СОДЕРЖАНИЕ

[Введение 3](#_Toc25129340)

[1 Аналитический обзор 4](#_Toc25129341)

[1.1 Описание предметной области 4](#_Toc25129342)

[1.2 Техническое задание 6](#_Toc25129343)

[1.2.1 Функциональные требования 6](#_Toc25129344)

[1.2.2 Минимальные системные требования 7](#_Toc25129345)

[1.2.3 Требование к документации 8](#_Toc25129346)

[1.3 Выбор средств разработки 8](#_Toc25129347)

[1.3.1 Выбор СУБД 8](#_Toc25129348)

[1.3.2 Выбор СУБД. Вывод 9](#_Toc25129349)

[1.3.3 Дополнительные средства разработки 9](#_Toc25129350)

[2 Разработка Базы Данных 11](#_Toc25129351)

[2.1 Создание структуры 11](#_Toc25129352)

[2.1.1 Начальная структура 11](#_Toc25129353)

[2.1.2 Нормализация. 11](#_Toc25129354)

[2.1.3 Связи между объектами базы данных. 16](#_Toc25129355)

[2.2 Разработка SQL запросов 17](#_Toc25129356)

[2.2.1 Создание таблиц 18](#_Toc25129357)

[2.2.2 Запросы на создание, удаление, редактирование и выборку записей 26](#_Toc25129358)

[2.3 Конечная структура базы данных 29](#_Toc25129359)

[2.4 Защита базы данных 29](#_Toc25129360)

[3 Руководство Администратора Базы Данных 30](#_Toc25129361)

[Заключение 31](#_Toc25129362)

[Использованные источники 32](#_Toc25129363)

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.

Заходя в игру, игроку необходимо создать собственного персонажа. Игрок настраивает модель персонажа, после настройки придумывает имя, после чего начинает играть.

Начальные характеристики, такие как максимальное здоровье и уровень, для всех персонажей одинаковые. Игроки могут набирать уровень, чтобы открывать навыки. Для применения навыка его необходимо выбрать, все не выбранные навыки не будут действовать. Выбранные навыки можно изменить в любой момент.

В процессе игры игрок будет натыкаться на персонажей, не принадлежащих игрокам. Одни из них будут игнорировать игрока или пытаться взаимодействовать, другие же попытаются его убить. Враждебность персонажа к игроку определяется такими условиями, как:

* отношение игрока к персонажу;
* действия игрока, о которых персонаж мог получить информацию;
* действия игрока по отношению к персонажу.

Некоторые персонажей враждебны ко всем.

Персонажи могут просить у игроков помощь с чем-либо (предлагать задания), предлагая награду. Награду получит тот игрок (группа игроков), который раньше остальных сообщил персонажу о выполнении задания.

Награда может быть:

* предметом, таким как: оружие, одежда, расходуемый предмет, аугментации, импланты, особые боеприпасы и дополнительные устройства для оружия;
* опытом, необходимым для уровня;
* следующим заданием, если задание было частью серии заданий;
* улучшение отношения персонажа к игроку;
* комбинации вышеперечисленных наград.

Также игрок может найти предметы в мире игры. Предметы имеют редкость, которая определяет количество экземпляров данного предмета. Самая высокая редкость предполагает наличие в мире всего одного экземпляра данного предмета. Для получения таких предметов необходимо пройти специальные задания, рассчитанные на одного игрока. Эти задания позволят игроку понять, нужен ли ему этот предмет и если да, то доказать, что он должен быть у него.

Для выполнения поставленных целей игрок может использовать ближнее и дальнее оружие. Каждое ближнее оружие основывается на применении игроком списка комбо (серии последовательных ударов) этого оружия. Дальнее оружие может быть модифицировано дополнительными устройствами, такими как: прицелы, магазины, приклады, насадки и т.д. При использовании дальнего не маловажным является подобрать подходящие к ситуации боеприпасы.

Важным так же является защита. Одежда может быть предназначена как для защиты, так и для декоративного использования. Второй тип одежды может использоваться поверх первого.

Многие проблемы могут решить расходуемые предметы. Применяются один раз. При максимально редкости, в зависимости от предмета, после применения могут быть найдены снова.

Не исключено, что игрок в процессе игры не справится с задачей, а персонаж умрёт… в таком случае игрок должен продолжить игру с ближайшей контрольной точки (к примеру, ближайший или последний посещённый город). При выходе из игры во время путешествия по миру игрок может продолжить путешествие с того места, где он был при выходе, если локация позволяет сохранять расположение игрока.

Основная особенность игры – постоянное разнообразие заданий. Задания, как основные, так и побочные, генерируются из самых простых элементов, применимых в любом заданий.

Множество элементов игры необходимо постоянно хранить, обновлять и обрабатывать. Самый эффективный способ хранения и обработки данных – база 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, 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. Выбранная СУБД

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.

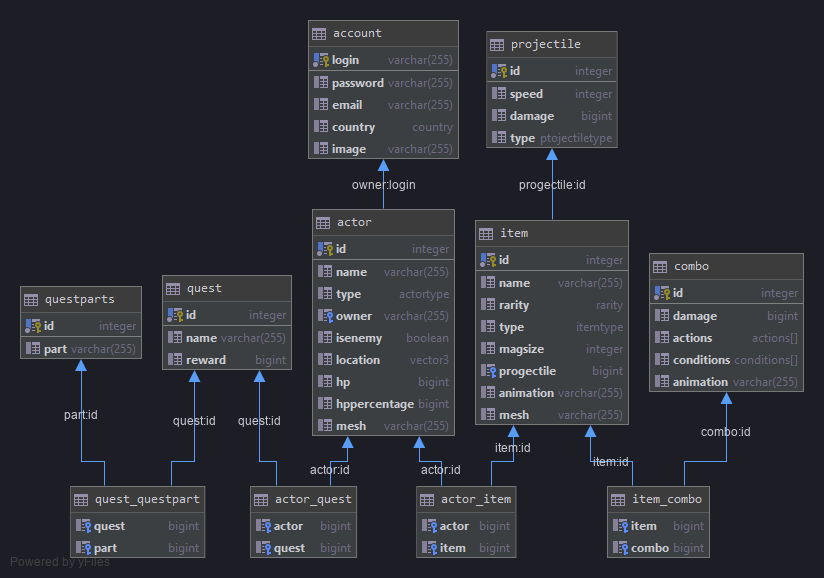


Рисунок 1 – начальная структура.

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 | Real | Максимальное здоровье персонажа. |
| Location | Vector3 | Трёхмерная точка. |
| Mesh | Varchar(255) | Путь к файлу в файловой системе сервера. |
| Level | Integer | Уровень персонажа |

Таблица PlayableActor

Таблица 3

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор персонажа. |
| Owner | Varchar(255) | Игрок, которому принадлежит персонаж. |
| Exp | Real | Текущее количество очков оптыта |

Таблица NPC

Таблица 4

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

Продолжение таблицы 4

|  |  |  |
| --- | --- | --- |
| Type | Varchar(255) | Тип отношения к персонажу. |

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 | Varchar(255) | Редкость предмета. |

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. |
| TakeAnimation | Varchar(255) | Путь к файлу в файловой системе сервера. |
| PutAnimation | Varchar(255) | Путь к файлу в файловой системе сервера. |

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. |
| ProjectileStartSpeed | Real | Начальная скорость |

Таблица Clothes

Таблица 10

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Id | Bigint | Уникальный идентификатор предмета. |
| DamageReduction | Real | 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 представлена таблицей 14.

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. |
| Sequence | Varchar(255)[] | Последовательность действий. |
| Conditions | Varchar(255)[] | Условия. |
| Weapon\_id | Bigint | Оружие, которому принадлежит. |

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.

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, а также информация о том, в какой слот он экипирован (для оружия), или количество экземпляров данного предмета.

Таблица 21

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Actor | Bigint | Id персонажа |
| Item | Bigint | Id навыка |
| Actor\_slot\_or\_count | Integer | Слот, в котором находится предмет, или количество данного 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. |

Ближнее оружие и комбо имеют связь один ко многим.

Персонаж игрока и NPC имею связь многие ко многим и информацию о враждебности NPC к персонажу.

Таблица 24

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| Actor | Bigint | Id персонажа |
| NPC | Bigint | Id NPC |
| Hostility | Real | Уровень враждебности. |

AugmImpl и AugmImplAction имеют связь многие ко многим.

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

Создание типа vector3 (трёхмерная точка).

CREATE TYPE vector3 AS (x double precision**,** y double precision**,** z double precision);

* + 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**;**

@Column (unique = true)  
private String email;

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

@OneToMany(mappedBy = "owner", fetch = FetchType.LAZY)  
private List<PlayableActor> actors;

// 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 float maxHP;  
@Transient  
protected float HP;  
@Column(columnDefinition = "vector3")  
protected Vector location;  
@Column  
protected String mesh;  
@ElementCollection  
@CollectionTable(name = "inventory",joinColumns = @JoinColumn(name = "actor"))  
@MapKeyJoinColumn(name = "item")  
@Column(name = "actor\_slot")  
protected Map<Item, Integer> 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**;**

@Column  
protected float exp;

@ElementCollection(fetch = FetchType.LAZY)  
@CollectionTable(name = "accepted\_quests", joinColumns = @JoinColumn(name = "quest") )  
@MapKeyJoinColumn(name = "actor")  
@Column(name = "IsCompleted")  
protected Map<Quest, Boolean> quests;

// Конструкторы, геттеры, сеттеры и методы

}

NPCType.java

public enum NPCType {  
// Элементы перечисления

}

NPC.java

@Entity  
public class NPC extends Actor {

@ElementCollection  
@CollectionTable(name = "hostility\_to\_player", joinColumns = @JoinColumn(name = "NPC"))  
@MapKeyJoinColumn(name = "player")  
@Column(name = "hostility")  
protected Map<PlayableActor, Double> hostility;

@Enumerated(EnumType.STRING)  
@Column  
protected NPCType type;

@OneToMany(mappedBy = "npc", fetch = FetchType.LAZY)  
protected List<Quest> quests;

// 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 enumRarity

// Элементы перечисления

}

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**;**

@Enumerated(EnumType.STRING)  
@Column  
protected Rarity 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 {

@OneToMany(mappedBy = “weapon”)  
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

@TypeDefs( @TypeDef(name = "varchar[]", typeClass = StringArrayType.class) )

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)  
@Type(type = "varchar[]")  
@Column(columnDefinition = "varchar(255)[]")

private List<Action> seq**;**

@Enumerated(EnumType.STRING)  
@Type(type = "varchar[]")  
@Column(columnDefinition = "varchar(255)[]")

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")  
private Map<Projectile, Boolean> projectiles;

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

@Column  
private double projectileStartSpeed;

@Column  
private int maxMagazineCount;

@Transient  
private int magazineCount;

@Column  
private String recoilCurve;

// 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**;**

@ElementCollection(fetch = FetchType.LAZY)  
@CollectionTable(name = "ranged\_devices", joinColumns = @JoinColumn(name = "ranged"))  
@MapKeyJoinColumn(name = "device")  
@Column(name = "IsEquipped")  
private Map<Ranged, Boolean> weapons;

// 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**;**

@ElementCollection(fetch = FetchType.LAZY)  
@CollectionTable(name = "ranged\_projectiles", joinColumns = @JoinColumn(name = "ranged"))  
@MapKeyJoinColumn(name = "projectile")  
@Column(name = "IsEquipped")  
private Map<Ranged, Boolean> weapons;

// 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

Для каждого класса имеются как стандартные методы add, delete, update, getOne и getAll, так и зависящие от класса.

Account:

* @Query("select a from Account a where a.country = :country")

List<Account> getAllByCountry(@Param("country") Country country);

* @Query("select a from Account a where a.email = :email")

Account getByEmail(@Param("email") String email);

* @Query("select a from Account a where a.name = :name")

List<Account> getAllByName(@Param("name") String name);

Actor

* @Query("select c from Actor c where c.name = :name")

List<Actor> getAllByName(@Param("name") String name);

PlayableActor

* @Query("SELECT a from PlayableActor a where a.owner.login = :owner")

List<PlayableActor> getAllByOwner(@Param("owner") String ownerLogin);

* @Query("select a from PlayableActor a where a.name = :name")

List<PlayableActor> getAllByName(@Param("name") String actorName);

NPC

* @Query("select c from NPC c where c.name = :name")

List<NPC> getAllByName(@Param("name") String name);

Skill

* @Query("select s from Skill s where s.type = :sType")

List<Skill> getAllByType(@Param("sType") SkillType type);

Item

* @Query("SELECT i from Item i where i.rarity = :rarity")

List<Item> getAllByRarity(@Param("rarity") Rarity rarity);

* @Query("select i from Item i where i.name = :name")

List<Item> getAllByName(@Param("name") String name);

Weapon

* @Query("SELECT i from Weapon i where i.rarity = :rarity")

List<Weapon> getAllByRarity(@Param("rarity") Rarity rarity);

* @Query("select i from Weapon i where i.name = :name")

List<Weapon> getAllByName(@Param("name") String name);

* @Query("select i from Weapon i where i.mount = :mount")

List<Weapon> getAllByMount(@Param("mount") WeaponMount mount);

Melee

* @Query("SELECT m from Melee m where m.rarity = :rarity")

List<Melee> getAllByRarity(@Param("rarity") Rarity rarity);

* @Query("select m from Melee m where m.name = :name")

List<Melee> getAllByName(@Param("name") String name);

* @Query("select m from Melee m where m.mount = :mount")

List<Melee> getAllByMount(@Param("mount") WeaponMount mount);

Ranged

* @Query("SELECT i from Ranged i where i.rarity = :rarity")

List<Ranged> getAllByRarity(@Param("rarity") Rarity rarity);

* @Query("select i from Ranged i where i.name = :name")

List<Ranged> getAllByName(@Param("name") String name);

* @Query("select i from Ranged i where i.mount = :mount")

List<Ranged> getAllByMount(@Param("mount")WeaponMount mount);

* @Query("select i from Ranged i where i.projectileStartSpeed >= :minSpeed and i.projectileStartSpeed <= :maxSpeed")

List<Ranged> getAllByProjectileStartSpeed(@Param("minSpeed") double minSpeed, @Param("maxSpeed") double maxSpeed);

* @Query("select i from Ranged i where i.maxMagazineCount >= :minCount and i.maxMagazineCount <= :maxCount")

List<Ranged> getAllByMagazineCount(@Param("minCount") double minCount, @Param("maxCount") double maxCount);

Device

* @Query("SELECT i from Device i where i.rarity = :rarity")

List<Device> getAllByRarity(@Param("rarity") Rarity rarity);

* @Query("select i from Device i where i.name = :name")

List<Device> getAllByName(@Param("name") String name);

* @Query("select i from Device i where i.type = :deviceType")

List<Device> getAllByType(@Param("deviceType") DeviceType type);

Projectile

* @Query("SELECT i from Projectile i where i.rarity = :rarity")

List<Projectile> getAllByRarity(@Param("rarity") Rarity rarity);

* @Query("select i from Projectile i where i.name = :name")

List<Projectile> getAllByName(@Param("name") String name);

* @Query("select i from Projectile i where i.type = :projType")

List<Projectile> getAllByType(@Param("projType") ProjectileType type);

* @Query("select i from Projectile i where i.damage >= :minDamage and i.damage <= :maxDamage")

List<Projectile> getAllByDamageBetween(@Param("minDamage") float minDamage, @Param("maxDamage") float maxDamage);

Clothes

* @Query("SELECT i from Clothes i where i.rarity = :rarity")

List<Clothes> getAllByRarity(@Param("rarity") Rarity rarity);

* @Query("select i from Clothes i where i.name = :name")

List<Clothes> getAllByName(@Param("name") String name);

* @Query("select i from Clothes i where i.type = :clothesType")

List<Clothes> getAllByType(@Param("clothesType") ClothesType type);

* @Query("select i from Clothes i where i.damageReduction >= :minReduction and i.damageReduction <= :maxReduction")

List<Clothes> getAllByDamageReductionBetween(@Param("minReduction") double min, @Param("maxReduction") double max);

Usable

* @Query("SELECT i from Usable i where i.rarity = :rarity")

List<Usable> getAllByRarity(@Param("rarity") Rarity rarity);

* @Query("select i from Usable i where i.name = :name")

List<Usable> getAllByName(@Param("name") String name);

* @Query("select i from Usable i where i.item = :itemT")

List<Usable> getAllByItem(@Param("itemT") UsableItemType itemType);

Augmentation\_implant

* @Query("SELECT i from Augmentation\_Implant i where i.rarity = :rarity")

List<Augmentation\_Implant> getAllByRarity(@Param("rarity") Rarity rarity);

* @Query("select i from Augmentation\_Implant i where i.name = :name")

List<Augmentation\_Implant> getAllByName(@Param("name") String name);

* @Query("select i from Augmentation\_Implant i where i.type = :augmType")

List<Augmentation\_Implant> getAllByType(@Param("augmType") AugmImplType type);

* @Query("select i from Augmentation\_Implant i where i.usageType = :usageType")

List<Augmentation\_Implant> getAllByUsageType(@Param("usageType") AugmImplUsageType usageType);

Quest

* @Query("select q from Quest q where q.type = :qType")

List<Quest> getAllByType(@Param("qType") QuestType type);

* 1. Конечная структура базы данных

Итоговая структура базы данных представлена на рисунке 2.

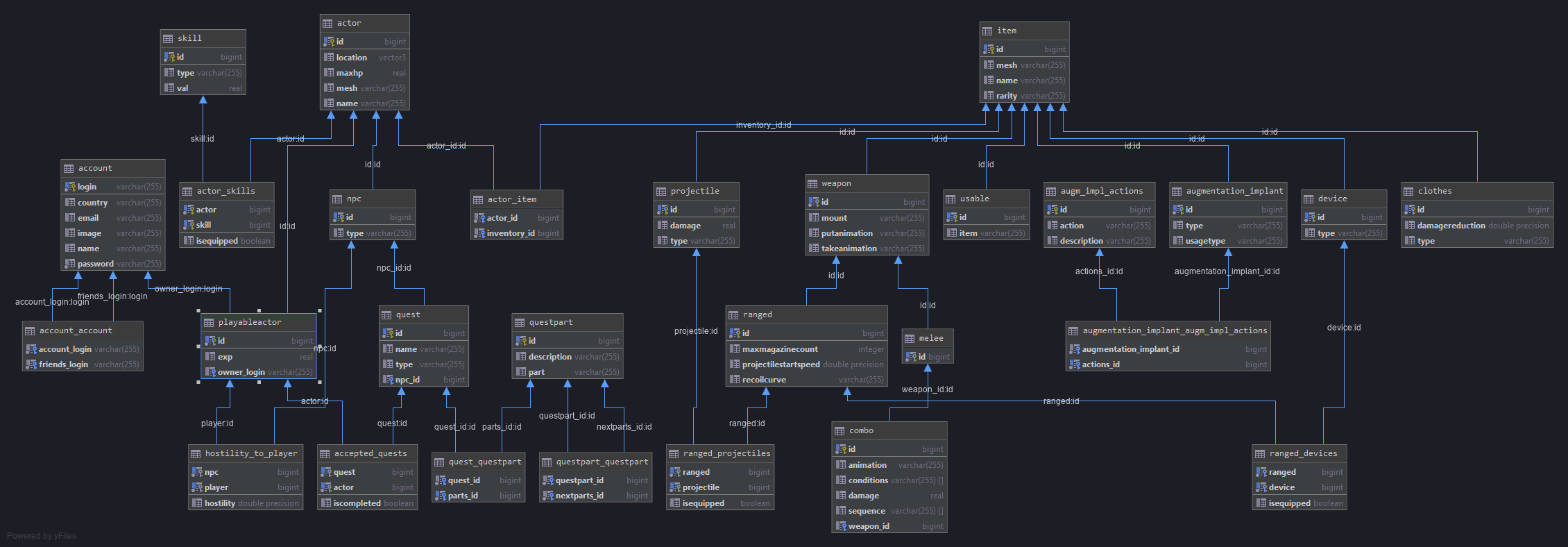


Рисунок 2 – Итоговая структура базы данных

1. Руководство Администратора Базы Данных

Заключение

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>