СОДЕРЖАНИЕ

[Введение 3](#_Toc24538111)

[1 Аналитический обзор 4](#_Toc24538112)

[1.1 Описание предметной области 4](#_Toc24538113)

[1.2 Техническое задание 6](#_Toc24538114)

[1.2.1 Функциональные требования 6](#_Toc24538115)

[1.2.2 Минимальные системные требования 7](#_Toc24538116)

[1.2.3 Требование к документации 7](#_Toc24538117)

[1.3 Выбор средств разработки 8](#_Toc24538118)

[1.3.1 Выбор СУБД 8](#_Toc24538119)

[1.3.2 Выбор СУБД. Вывод 9](#_Toc24538120)

[1.3.3 Дополнительные средства разработки 9](#_Toc24538121)

[2 Разработка Базы Данных 11](#_Toc24538122)

[2.1 Создание структуры 11](#_Toc24538123)

[2.1.1 Начальная структура 11](#_Toc24538124)

[2.1.2 Нормализация. 11](#_Toc24538125)

[2.1.3 Связи между объектами базы данных. 16](#_Toc24538126)

[2.2 Разработка SQL запросов 17](#_Toc24538127)

[2.2.1 Создание таблиц 17](#_Toc24538128)

[2.2.2 Запросы на создание, удаление, редактирование и выборку записей 25](#_Toc24538129)

[2.3 Конечная структура базы данных 25](#_Toc24538130)

[2.4 Защита базы данных 25](#_Toc24538131)

[Использованные источники 26](#_Toc24538132)

Введение

База данных – совокупность материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью ЭВМ.

История современных баз данных начинается с 1955 года, когда появилось программируемое оборудование обработки записей. Важный этап связан с появлением в начале 1970-х реляционной модели данных.

1. Аналитический обзор
   1. Описание предметной области

Игра представляет собой Массовую многопользовательскую онлайн игру (ММО) с элементами слешера, шутера от третьего лица (TPS) и ролевых игр (RPG), выполненную в киберпанк сеттинге.

Массовые многопользовательские онлайн игры предполагают одновременное нахождение на одном сервере большого количества игроков.

Перед тем как играть, игрок регистрируется, указывая логин, пароль и почту. Так же игрок может указать никнейм, страну, изображение профиля, а также добавлять других пользователей в список друзей, что упрощает совместную игру.

Каждый игрок имеет как минимум одного персонажа для игры на сервере, каждый из которых имеет свои характеристики, умения, подобранные и экипированные предметы и полигональную сетку (Mesh) для отрисовки персонажа в самой игре. Те же характеристики и умения имеются и у неигровых персонажей, которые появляются в игре с различными целями. К характеристикам персонажей относятся: имя, максимальное здоровье, % от максимального здоровья.

Игрок может изучить любые из умений. Применяться будут только выбранные умения.

Персонажи используют различные предметы. Каждый из предметов имеет название, редкость и свой mesh. Редкость предмета определяет количество экземпляров в мире игры.

Предметы подразделяются на:

* Оружие;
* Одежду;
* Расходуемые предметы;
* Дополнительные устройства.
* Аугментации / Импланты.

Оружие располагается на персонаже в установленных местах. У каждого оружия свои анимации для того, чтобы взять или убрать его.

Оружие подразделяется на ближнее и дальнее.

У ближнего оружия есть свои последовательности ударов (комбо). В комбо определены урон, последовательность, условия (к примеру, может быть выполнено только в воздухе) и воспроизводимая анимация.

У дальнего оружия свои размер магазина, возможные и установленные дополнительные устройства, кривая отдачи, начальная скорость боеприпасов и возможные и используемые боеприпасы. У боеприпасов свои скорость, урон и тип.

Дополнительные устройства крепятся к дальнему оружию, улучшая характеристики оружия или добавляя ему какие-либо свойства.

Одежда может использоваться игроком и как элемент внешнего вида, и как защита. При любом из применений у одежды есть некоторый коэффициент защиты.

Аугментации и импланты улучшают характеристики персонажа. Заменяют или дополняют какую-либо часть тела персонажа. У каждого свой набор действий. Действия применяются либо пока экипированы, либо в определённый момент.

Все предметы самой высокой редкости имеются в одном единственном экземпляре, а единственный способ их получения – специальные задания.

Так как большинство игр данного жанра представляют собой одиночный сюжет в многопользовательской игре, что воспринимается не самым приятным образом, из заготовленных частей генерируются как побочные, так и сюжетные задания, но не задания для уникальных предметов. Задания для уникальных предметов прорабатываются и создаются вручную из тех же частей.

Сюжетные задания генерируются для конкретного игрока или группы игроков и выполняются только ими.

За побочное задание может любой игрок. Побочные задания даются неигровыми персонажами. Выполнить побочное задание может один игрок или группа игроков. За задания предполагается какая-либо награда.

Специальные задания выполняются в одиночку.

Для каждой из заготовленных частей определены возможные продолжения в виде других (либо той же самой) частей.

Вся вышеописанная информация должна храниться в удобном для редактирования и обработки виде. Единственный способ достичь этого удобства – создать базу данных.

* 1. Техническое задание
     1. Функциональные требования

В базе данных должна храниться информация об игроках: логин, пароль, почта, страна, никнейм, изображение профиля и список друзей. Так же должны храниться персонажи, как игроков, так и неигровые, их характеристики, имеющиеся и используемые умения, имеющиеся у них предметы, сами предметы, их свойства, а также задания.

У персонажа кроме характеристик должны храниться имя, максимальное здоровье, % здоровья, а так же точка, к которой необходимо его возвращать, Mesh и подобранные предметы.

У предметов необходимо хранить название, редкость, mesh, информацию о том, что это за предмет.

Для оружия необходимо хранить место крепления и необходимые анимации, для дальнего – устройства, размер магазина, боеприпасы, начальную скорость боеприпаса и кривую отдачи.

Для ближнего предусмотрены комбо. Они содержат саму последовательность ударов, наносимый урон, анимацию и необходимые для выполнения условия.

У одежды имеется такая информация, как количество подавляемого урона и тип (внешний вид или полноценная защита).

У аугментаций и имплантов необходимо хранить то, как они действуют, что делают и где располагаются на персонаже.

Необходимо хранить задания, их тип, кто принял и кто выполнил.

Задания составляются из частей, поэтому необходимо хранить части, подходящие продолжения и из каких частей состоит конкретное задание.

* + 1. Минимальные системные требования
* Процессор: 64 bit, 2 ядра;
* Операционная система: 64 bit;
* Оперативная память: 2 ГБ;
* RAID 1 и выше;

RAID – технология виртуализации данных, которая объединяет несколько дисков в логический элемент для избыточности и повышения производительности.

RAID 1 – массив из двух (или более) дисков, являющихся полными копиями друг друга.

* + 1. Требование к документации

Необходимо создать руководство администратора базы данных.

* 1. Выбор средств разработки
     1. Выбор СУБД

1. PostgreSQL – объектно-реляционная СУБД.

Преимущества:

* высокопроизводительные и надёжные механизмы транзакций и репликации;
* расширяемая система встроенных языков программирования;
* наследование;
* возможность индексирования геометрических объектов;
* бесплатность.

Недостатки:

* низкая производительность при чтении;
* сложность.

1. MySQL – реляционная СУБД, разрабатывается и поддерживается Oracle. Обычно используется в качестве сервера. Гибкость обеспечивается поддержкой большого количества типов таблиц.

Преимущества:

* простота;
* скорость;
* безопасность;
* масштабируемость – может быть использована для работы и с малыми, и с большими объёмами данных;
* бесплатность.

Недостатки:

* недостаточная надёжность;
* низкая скорость разработки;
* проблемы с многопоточностью.

1. Microsoft SQL Server – Реляционная СУБД, разработанная Microsoft. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия. Использует Transact-SQL (T-SQL) – реализацию ANSI/ISO по SQL с расширениями.

Преимущества:

* производительность;
* надёжность и безопасность;
* простота;
* масштабируемость.

Недостатки:

* даже при тщательной настройке способен занять все доступные ресурсы.
* высокая цена.
  + 1. Выбор СУБД. Вывод

Для разработки была выбрана СУБД PostgreSQL, так как в ней имеются необходимые возможности для реализации требуемой БД.

* + 1. Дополнительные средства разработки

Так же для создания базы данных были выбраны следующие средства разработки:

* Java – Объектно-ориентированный язык программирования;
* Intellij IDEA – среда разработки для Java;
* Spring framework – универсальный фреймворк с открытым исходным кодом для Java;
* Hibernate – реализация спецификации JPA, предназначенная для решения задач объектно-реляционного отображения (ORM);
* Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации;
* Maven – фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM.

1. Разработка Базы Данных
   1. Создание структуры
      1. Начальная структура

Начальная структура базы данных представлена на рисунке 1.

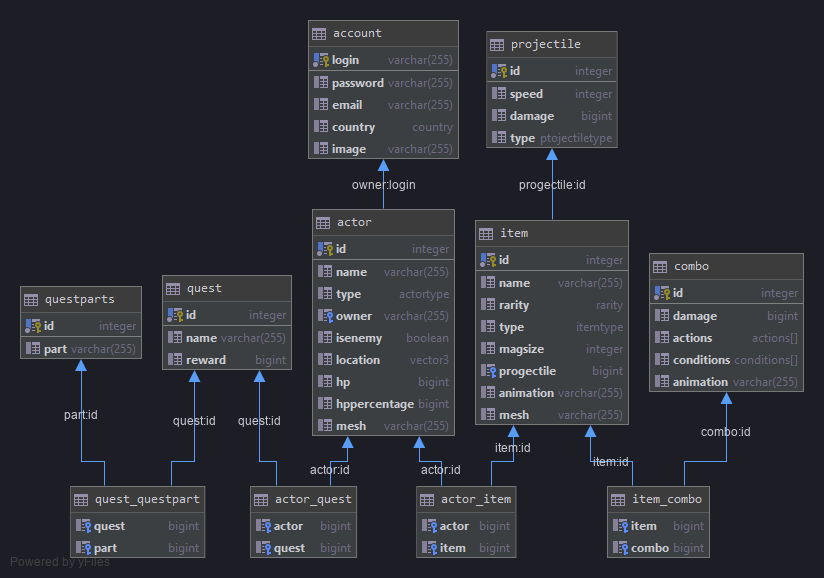


Рисунок 1 – начальная структура.

Проблема данной базы данных в лишних полях, которые могут никогда не использоваться конкретным предметом и неудобстве определения предмета, необходимости работать с базой данных вручную.

* + 1. Нормализация.

Таблица Account представлена таблицей 1.

Таблица 1 – таблица Account

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Login | Varchar(255) | Уникальный идентификатор игрока. |
| Password | Varchar(255) | Пароль игрока |
| Name | Varchar(255) | Имя (псевдоним) игрока |
| Image | Varchar(255) | Путь к файлу в файловой системе сервера |
| Email | Varchar(255) | Почта игрока |
| Country | Varchar(255) | Страна игрока |

Для нормализации таблицу Actor необходимо разделить на 3 таблицы: Actor, PlayableActor и NPC. В таблице Actor, которая представлена таблицей 2, будут описываться общие свойства.

Таблица 2 – Actor

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор персонажа. |
| Name | Varchar(255) | Имя персонажа. |
| MaxHP | Integer | Максимальное здоровье персонажа. |
| Location | Vector3 | Трёхмерная точка. |
| Mesh | Varchar(255) | Путь к файлу в файловой системе сервера. |

Таблица PlayableActor

Таблица 3

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор персонажа. |
| Owner | Varchar(255) | Игрок, которому принадлежит персонаж. |

Таблица NPC

Таблица 4

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор персонажа. |
| IsEnemy | Varchar(255) | Враждебен ли к персонаж к игроку. |

Таблица Skill

Таблица 5

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор навыка. |
| Type | Varchar(255) | Тип навыка. |
| Val | Real | Насколько изменяет то, на что влияет. |

Таблицу Item необходимо разделить на таблицы Item, Weapon, Clothes, Usable, Augmentation\_Implant. Таблица Item представлена таблицей 5.

Таблица 6

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |
| Name | Varchar(255) | Название предмета. |
| Mesh | Varchar(255) | Путь к файлу в файловой системе сервера. |
| Rarity | Bigint | Константное значение количества предметов данной редкости. |

Таблицу Weapon необходимо разделить на таблицы Weapon, Melee и Ranged.

Таблица 7

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |
| Mount | Varchar(255) | Описывает, где крепится оружие. |

Таблица Melee

Таблица 8

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |

Таблица Ranged

Таблица 9

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |
| RecoilCurve | Varchar(255) | Путь к файлу в файловой системе сервера |
| MaxMagazineCount | Integer | Размер магазина. |

Продолжение таблицы 9

|  |  |  |
| --- | --- | --- |
| ProjectileStartSpeed | Integer | Начальная скорость |

Таблица Clothes

Таблица 10

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |
| DamageReduction | Integer | Количество подавляемого урона |
| Type | Varchar(255) | Тип предмета. |

Таблица Usable

Таблица 11

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |
| Type | Varchar(255) | Тип предмета. |

Таблица Augmentation\_Implant

Таблица 12

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |
| Type | Varchar(255) | Тип предмета. |
| UsageType | Varchar(255) | Тип использования. |

Таблица AugmImplAction

Таблица 13

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор действия. |
| Type | Varchar(255) | Тип действия. |
| Description | Text | Описание действия. |

Для ближнего оружия имеется система комбо. Таблица Combo

Таблица 14

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор комбо. |
| Damage | Real | Урон, наносимый атакой. |
| Animation | Varchar(255) | Путь к файлу в файловой системе сервера. |

Для дальнего оружия необходимы таблицы Device (таблица 15) и Projectile (таблица 16), которые так же наследуются от Item.

Таблица 15

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |
| Type | Integer | Тип устройства. |

Таблица 16

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |
| Type | Varchar(255) | Тип боеприпаса. |
| Damage | Real | Урон, наносимый боеприпасом. |

Таблица Quest

Таблица 17

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор задания. |
| Name | Varchar(255) | Название задания. |
| Type | Varchar(255) | Тип задания. |
| NPC | Bigint | Персонаж, предоставляющий задание. |

Таблица QuestPart

Таблица 18

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор части задания. |
| Part | Varchar(255) | Путь к файлу в файловой системе сервера. |

* + 1. Связи между объектами базы данных.

Список друзей игрока необходимо отобразить связью многие ко многим. Таблица Friends

Таблица 19

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Account | Varchar(255) | Логин игрока |
| Friend | Varchar(255) | Логин другого игрока. |

Между игроком и персонажем связь один ко многим (несколько персонажей у игрока, один игрок, которому принадлежит персонаж).

Между персонажем и навыками связь многие ко многим, а так же необходима информация, используется ли конкретным персонажем конкретный навык.

Таблица 20

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Actor | Bigint | Id персонажа |
| Skill | Bigint | Id навыка |
| IsEquipped | Boolean | Используется ли данный навык. |

Между предметом и персонажем связь многие ко многим.

Таблица 21

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Actor | Bigint | Id персонажа |
| Item | Bigint | Id предмета |

Между дальним оружием и боеприпасами связь многие ко многим, а также информация, используются ли они в данный момент.

Таблица 22

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Ranged | Bigint | Id предмета. |
| Projectile | Bigint | Id боеприпаса. |
| IsEquipped | Boolean | Используется ли данный вид боеприпасов. |

Между дальним оружием и устройствами связь многие ко многим, а также информация, используются ли они в данный момент.

Таблица 23

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Ranged | Bigint | Id предмета. |
| Device | Bigint | Id устройства. |
| IsEquipped | Boolean | Используется ли данное устройство. |

Поскольку необходимо обрабатывать большое количество данных, база данных должна быть автоматизирована. Для автоматизации большинства процессов, а также удобства совместной работы базы данных и игры база данных будет разрабатываться на java.

Разработка на Java так же позволяет сконцентрироваться на основных объектах, все вспомогательные таблицы будут создаваться и редактироваться автоматически.

* 1. Разработка SQL запросов
     1. Создание таблиц

Account.java

@Entity  
@Table(name = "Account")  
public class Account {

@Id  
@Column(nullable = false)  
private String login**;**

@Column(nullable = false)

private String password**;**

@Column  
private String name**;**

@ManyToMany  
private List<Account> friends**;**

@Column  
private String image**;**

@Enumerated(EnumType.STRING)   
@Column  
private Country country = Country.NOT\_SELECTED**;**

// Конструкторы, геттеры, сеттеры и методы

}

Actor.java

@Entity  
@Inheritance(strategy = InheritanceType.JOINED)   
@TypeDefs( { @TypeDef(typeClass = VectorHibernateType.class**,** defaultForType = Vector.class) } )  
public class Actor {

@Id  
@GeneratedValue(strategy = GenerationType.SEQUENCE**,** generator = "Actor\_generator")  
@SequenceGenerator(name = "Actor\_generator"**,** sequenceName = "Actor\_sequence")  
protected long id**;**

@Column  
protected String name**;**

@Column  
protected int maxHP**;**

@Column  
protected int HPPersentage**;**

@Transient  
protected int HP**;**

@Column(columnDefinition = "vector3")  
protected Vector location**;**

@Column  
protected String mesh**;**

@ManyToMany  
protected List<Item> inventory**;**

@ElementCollection  
@CollectionTable(name = "actor\_skills"**,** joinColumns = { @JoinColumn(name = "actor") } )   
@MapKeyJoinColumn(name = "skill")  
@Column(name = "IsEquipped")  
protected Map<Skill**,** Boolean> skills**;**

// Конструкторы, геттеры, сеттеры и методы

}

PlayableActor.java

@Entity  
public class PlayableActor extends Actor {

@ManyToOne  
protected Account owner**;**

}

NPC.java

@Entity  
public class NPC extends Actor {

@Column  
protected boolean bIsEnemy**;**

// Конструкторы, геттеры, сеттеры и методы

}

SkillType.java

public enum SkillType{

// Элементы перечисления

}

Skill.java

@Entity  
@Inheritance(strategy = InheritanceType.JOINED)  
public class Skill {

@Id  
@GeneratedValue(strategy = GenerationType.SEQUENCE**,** generator = "Skill\_generator")  
@SequenceGenerator(name = "Skill\_generator"**,** sequenceName = "Skill\_sequence")  
protected long id**;**

@Column  
protected float val**;**

@Enumerated(EnumType.STRING)  
@Column  
protected SkillType type**;**

// Конструкторы, геттеры, сеттеры и методы

}

Vector.java

public class Vector implements Serializable {

public double x = 0**;**

public double y = 0

public double z = 0**;**

// Конструкторы, реализация методов Serializable

}

VectorHibernateType.java

public class VectorHibernateType implements UserType {

// Реализация методов UserType

}

Rarity.java

public final class Rarity

// Константы количества экземпляров по редкости

}

Item.java

@Entity  
@Inheritance(strategy = InheritanceType.JOINED)  
public class Item {

@Id  
@GeneratedValue(strategy = GenerationType.SEQUENCE**,** generator = "Item\_generator")  
@SequenceGenerator(name = "Item\_generator"**,** sequenceName = "Item\_sequence")  
protected long id**;**

@Column  
protected String name**;**

@Column  
protected String mesh**;**

@Column  
protected long rarity = Rarity.C**;**

// Конструкторы, геттеры, сеттеры и методы

}

WeaponMount.java

public enum WeaponMount {

// Элементы перечисления

}

Weapon.java

@Entity  
@Inheritance(strategy = InheritanceType.JOINED)  
public class Weapon extends Item {

@Enumerated(EnumType.STRING)  
@Column  
protected WeaponMount mount**;**

@Column  
protected String takeAnimation**;**

@Column  
protected String putAnimation**;**

// Конструкторы, геттеры, сеттеры и методы

}

Melee.java

@Entity  
public class Melee extends Weapon {

@ElementCollection  
protected List<Combo> comboList**;**

@Transient  
protected float damage**;**

// Конструкторы, геттеры, сеттеры и методы

}

Action.java

public enum Action {

// Элементы перечисления

}

Condition.java

public enum Condition {

// Элементы перечисления

}

Combo.java

@Entity  
public class Combo {

@Id  
@GeneratedValue(strategy = GenerationType.SEQUENCE**,** generator = "Combo\_generator")  
@SequenceGenerator(name = "Combo\_generator"**,** sequenceName = "Combo\_sequence")  
private long id**;**

@Column  
private float damage**;**

@Enumerated(EnumType.STRING)  
@ElementCollection  
private List<Action> seq**;**

@Enumerated(EnumType.STRING)  
@ElementCollection  
private List<Condition> conditions**;**

// Конструкторы, геттеры, сеттеры и методы

}

Ranged.java

@Entity  
public class Ranged extends Weapon {

@ElementCollection  
@CollectionTable(name = "ranged\_projectiles"**,** joinColumns = { @JoinColumn(name = "ranged") } )  
@MapKeyJoinColumn(name = "projectile")  
@Column(name = "IsEquipped")  
protected Map<Projectile**,** Boolean> projectiles**;**

@ElementCollection  
@CollectionTable(name = "ranged\_devices" joinColumns = { @JoinColumn(name = "ranged") } )  
@MapKeyJoinColumn(name = "device")  
@Column(name = "IsEquipped")  
protected Map<Device**,** Boolean> devices**;**

// Конструкторы, геттеры, сеттеры и методы

}

DeviceType.java

public enum DeviceType {

// Элементы перечисления

}

Device.java

@Entity  
public class Device extends Item {

@Enumerated(EnumType.STRING)  
@Column  
protected DeviceType type**;**

// Конструкторы, геттеры, сеттеры и методы

}

ProjectileType.java

public enum ProjectileType {

// Элементы перечисления

}

Projectile.java

@Entity  
public class Projectile extends Item {

@Column  
private float damage**;**

@Enumerated(EnumType.STRING)  
@Column  
private ProjectileType type**;**

// Конструкторы, геттеры, сеттеры и методы

}

QuestType.java

public enum QuestType {

// Элементы перечисления

}

Quest.java

@Entity  
public class Quest {

@Id  
@GeneratedValue(strategy = GenerationType.SEQUENCE**,** generator = "Quest\_generator")  
@SequenceGenerator(name = "Quest\_generator"**,** sequenceName = "Quest\_sequence")  
protected long id**;**

@Column  
protected String name**;**

@Enumerated(EnumType.STRING)  
@Column  
protected QuestType type**;**

@ManyToOne  
protected NPC npc**;**

@OneToMany  
protected List<QuestPart> parts**;**

@ElementCollection  
@CollectionTable(name = "accepted\_quests"**,** joinColumns = { @JoinColumn(name = "quest") } )  
@MapKeyJoinColumn(name = "actor")  
@Column(name = "IsCompleted")  
protected Map<PlayableActor**,** Boolean> actors**;**

// Конструкторы, геттеры, сеттеры и методы

}

QuestPart.java

@Entity  
public class QuestPart {

@Id  
@GeneratedValue(strategy = GenerationType.SEQUENCE**,** generator = "QuestPart\_generator")  
@SequenceGenerator(name = "QuestPart\_generator"**,** sequenceName = "QuestPart\_sequence")  
protected long id**;**

@Column  
private String description**;**

@Column  
protected String part**;**

@ManyToMany  
protected List<QuestPart> nextParts**;**

// Конструкторы, геттеры, сеттеры и методы

}

* + 1. Запросы на создание, удаление, редактирование и выборку записей
  1. Конечная структура базы данных

Итоговая структура базы данных представлена на рисунке 2.

Рисунок 2 – Итоговая структура базы данных

* 1. Защита базы данных

Использованные источники

Postgresql <https://ru.wikipedia.org/wiki/PostgreSQL>

Недостатки postgresql <https://tproger.ru/translations/sqlite-mysql-postgresql-comparison/>

Достоинства и недостатки MySQL [https://depix.ru/articles/sistema \_upravleniya\_bazami\_dannyh\_mysql](https://depix.ru/articles/sistema%20_upravleniya_bazami_dannyh_mysql), <http://drach.pro/blog/hi-tech/item/145-db-comparison>

MySQL <https://ru.wikipedia.org/wiki/MySQL>

Microsoft SQL Server <https://ru.wikipedia.org/wiki/Microsoft_SQL_Server>

Java <https://ru.wikipedia.org/wiki/Java>

Itellij IDEA <https://ru.wikipedia.org/wiki/IntelliJ_IDEA>

Spring framework <https://ru.wikipedia.org/wiki/Spring_Framework>

Hibernate <https://ru.wikipedia.org/wiki/Hibernate_(библиотека)>

Docker <https://ru.wikipedia.org/wiki/Docker>

Maven <https://ru.wikipedia.org/wiki/Apache_Maven>

База данных <https://ru.wikipedia.org/wiki/База_данных>