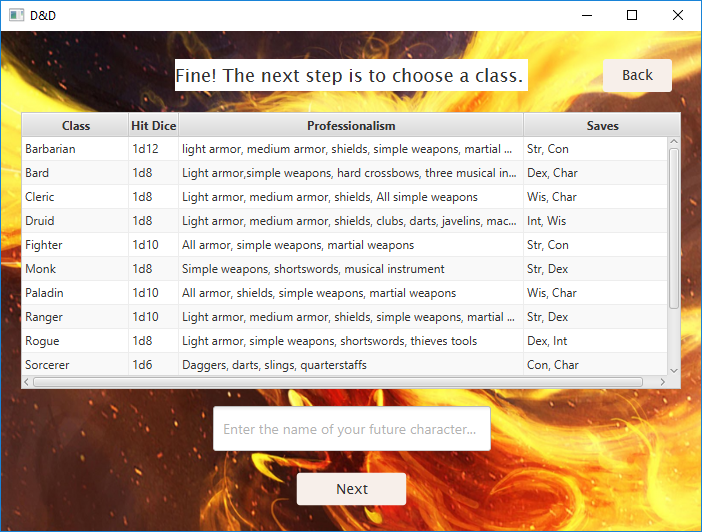


Рис.15. Сцена вибору класу

Після того як користувач обрав клас відкривається сцена на якій він може обрати свою майбутню расу. Вона складаеться з двох таблиць, поля вводу даних, кнопки продовжити та кнопки повернення на попередню сторінку. У першій таблиці за допомогою запиту відображається одна раса, яка найбільш підходить до обраного користувачем классу. Також на випадок як що він не хоче брати саме її друга таблиця виводить всі можливі класси, які він може обрати(рис.16).

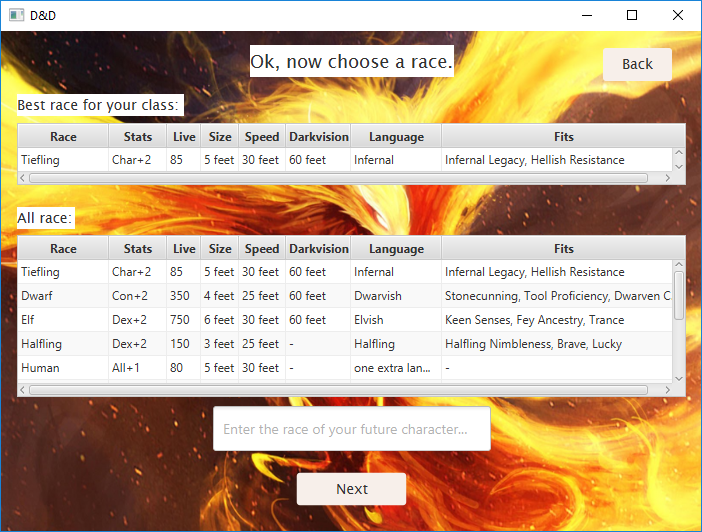


Рис.15. Сцена вибору раси

У всесвіті цієї настільної гри у деяких рас існують підраси. Вони дають додаткові бонуси до рас, а іноді обмеження (наприклад як з нічними ельфами), у рас з підрассами існує свій унікальний список підрас. Саме тому вони відображаються у таблиці за допомогою запиту стосовно минулих виборів користувача(рис.16).

Якщо було обрано расу у якої неіснує підраси користувач одразу перейде на наступну сцену з введенням початкового рівня(рис.17).

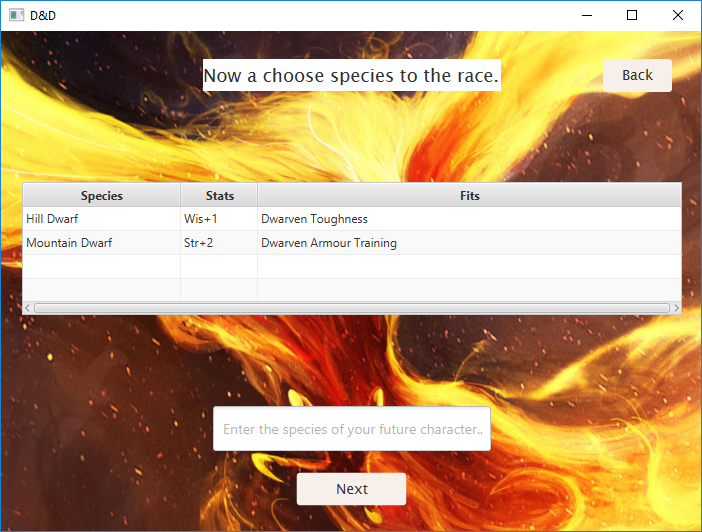


Рис.16. Сцена вибору підраси

При введені початкового рівня застосовується допоміжна база з магічними властивостями, яка не відображається користувачу. Вона потрібна для того щоб дізнатись чи вміє обраний користувачем клас накладати закляття, якщо так то с якого рівня і чи доступно це для нього зараз.

Кожен клас має свою індивідуальну механіку отримання заклять чи заговорів. У деяких з них вони збігаються, деяки не можуть чаклувати зовсім, деякі з окремих рівней. І кожен з них на окремих рівнях може вивчити окрему кількість заклять чи заговорів окремого рівня саме заклять. Самі ці данні знаходяться у допоміжній базі.

Після вводу користувачем свого рівня програма відкриває додаткову базу та перевіряє чи може цей клас на цьому ревні накладати заговори. Якщо так то він переходить до таблиці заговорів саме для його классу, адже не всі з них доступні всім. Якщо цей клас не може накладати заговори, але знає зккляття (наприклад паладін або мисливець) відкривається сцена с закляттями обраними за допомогою запитів доступним саме для обраного класу і в залежності від його рівня відображаються відповідного рівня закляття. Інакше якщо обраний клас не може накладати хочаб якісь закляття користувач одразу переходить на список персонажів з вже доданим його класом.

Саме на сторінці з вибором початкового рівня після заповнення поля створюються всі нові персонажі. З додаткової таблиці знаходяться всі невистачаючі данні (такі як кількість заговорів та кількість заклять які знає персонаж) та створюється нове поле.

Запит додавання нового персонажу:

String insert = "insert into characters values('" + Main.*nameCharacter* + "','" + Main.*classCharacter* +  
 "','" + Main.*raceCharacter* + "','" + Main.*speciesCharacter* + "','" + Main.*levelCharacter* +  
 "','" + Main.*spellsKnowCharacter* + "','" + Main.*cantripsKnowCharacter* + "');";

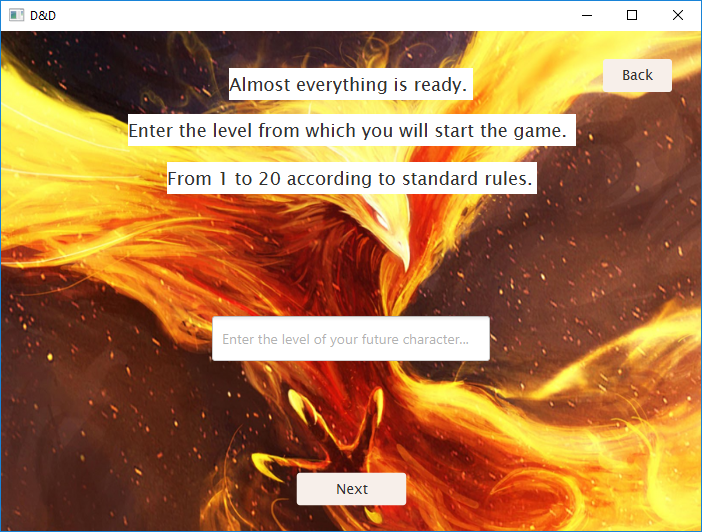


Рис.17. Сцена вибору початкового рівня

Після рівня відкривається сцена у якій відображаються всі заговори, які може знати новий персонаж, створений на менулій сцені, ри умові що він знає хочаб один заговір. Інакше програма пропустить цей крок (рис.18).

Також тут відсутня кнопка повернення оскільки всі зміни було збережено та додано новий персонаж. Після перегляду заговорів відкривається сцена перегляду можливих заклять.

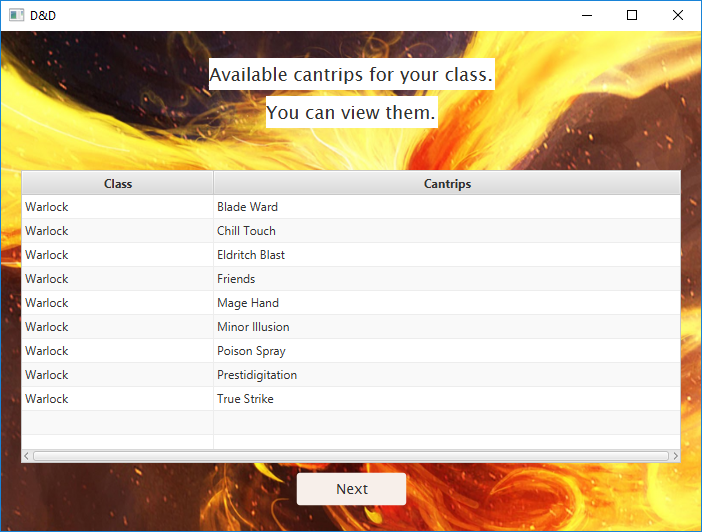


Рис.18. Сцена заговорів

Наступна сцена за допомогою запиту відображає ті закляття до яких новостворенний персонаж має доступ з обмеженням на його клас та рівень (рис.19). Обгрунтування всіх заклять та заговорів можно знайти у книжці гравця по D&D 5-ї редакції.

Тут також відсутня кнопка повернення.

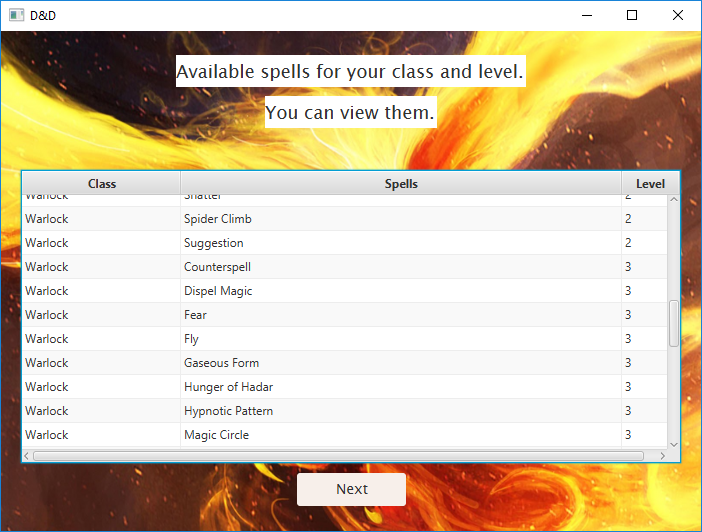


Рис.19. Сцена заклять

Після виконання всього відкривається остання сцена у якій відображені всі персонажі, яких створив користувач. Ця сцена складається з таблиці, декількох полев вводу тексту та кнопки. Поля вводу потрібні для зміни доступних для зміни користувачу данних у таблиці, також видалення вже непотрібних персонажів(рис.20).

Код видалення поля:

if(Main.*con*.createStatement().executeQuery("select name from characters where name = '" + delete\_name.getText()  
 + "'").next()) {  
 String delete = "delete from characters where name = '" + delete\_name.getText() + "'";  
 try {  
 PreparedStatement prSt = Main.*con*.prepareStatement(delete);  
 prSt.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/characters.fxml");

Код зміни іменю персонажа:

if(Main.*con*.createStatement().executeQuery("select name from characters where name = '" + old\_name.getText()  
 + "'").next() && !new\_name.getText().equals("")) {  
  
 String update = "update characters set name = '" + new\_name.getText() + "' where name = '" + old\_name.getText() + "'";  
 try {  
 PreparedStatement prSt = Main.*con*.prepareStatement(update);  
 prSt.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/characters.fxml");  
}

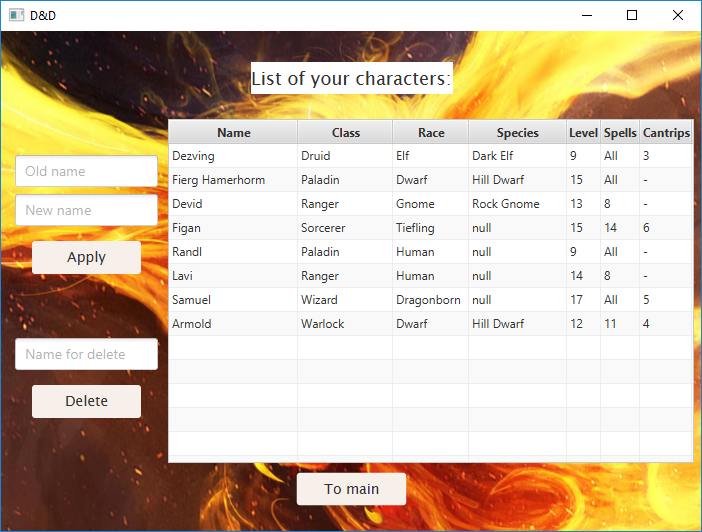


Рис.20. Сцена персонажів

Також було додано невелику кількість анімацій для того щоб показати користувачу де саме він помилився, або не заповнив поля. За для цього було створено окремий невеликий клас з анімацією, яка трясе незаповнену, або невірно заповнену частину сцени.

На прикінці було додано декілька триггерів, які забороняють користуачу змінювати ті данні, які будуть повністю суперечать правилам цієї гри. А токож тригер який збирає данні про сикористані класи для подальшого удосконалення та використання у додатку з базою данних(рис.21).

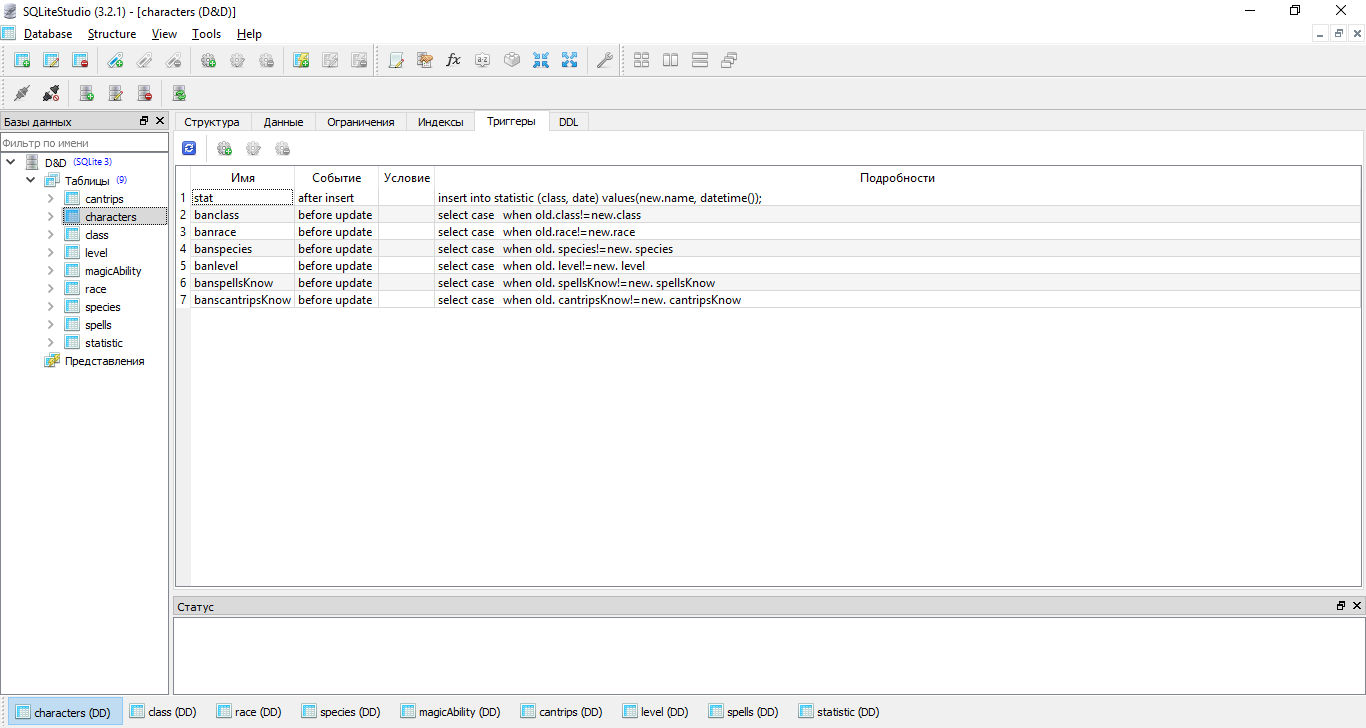


Рис.21. Триггери

# Виснувок

У ході цієї курсової роботи було створено базу данних с користувацьким інтерфейсом на тему «Dungeons&Dragons» 5-ї редакції, яка полегше створення нових персонажів та значно заощадить час нових гравців, які захочуть вступити у цю сферу. Існує досить багато додатків по цій грі, які дозволяють заповнити лист персонажа, передивлятись свій інвентар, читати про заклятя чи дивитися монстрів, але ще немає бази яка б допомагала новим гравцям створити свого персонажа за декілька хвилин замість того аби читати декілька сотен сторінок з правилами у яких вони не знажуть одразу орієнтуватися. Ця база данних направляна саме на таких користувачів. Можливо після подібних додатків у нашій країні ця гра і стане настільки ж популярною як у англомовних країнах, адже найчастіше нових гравців лякає саме велика абілія правил, в яких самостійно розбиратись дуже довго.

Для побудови цієї бази були досліджені бази переліковані вишче, та взяття з них необхідного матеріалу, котрий може бути дуже корисний для нових гравців, та не дуже багато щоб не відпугнути їх складністтю розуміння.

Вся паза данних а продовж виконання программи супроводжується мовою програмування Java та розроблена за допомогою JavaFx у середовищі Intellij IDEA та сцени зроблено у додатку SceneBuilder за для супроводження користувача атмосферним графічним інтерфейсом.

База даних була розроблена з використанням реліаційної системи SQLite та SQLiteStudio.

# Список використаних джерел

1. <https://studall.org>
2. <https://ru.wikipedia.org/wiki/Dungeons_%26_Dragons>
3. <https://upread.ru/art.php?id=3>
4. <http://qaru.site/questions/1475730/image-animation-in-java-swing-on-click>
5. <https://habr.com/ru/post/149356/>
6. <https://habr.com/ru/sandbox/88039/>
7. <https://www.tutorialspoint.com/sqlite/sqlite_java.htm>
8. <http://qaru.site/questions/134607/how-to-connect-sqlite-with-java>
9. <https://devcolibri.com/java-sqlite-orm/>
10. <https://github.com/xerial/sqlite-jdbc#usage>
11. <https://toster.ru/q/384697>
12. <http://qaru.site/questions/20660/java-and-sqlite>
13. <https://startandroid.ru/ru/uroki/vse-uroki-spiskom/74-urok-34-hranenie-dannyh-sqlite.html>
14. <http://www.ch-werner.de/javasqlite/overview-summary.html>
15. <http://www.sqlitetutorial.net/sqlite-java/>

# Додатки

## Анімація

public Shake(Node node){  
 tt = new TranslateTransition(Duration.*millis*(120), node);  
 tt.setFromY(0f);  
 tt.setByY(10f);  
 tt.setCycleCount(5);  
 tt.setAutoReverse(true);  
}

## Sample

void butCharacters(ActionEvent event) throws IOException {  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/characters.fxml");  
}  
  
@FXML  
void butEnterName(ActionEvent event) throws IOException {  
 if(nameCharacters.getText().equals("")){  
 Shake errorAnim = new Shake(nameCharacters);  
 errorAnim.playAnin();  
 }  
 else{  
 Main.*nameCharacter* = nameCharacters.getText();  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/class.fxml");  
 }  
}

## Class

@FXML  
void butBackToName(ActionEvent event) throws IOException {  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/sample.fxml");  
}  
  
@FXML  
void butEnterClass(ActionEvent event) throws IOException {  
  
 String cC = classCharacter.getText();  
  
 try{  
 if(Main.*con*.createStatement().executeQuery("select name from class where name = '" + cC + "'").next()){  
 Main.*classCharacter* = cC;  
  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/race.fxml");  
 }  
 else{  
 Shake errorAnim = new Shake(classCharacter);  
 errorAnim.playAnin();  
 }  
  
 }  
 catch (SQLException e){  
 e.printStackTrace();}  
}

## Race

@FXML  
void butBackToClass(ActionEvent event)throws IOException {  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/class.fxml");  
}  
  
@FXML  
void butEnterRace(ActionEvent event) throws IOException {  
 String rC = raceCharacter.getText();  
  
 try{  
 if(Main.*con*.createStatement().executeQuery("select name from race where name = '" + rC + "'").next()){  
 Main.*raceCharacter* = rC;  
 NewScene n = new NewScene();  
  
 if (Main.*con*.createStatement().executeQuery("select race from species where race = '" + Main.*raceCharacter* + "'").next()) {  
 n.newScene(event, "/fxml/species.fxml");  
 }  
 else{  
 n.newScene(event, "/fxml/level.fxml");  
 }  
 }  
 else{  
 Shake errorAnim = new Shake(raceCharacter);  
 errorAnim.playAnin();  
 }  
 }  
 catch (SQLException e){  
 e.printStackTrace();}  
}

## Species

void butBackToRace(ActionEvent event) throws IOException {  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/race.fxml");  
}  
  
@FXML  
void butEnterSpecies(ActionEvent event) throws IOException {  
 String sC = speciesCharacter.getText();  
 try{  
 if(Main.*con*.createStatement().executeQuery("select name from species where name = '" + sC + "'").next()){  
 Main.*speciesCharacter* = sC;  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/level.fxml");  
 }  
 else{  
 Shake errorAnim = new Shake(speciesCharacter);  
 errorAnim.playAnin();  
 }  
 }  
 catch (SQLException e){  
 e.printStackTrace();}  
}

## Level

@FXML  
void butBackToSpecies(ActionEvent event)throws IOException {  
 try {  
 NewScene n = new NewScene();  
 if (Main.*con*.createStatement().executeQuery("select race from species where race = '" + Main.*raceCharacter* + "'").next()) {  
 n.newScene(event, "/fxml/species.fxml");  
 }  
 else {  
 n.newScene(event, "/fxml/race.fxml");  
 }  
 }  
 catch (SQLException e){  
 e.printStackTrace();}  
}  
  
@FXML  
void butEnterLevel(ActionEvent event)throws IOException {  
  
 try{  
 Integer level = Integer.*parseInt*(levelCharacter.getText());  
 if(level >= 1 && level <= 20){  
 Main.*levelCharacter* = level.toString();  
  
 ResultSet rsCantrips = Main.*con*.createStatement().executeQuery("select cantripsKnow from magicAbility " +  
 "where class = '" + Main.*classCharacter* + "' and level = '" + Main.*levelCharacter* + "'");  
 ResultSet rs = Main.*con*.createStatement().executeQuery("select spellsKnow from magicAbility " +  
 "where class = '" + Main.*classCharacter* + "' and level = '" + Main.*levelCharacter* + "'");  
  
 Main.*cantripsKnowCharacter* = rsCantrips.getString(1);  
 Main.*spellsKnowCharacter* = rs.getString(1);  
  
  
 String insert = "insert into characters values('" + Main.*nameCharacter* + "','" + Main.*classCharacter* +  
 "','" + Main.*raceCharacter* + "','" + Main.*speciesCharacter* + "','" + Main.*levelCharacter* +  
 "','" + Main.*spellsKnowCharacter* + "','" + Main.*cantripsKnowCharacter* + "');";  
 try {  
 PreparedStatement prSt = Main.*con*.prepareStatement(insert);  
 prSt.executeUpdate();  
 }  
 catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 NewScene n = new NewScene();  
  
 if(rsCantrips.next() && rsCantrips.getString(1).chars().allMatch(Character::*isDigit*)) {  
 n.newScene(event, "/fxml/cantrips.fxml");  
 }  
 else if(rs.next()){  
 n.newScene(event, "/fxml/spells.fxml");  
 }  
 else{  
 n.newScene(event, "/fxml/characters.fxml");  
 }  
 }  
 else{  
 Shake errorAnim = new Shake(levelCharacter);  
 errorAnim.playAnin();  
 }  
 }  
 catch(Exception e){  
 }  
  
}

## Cantrips

void butViewCantrips(ActionEvent event) throws IOException {  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/spells.fxml");  
}  
  
ObservableList<TableCantrips> oblist = FXCollections.*observableArrayList*();  
@FXML  
void initialize() {  
 try{  
 ResultSet rs = Main.*con*.createStatement().executeQuery("select \* from cantrips " +  
 "where class = '" + Main.*classCharacter* + "'");  
  
 while (rs.next()){  
 oblist.add(new TableCantrips(rs.getString("class"), rs.getString("name")));  
 }  
 }  
 catch (SQLException e){  
 e.printStackTrace();}  
  
 tableCantrips\_colClass.setCellValueFactory(new PropertyValueFactory<>("classCh"));  
 tableCantrips\_colCantrips.setCellValueFactory(new PropertyValueFactory<>("name"));  
  
 tableCantrips.setItems(oblist);  
}

## Spells

void butViewSpells(ActionEvent event) throws IOException {  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/characters.fxml");  
}  
  
ObservableList<TableSpells> oblist = FXCollections.*observableArrayList*();  
@FXML  
void initialize() {  
 try{  
 ResultSet r = Main.*con*.createStatement().executeQuery("select maxSlellLevel from magicAbility " +  
 "where class = '" + Main.*classCharacter* + "' and level = '" + Main.*levelCharacter* + "'");  
 String maxSlellLevel = r.getString(1);  
 ResultSet rs = Main.*con*.createStatement().executeQuery("select \* from spells " +  
 "where class = '" + Main.*classCharacter* + "' and spellLevel <= " + maxSlellLevel + " order by spellLevel asc");  
  
 while (rs.next()){  
 oblist.add(new TableSpells(rs.getString("class"), rs.getString("spellLevel"),  
 rs.getString("name")));  
 }  
 }  
 catch (SQLException e){  
 e.printStackTrace();}  
  
 tableSpalls\_colClass.setCellValueFactory(new PropertyValueFactory<>("classCh"));  
 tableSpalls\_colSpalls.setCellValueFactory(new PropertyValueFactory<>("name"));  
 tableSpalls\_colLevel.setCellValueFactory(new PropertyValueFactory<>("spellLevel"));  
  
  
 tableSpells.setItems(oblist);  
}

## Characters

void butApply(ActionEvent event) throws IOException, SQLException {  
 if(Main.*con*.createStatement().executeQuery("select name from characters where name = '" + old\_name.getText()  
 + "'").next() && !new\_name.getText().equals("")) {  
  
 String update = "update characters set name = '" + new\_name.getText() + "' where name = '" + old\_name.getText() + "'";  
 try {  
 PreparedStatement prSt = Main.*con*.prepareStatement(update);  
 prSt.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/characters.fxml");  
 }  
 else{  
 Shake errorAnim = new Shake(old\_name);  
 Shake errorAnim\_1 = new Shake(new\_name);  
 errorAnim.playAnin();  
 errorAnim\_1.playAnin();  
 }  
}  
  
@FXML  
void butDelete(ActionEvent event) throws IOException, SQLException {  
 if(Main.*con*.createStatement().executeQuery("select name from characters where name = '" + delete\_name.getText()  
 + "'").next()) {  
 String delete = "delete from characters where name = '" + delete\_name.getText() + "'";  
 try {  
 PreparedStatement prSt = Main.*con*.prepareStatement(delete);  
 prSt.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/characters.fxml");  
 }  
 else{  
 Shake errorAnim = new Shake(delete\_name);  
 errorAnim.playAnin();  
 }  
  
}  
  
  
  
@FXML  
void butTo\_Main(ActionEvent event) throws IOException {  
 NewScene n = new NewScene();  
 n.newScene(event, "/fxml/sample.fxml");  
}

## Перевірка на плагіат

