

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Разработка структуры мультимодельного хранилища данных для применения в
игровой индустрии

Обучающийся / Student Коблов Андрей Александрович
Факультет/институт/кластер/ Faculty/Institute/Cluster школа разработки видеоигр
Группа/Group J4221
Направление подготовки/ Subject area 09.04.03 Прикладная информатика
Образовательная программа / Educational program Технологии разработки
компьютерных игр 2022
Язык реализации ОП / Language of the educational program Русский
Квалификация/ Degree level Магистр
Руководитель ВКР/ Thesis supervisor Ромакина Оксана Михайловна, кандидат физико-
математических наук, Университет ИТМО, факультет инфокоммуникационных технологий,
ведущий инженер

Обучающийся/Student

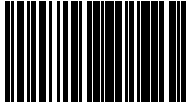
Документ подписан	
Коблов Андрей Александрович	
26.05.2024	

(эл. подпись/ signature)

Коблов Андрей
Александрович

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Ромакина Оксана Михайловна	
26.05.2024	

(эл. подпись/ signature)

Ромакина
Оксана
Михайловна

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Коблов Андрей Александрович
Факультет/институт/кластер/ Faculty/Institute/Cluster школа разработки видеоигр
Группа/Group J4221
Направление подготовки/ Subject area 09.04.03 Прикладная информатика
Образовательная программа / Educational program Технологии разработки компьютерных игр 2022
Язык реализации ОП / Language of the educational program Русский
Квалификация/ Degree level Магистр
Тема ВКР/ Thesis topic Разработка структуры мультимодельного хранилища данных для применения в игровой индустрии
Руководитель ВКР/ Thesis supervisor Ромакина Оксана Михайловна, кандидат физико-математических наук, Университет ИТМО, факультет инфокоммуникационных технологий, ведущий инженер

Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: да / yes

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Анализ существующих одномодельных СУБД, их преимуществ и недостатков. Анализ существующих мультимодельных хранилищ и подходов к работе с ними. Сравнение эффективности использования мультимодельного подхода по отношению к одномодельному. Исследование потребностей игровой отрасли. Анализ и оценка существующих решений в игровой отрасли. Разработка структуры мультимодельного хранилища данных для применения в игровой индустрии. Анализ полученных результатов.

Форма представления материалов ВКР / Format(s) of thesis materials:

текст ВКР в формате pdf, презентация, программный код

Дата выдачи задания / Assignment issued on: 22.02.2024

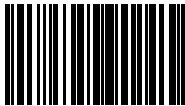
Срок представления готовой ВКР / Deadline for final edition of the thesis 24.05.2024

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/

Документ	
----------	--


Thesis supervisor

подписан	
Ромакина Оксана Михайловна	
30.04.2024	

(эл. подпись)

Ромакина
Оксана
Михайловна


Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Коблов Андрей Александрович	
30.04.2024	

(эл. подпись)

Коблов Андрей
Александрович

Руководитель ОП/ Head
of educational program

Документ подписан	
Карсаков Андрей Сергеевич	
22.05.2024	

(эл. подпись)

Карсаков
Андрей
Сергеевич

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Коблов Андрей Александрович

Факультет/институт/кластер/ Faculty/Institute/Cluster школа разработки видеоигр

Группа/Group J4221

Направление подготовки/ Subject area 09.04.03 Прикладная информатика

Образовательная программа / Educational program Технологии разработки компьютерных игр 2022

Язык реализации ОП / Language of the educational program Русский

Квалификация/ Degree level Магистр

Тема ВКР/ Thesis topic Разработка структуры мультимодельного хранилища данных для применения в игровой индустрии

Руководитель ВКР/ Thesis supervisor Ромакина Оксана Михайловна, кандидат физико-математических наук, Университет ИТМО, факультет инфокоммуникационных технологий, ведущий инженер

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Исследовать применимость мультимодельных СУБД в игровой индустрии и сформировать структуру мультимодельной базы данных для оптимального соответствия потребностям этой отрасли

Задачи, решаемые в ВКР / Research tasks

Проанализировать существующие модели представления данных, выделить их преимущества и недостатки; Выполнить обзор существующих мультимодельных СУБД; Рассмотреть общие тенденции и потребности игровой индустрии; Разработать общую структуру мультимодельной СУБД; Провести сравнительный анализ мультимодельной СУБД с одномодельными; Провести анализ полученных результатов.

Краткая характеристика полученных результатов / Short summary of results/findings

Были проанализированы существующие мультимодельные подходы, их преимущества и недостатки. Помимо этого, было проведено исследование области игровой индустрии, а именно выделены наиболее часто использующиеся механики, системы и компоненты. На основе этих исследований был спроектирован пример мультимодельного хранилища на базе ArangoDB, основываясь на потребностях RPG-игр. Кроме того, был проведён анализ производительности и удобства пользования получившейся мультимодельной СУБД по сравнению с одномодельными подходами. По итогу работы были сформированы рекомендации по применению мультимодельной СУБД в игровых проектах.

Обучающийся/Student

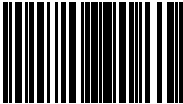
Документ подписан	
Коблов Андрей Александрович	
26.05.2024	

(эл. подпись/ signature)

Коблов Андрей
Александрович

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Ромакина Оксана Михайловна	
26.05.2024	

(эл. подпись/ signature)

Ромакина
Оксана
Михайловна

(Фамилия И.О./ name
and surname)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	8
1 АНАЛИЗ ПОДХОДОВ К ХРАНЕНИЮ ДАННЫХ	11
1.1 Обзор подходов мультимодельного хранения данных	11
1.2 Анализ существующих моделей СУБД.....	12
1.3 Анализ существующих мультимодельных СУБД.....	14
1.4 Анализ преимуществ и недостатков различных моделей	16
1.5 Структурированное сравнение различных моделей	19
1.6 Анализ мультимодельных подходов к СУБД.....	20
1.7 Обзор изначально мультимодельных СУБД.....	21
2 АНАЛИЗ ПОТРЕБНОСТЕЙ ИГРОВОЙ ИНДУСТРИИ.....	24
2.1 Обзор жанров видеоигр	24
2.2 Анализ компонентов видеоигр.....	25
2.3 Особенности игровой индустрии	31
2.4 Существующие решения по хранению игровых данных	32
3 ПРОЕКТИРОВАНИЕ МУЛЬТИМОДЕЛЬНОГО ХРАНИЛИЩА ...	34
3.1 Проектирование основных компонентов хранилища	34
3.2 Проектирование системы рецептов с помощью графовой модели	36
3.3 Проектирование системы таблиц добычи с помощью документной модели.....	39
3.4 Проектирование системы внутриигровой энциклопедии	43
3.5 Проектирование системы подбора игроков	45
3.6 Выводы	47
4 ПРОВЕДЕНИЕ ЭКСПЕРИМЕНТОВ	48
4.1 Выбор одномодельных СУБД для проведения анализа	48
4.2 Сравнение производительности системы квестов	48
4.3 Сравнение производительности системы кланов.....	52
4.4 Сравнение производительности системы диалогов	55
4.5 Анализ результатов	58

ЗАКЛЮЧЕНИЕ.....	59
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	62
ПРИЛОЖЕНИЕ А	67
ПРИЛОЖЕНИЕ В.....	71
ПРИЛОЖЕНИЕ С	72
ПРИЛОЖЕНИЕ D	74
ПРИЛОЖЕНИЕ E.....	75

ВВЕДЕНИЕ

Игровая индустрия, которая зародилась сорок лет назад, за это время смогла стать значимой частью глобальной экономики с многомиллиардным бюджетом и зарекомендовала себя как одну из крупнейших отраслей в сфере развлечений. Благодаря развитию и совершенствованию технологий, особенно в сферах аппаратного и программного обеспечения, игровые компании постепенно переходили от разработки простых казуальных игр с пиксельной графикой к реализации комплексных игровых проектов с реалистичной визуальной составляющей и сложными с точки зрения вычисления игровыми механиками.

По результатам изучения текущих тенденций роста экономических показателей видеоигровой индустрии видно, что эта сфера является одной из наиболее растущих и перспективных на мировом рынке программного обеспечения [1]. Стоит отметить, что особенно активный рост приходится на последние пять лет (рисунок 1).

Одной из задач, с которыми приходится сталкиваться компаниям при разработке игровых проектов, является проектирование систем хранения игровых данных. В зависимости от нужд предпочтение при выборе хранилищ отдаётся либо своим собственным наработкам, уникальным для конкретной студии, движка или игры, либо уже готовым решениям в виде баз данных. Собственное хранилище при правильном проектировании является одним из наиболее удобных и эффективных способов хранения, поскольку создаётся под обработку данных, специфичных для конкретного игрового проекта. Отдельной задачей также может ставиться наибольшая скорость доступа к данным, где собственные хранилища также будут в выигрыше в сравнении с готовыми решениями. Однако создание таких систем является очень трудоёмким процессом, поэтому их выбирают только в исключительных случаях, например, если объём данных

небольшой. Такая ситуация часто распространена среди однопользовательских игр. Однако если рассматривать многопользовательские игры, где объём данных постоянно растёт, либо необходимо часто получать доступ к данным, сложность создания такой системы сильно растёт. Помимо этого, существенным недостатком хранилищ собственной разработки является необходимость их дальнейшей поддержки.

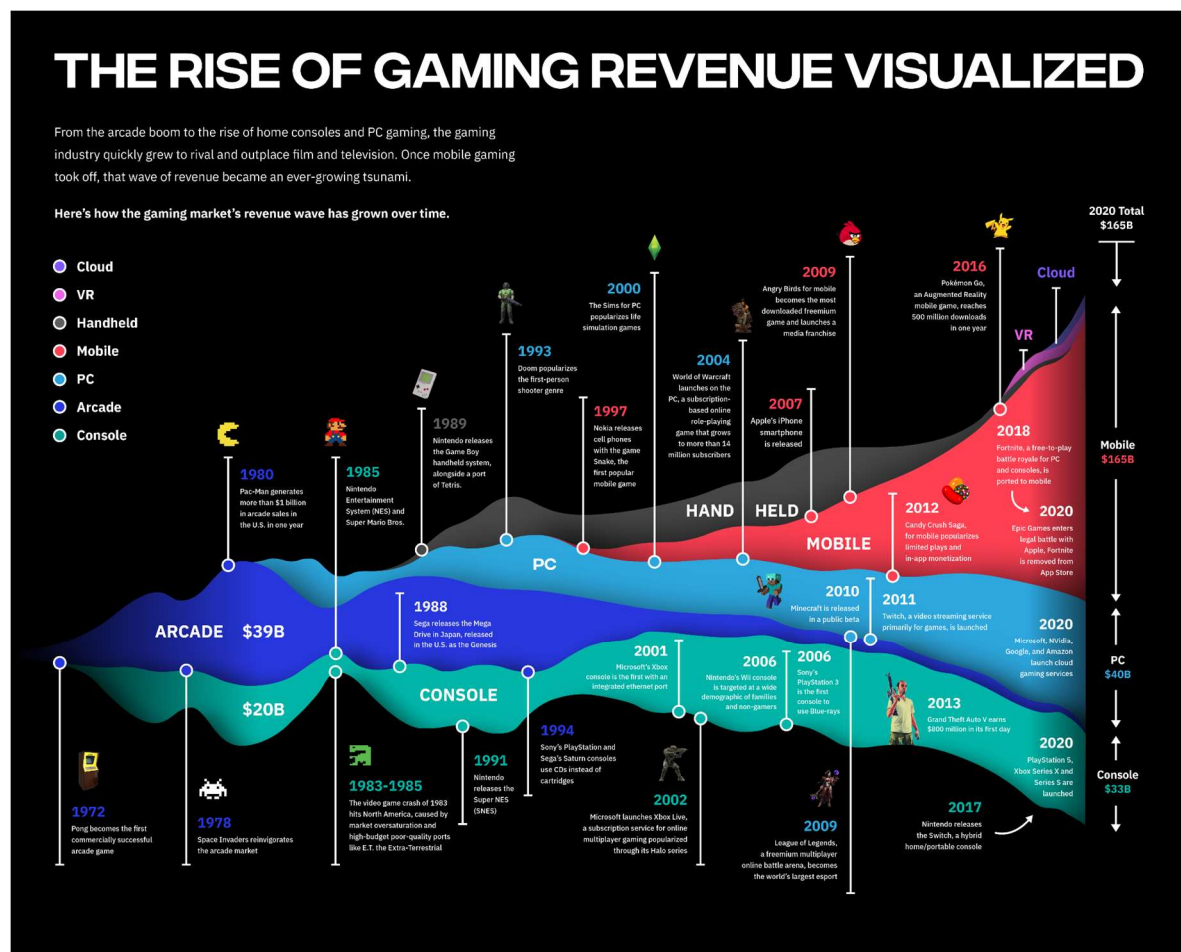


Рисунок 1 – Темпы роста рынка видеоигр

Очевидно, что для обеспечения львиной доли рынка ввиду особенностей платформ требуется классическая клиент-серверная архитектура, а значит необходимо сопровождение огромных баз клиентских данных даже без учета самой игровой архитектуры [2]. По этой причине часть игровых проектов использует уже готовые решения – СУБД. СУБД давно стали неотъемлемой частью при разработке как небольших

приложений, так и крупных IT-проектов. Они позволяют надёжно хранить и изменять данные, обеспечивать к ним параллельный доступ, а также, как правило, предоставляют инструменты для анализа взаимодействия с базой данных. Однако количество данных, которое необходимо обрабатывать, растёт с каждым годом. Помимо этого, растёт также и разнообразие самих данных, что становится достаточно сложной в решении проблемой, поскольку использование одного типа СУБД создаёт ограничения и сложности для реализации и поддержки определенных функций.

Цель данной работы – исследовать применимость мультимодельных СУБД в игровой индустрии и сформировать структуру мультимодельной базы данных для оптимального соответствия потребностям этой отрасли, то есть выполнить описание методологии и её практическую апробацию в условиях, симулирующих реальные.

Задачи:

1. Проанализировать существующие модели представления данных, выделить их преимущества и недостатки.
2. Выполнить обзор существующих мультимодельных СУБД.
3. Рассмотреть общие тенденции и потребности игровой индустрии.
4. Разработать общую структуру мультимодельной СУБД.
5. Провести сравнительный анализ мультимодельной СУБД с одномодельными экспериментальным путём, применив к разным игровым компонентам.
6. Провести анализ полученных результатов.

1 АНАЛИЗ ПОДХОДОВ К ХРАНЕНИЮ ДАННЫХ

Мультимодельные СУБД, несмотря на усложнённые требования к проектированию по сравнению с обычными моделями, сейчас уже активно используются в коммерческой сфере.

Для выявления потенциала использования мультимодельных СУБД над одномодельными в игровой отрасли необходимо провести анализ существующих моделей данных, их преимуществ и недостатков, а также мультимодельных подходов к хранению информации.

1.1 Обзор подходов мультимодельного хранения данных

Данные по своему разнообразию подразделяют на три вида: структурированные, полуструктурированные или вовсе неструктурированные. Для каждого типа данных необходимы свои собственные способы хранения, которые будут наиболее эффективны для конкретных задач.

Существует два способа хранения вариативных данных: «многовариантное хранение» (polyglot persistence) и мультимодельные системы.

Многовариантное хранение представляет собой идею одновременного использования нескольких СУБД для решения задач, где данные будут храниться в подходящем для них месте — отдельных СУБД (рисунок 2).

Одним из способов реализации такой идеи является разделение доменного уровня приложения на несколько подуровней, таким образом, чтобы каждый подуровень обращался к собственной базе данных. При этом на уровне верхнего домена возникает задача о соблюдении согласованности транзакций [3]. Однако у такого способа организации есть большое количество существенных недостатков. Во-первых, в случае

имплементации такой идеи будет сложно реализовать один единый способ обращения к хранящимся данным, производить их обработку, поскольку за разные части отвечают разные системы. Ситуация усложняется, если данные сильно связаны. Во-вторых, такой подход не позволяет в полной мере обеспечить транзакционность и все её свойства. В-третьих, при увеличении объёма используемых СУБД растёт нагрузка на поддержание стабильной работы БД, т.е. усложняется её масштабирование, затраты на обеспечение оптимальной скорости доступа к данным и т.д.

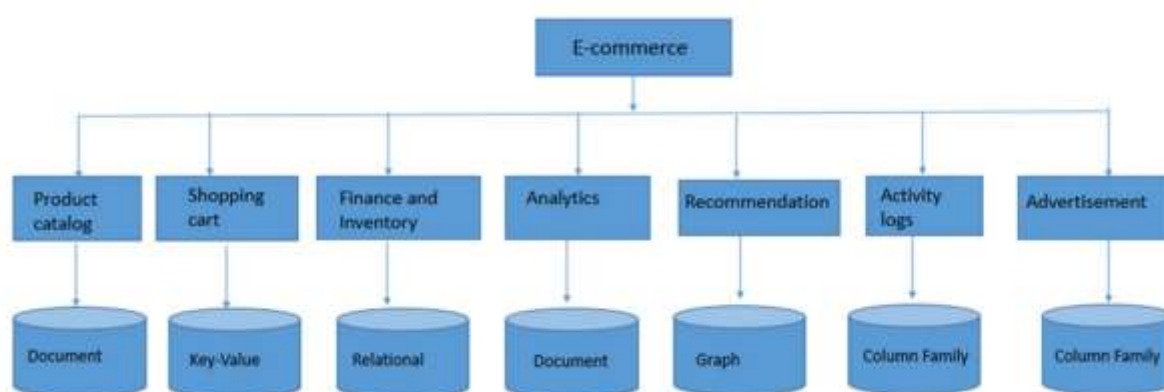


Рисунок 2 – Пример представления данных при соблюдении многовариантного хранения [3]

Вторым подходом к хранению разнообразных данных являются мультимодельные СУБД. Основное преимущество таких СУБД состоит в том, что в пределах одной системы можно поддерживать множество различных представлений данных.

1.2 Анализ существующих моделей СУБД

На основе статей [4, 5, 6, 7] были выделены несколько основных типов по категориям, на основе которых создаются мультимодельные СУБД.

По расположению данных СУБД делятся на:

- централизованные;

- распределённые.

Централизованные СУБД — это СУБД, которые хранят и взаимодействуют со всеми своими данными на одном устройстве. При этом доступ к ним может производиться либо через «файл-сервер», либо через отдельный выделенный сервер. Однако на текущий момент архитектура «файл-сервер» считается устаревшей и взамен используется клиент-серверная архитектура, в основе которой лежит взаимодействие с клиентом через запросы на чтение и запись к серверу. Такой сервер помимо хранения информации также занимается её предварительной обработкой в соответствии с полученным запросом клиента.

Распределённые СУБД — это следующий шаг в развитии централизованных СУБД. Все их данные хранятся на распределённых серверах, но для конечного пользователя этот факт скрыт.

По модели организации данных СУБД делятся на:

- 1) Иерархические СУБД. В основе организации данных таких СУБД стоит древовидная структура, узлы которой связаны между собой отношениями «родитель-ребёнок»: у каждого родителя может быть множество узлов-детей, но у каждого узла-ребенка может быть только один родитель.
- 2) Сетевые СУБД. По структуре напоминают иерархическую, однако представление данных в таких СУБД организовано в виде сети: каждому узлу может соответствовать множество вышестоящих узлов. Это позволяет переходить от одного узла к другому не только вертикально, но и в пределах одного уровня.
- 3) Реляционные СУБД. Все данные таких СУБД упорядочены по строкам и столбцам в таблицах, которые могут быть связаны с другими с помощью одного или нескольких полей. На данный момент Реляционные СУБД являются наиболее популярной моделью организации данных [8].

- 4) Графовые СУБД. Представляют собой организацию данных в виде связного графа, где узел хранит саму сущность, а ребра — связи между ними.
- 5) Документные СУБД. Данные в такой СУБД хранятся в виде документов — сущностей, каждая из которых можно иметь уникальную структуру и содержать разное количество полей.
- 6) Key-value СУБД, в которых данные представлены в виде пары ключ-значение, где каждый ключ является уникальным идентификатором, связанным с конкретной информацией.
- 7) Wide-Column СУБД. Работает по принципу Key-value СУБД, однако каждое «значение» может содержать несколько столбцов и соответствующие им значения, что позволяет удобно хранить связанную информацию. При этом у каждого «значения» может быть свой собственный набор столбцов.

Отдельно стоит отметить обобщающий термин «базы данных NoSQL» или, по-другому, нереляционные СУБД. Такие базы данных не обладают характеристиками, свойственными реляционным СУБД, а также в них не используется стандартный подход с созданием таблиц [9]. У них может не быть жесткой структуры. Под эту категорию подходят графовые, документы, key-value и колоночные СУБД.

1.3 Анализ существующих мультимодельных СУБД

На основе проанализированных статей был составлен перечень СУБД, поддерживающих более одной модели данных [10, 11] (таблица 1).

Таблица 1 – Виды мультимодельных СУБД

Название СУБД	Изначальная модель	Дополнительно поддерживаемые модели
PostgreSQL	Реляционная	Документная, графовая, key-value (расширение HStore)
Oracle	Реляционная	Документная, графовая
MySQL	Реляционная	Key-value
SQL Server	Реляционная	Документная, графовая
Cassandra	Колоночная	Графовая, документная (оба с помощью DataStax)
MarkLogic	Документная	Реляционная, графовая
ArangoDB	Изначально мультимодельная	Документная, графовая, key-value
MongoDB	Документная	Графовая, key-value
CosmosDB	Документная	Графовая, key-value, колоночная, реляционная
Redis	Key-Value	Графовая, документная (оба с помощью аддонов)
OrientDB	Изначально мультимодельная	Графовая, документная, key-value

1.4 Анализ преимуществ и недостатков различных моделей

Для того чтобы определить, какие комбинации при выборе мультимодельных СУБД будут наиболее подходящими, необходимо рассмотреть преимущества и недостатки каждой модели базы данных.

Иерархические базы данных могут обладать быстрой скоростью и эффективностью, когда речь идет о запросе на чтение, поскольку местоположение данных предопределено и структурировано. Поскольку в БД такого типа установлены строгие правила, что у узла-ребенка может быть только один родитель, это позволяет легко соблюдать целостность данных. Однако одним из существенных недостатков такой БД является сложность обновления и масштабирования (поскольку изменения могут затронуть большую часть данных), что делает необходимостью продумывание структуры базы в самом начале затрагивание по возможности только крайних вершин дерева в случае изменения.

Сетевая модель, несмотря на более гибкую структуру, оказалась ещё более сложной в поддержке (как минимум, из-за отсутствия стандартов) и также имела проблемы с масштабированием.

Иерархические и сетевые модели представления данных со временем были признаны устаревшими и почти не используются [12].

Одним из ключевых моментов реляционной модели является наличие строгой структуры каждой таблицы и обеспечение целостности данных за счёт первичных и внешних ключей, а также пользовательских ограничений на этапе построения таблиц. Помимо этого, СУБД обладает достаточным количеством стандартов и общепринятых правил, что сильно упрощает разработку. Принципы ACID — один из таких стандартов, состоящих из набора требований, обеспечивающих сохранность и валидность данных, содержащихся в БД. Реляционные СУБД поддерживают мощный и гибкий язык запроса SQL, позволяющий достать и обработать требуемые данные.

Большим недостатком реляционной модели является ухудшение производительности во время сложных операций, состоящие из нескольких подзапросов, например, использующих функции агрегации или неоптимальным образом соединяющих большое количество таблиц. Такая проблема особенно заметна с увеличением объёма содержащихся данных. Поэтому в случае ожидания большой нагрузки от запросов, необходимо заранее задуматься о возможных оптимизациях. Кроме этого, недостатком системы также можно отметить сложность изменения таблиц и добавления новых столбцов, поскольку, во-первых, это скорее всего потребует также изменить данные в приложении, которое обращается в БД, а также необходимо будет мигрировать существующие данные, что требует определенных навыков и может занять довольно продолжительное время [13].

При рассмотрении графовой модели ясно, что её применимость относительно реляционной модели сильно меньше, однако в ситуациях с высокосвязными данными, например в проектировании социальных взаимодействий, такая система позволяет наиболее эффективно проводить обработку, анализ и распознавать тенденции. На их основе можно создавать рекомендательные системы, а также использовать для определения наиболее быстрых маршрутов. Также графовая модель демонстрирует высокую производительность при локальном чтении во время перемещения по графу [14].

Из недостатков графовой модели можно выделить отсутствие способа запрашивать данные с помощью декларативных языков, только некоторые производители позволяют это проводить. Помимо этого, графовые модели сложно делить для оптимизации запросов, чтобы запросы не касались сразу нескольких поделённых частей графа, хотя это важно, если мы рассматриваем горизонтальное масштабирование системы.

Документные модели представления данных обладает очень высокой производительностью и возможностью масштабирования, поскольку формат хранимых данных спроектирован таким образом, чтобы его можно было найти и достать за как можно меньшее время. При этом документные модели достаточно гибкие, поскольку не имеют структуры и представлены в виде JSON-объекта.

Недостатком такой модели является ограниченная поддержка транзакций. Некоторые из документных СУБД позволяют использовать транзакции, однако они не обладают всеми принципами ACID, как это сделано в реляционных СУБД. Документные модели также неэффективны, если данные приходится часто обновлять, поскольку для этого потребуется заново сохранить всю структуру [15].

В документной модели, несмотря на возможность хранить в одной коллекции объекты с разным набором свойств, также возникает проблема при составлении связей между объектами из разных коллекций. Для того чтобы хранить связи можно создать из двух коллекций одну и дублировать общие данные в каждой отдельной записи. Однако при увеличении количества записей изменения общей части резко станут неэффективными, так как придётся перезаписывать каждый дубликат общей части. Данные также могут обновиться не во всех записях, поскольку гарантии на это нет. Второй способ хранения связей предполагает хранение ссылок на другие коллекции. Представим коллекцию `users`, которая хранит ссылки на купленные товары для каждого пользователя, то есть идентификаторы из коллекции `orders`. В таком случае данные не потеряются. В случае если понадобится получить всех пользователей, которые купили конкретный товар, то хранить ссылки необходимо будет ещё и в коллекции `orders`, что в будущем может создать проблему несогласованности данных.

Key-value модели хороши тем, что являются простым форматом, который может хранить любую информацию, будь то примитив, JSON (это

уже ближе к документной модели) или бинарные данные. Данные легко масштабируются и идеально подходят для параллельной обработки. При этом, как и в случае с документными моделями, они имеют высокую производительность при поиске конкретных ключей. Однако, эта производительность сильно падает, если необходим поиск по нескольким ключам [16].

Wide-Column модели сохраняют преимущества key-value модели, а также предоставляют удобный способ деления участков данных на столбцы. При этом сохраняются у нее также и все недостатки key-value.

1.5 Структурированное сравнение различных моделей

Таблица 2 – Сравнение моделей СУБД

Название СУБД	Преимущества	Недостатки
1	2	3
Иерархические	Быстродействие при заранее определенной структуре	Сложность обновления и масштабирования
Сетевые	Быстродействие при заранее определенной структуре, гибкость	Увеличенная сложность поддержки работоспособности, масштабирования
Реляционные	Принципы ACID, поддержка SQL, структурированность	Сложность редактирования структуры таблиц, медлительность при соединении таблиц

Продолжение таблицы 2

1	2	3
Графовые	Быстродействие с высокосвязными данными	Отсутствие стандартов, сложность (или невозможность) горизонтального масштабирования
Документные	Высокая производительность при поиске через ключи, неструктурированность	Отсутствие транзакционности, медлительность при частом обновлении, отсутствие механизма связей сущностей
Key-value	Высокая производительность при поиске через ключи, неструктурированность	Те же, что и у документной; медлительность при поиске по нескольким ключам
Wide-Column	Высокая производительность при поиске через ключи, полуструктурированность	Те же, что и у Key-value

1.6 Анализ мультимодельных подходов к СУБД

Наибольшая часть мультимодельных СУБД в своей основе имеют реляционную либо документную модели, при этом СУБД, основанные на

реляционной модели, поддерживают большее количество других моделей по сравнению с остальными. Это связано с тем, что, во-первых, реляционные СУБД обладают стандартом SQL. Он, в свою очередь, за последние 20 лет накопил большое количество расширений, позволив с выходом в 2003 году работать с XML-данными, и не так давно — ещё и с данными формата JSON (в стандарте ISO/IEC 9075:2023 от 2023 года), что позволяет организовать документную модель. Работа с XML и JSON — не единственные расширения стандарта, он предоставляет возможность работать и с другими типами данных. Согласно статье [11] тот факт, что реляционные СУБД универсальны и просты, позволяет им расширять поддержку других моделей представления данных, становясь таким образом мультимодельными.

Однако такой подход не является полностью мультимодельным, а скорее его комбинацией с многовариантным хранением, потому что большую часть данных всё равно приходится обрабатывать за пределами СУБД. Поскольку изначально СУБД создавалась для работы с одной (реляционной) моделью хранения данных, применять другие подходы требует больших усилий и громоздких запросов.

По этой причине направленность работы будет уходить в сторону исследования эффективности изначально мультимодельных СУБД. Преимущество таких баз данных заключается в том, что они позволяют использовать единый формат обращения к данным, и при этом имеют готовые решения для хранения определённых структур. К тому же часть подобных БД также позволяют использовать принципы ACID.

1.7 Обзор изначально мультимодельных СУБД

Наиболее популярными СУБД, которые изначально проектировались как мультимодельные являются CosmosDB, OrientDB и ArangoDB.

CosmosDB – это глобально распределённая БД, поставляющаяся в виде полностью облачной базы [17]. Одной из её главных особенностей является доступ через несколько API, она одновременно поддерживает SQL, MongoDB, Cassandra, Gremlin и прочие. Все данные хранятся в специальных контейнерах, которые в свою очередь состоят из элементов.

Из существенных недостатков можно выделить отсутствие бессрочного бесплатного доступа. Помимо этого, CosmosDB не поддерживает возможность слияния нескольких таблиц (контейнеров). Из-за того, что выбрать API можно только при создании аккаунта, не получится запросить данные, сохранённые одним API через другой, что по сути является близким аналогом многовариантного хранения.

OrientDB и ArangoDB по своей структуре сильно отличаются от CosmosDB. Они позволяют хранить данные в документной и графовой модели данных, а также поддерживают принципы ACID и транзакции. OrientDB для доступа к данным использует диалект SQL, что упрощает обучение новых пользователей. OrientDB отличается возможностью поддержки классов с свойствами и наследованием.

ArangoDB для запросов использует свой собственный язык – AQL (ArangoDB Query Language). За счёт того, что этот язык не основывается на SQL и создан специально для мультимодельных задач, это добавляет ему гибкости при обращении к данным разного типа. Кроме того, СУБД отличается полнотой документации и поддержкой драйверов под различные языки программирования. Отличительной особенностью ArangoDB является наличие поискового движка ArangoSearch, который в том числе работает с русским языком.

Данные в ArangoDB хранятся либо в обычных коллекциях, как при документной модели данных, либо в коллекциях рёбер (edge collections). С помощью последних реализуется связь между элементами нескольких

коллекций. Связи при этом могут хранить дополнительные данные, которые затем можно использовать в запросах.

Отдельно стоит отметить анализ производительности, проведённый разработчиками ArangoDB [18]. Тестирование на чтение, запись, агрегацию, занимаемую память, а также на специфичные для поиска по графам запросы показали значительное преимущество по отношению к OrientDB (рисунок 3). Это также подтверждается в статье [19].

NoSQL Performance Benchmark 2018

Absolute & normalized results for ArangoDB, MongoDB, Neo4j and OrientDB

	single read (s)	single write (s)	single write sync (s)	aggregation (s)	shortest (s)	neighbors 2nd (s)	neighbors 2nd data (s)	memory (GB)
ArangoDB	100%	100%	100%	100%	100%	100%	100%	100%
3.3.3 (rocksdb)	23.25	28.07	28.27	01.08	0.42	1.43	5.15	15.36
ArangoDB	102.16%	102.55%	103.89%	102.40%	816.06%	122.07%	99.32%	92.87%
3.3.3 (mmfiles)	23.76	28.79	29.37	1.10	3.40	1.75	5.12	14.27
MongoDB	422.38%	1123.36%	1652.09%	136.65%		518.83%	192.88%	50.64%
3.6.1 (Wired Tiger)	98.24	315.33	466.99	1.47		7.42	9.94	7.70
Neo4j	153.65%		149.37%	203.45%	199.94%	208.96%	214.22%	240.68%
3.3.1	35.73		43.22	2.18	0.83	2.99	11.04	37.00
Postgres	231.17%	129.03%	127.70%	29.62%		307.96%	76.87%	26.68%
10.1 (tabular)	53.77	36.22	36.10	0.32		4.41	3.96	4.10
Postgres	135.96%	104.34%	101.55%	204.55%		292.57%	126.14%	35.36%
10.1 (jsonb)	31.62	29.29	28.70	2.20		4.19	6.50	5.43
OrientDB	198.84%	110.37%		2526.29%	12323.67%	636.45%	400.97%	107.04%
2.2.29	46.25	30.98		27.19	51.34	9.11	20.67	16.45

Рисунок 3 – Тест на производительность от ArangoDB

Учитывая всё вышесказанное для анализа применимости мультимодельных СУБД в игровой отрасли было решено использовать ArangoDB.

2 АНАЛИЗ ПОТРЕБНОСТЕЙ ИГРОВОЙ ИНДУСТРИИ

Для исследования применимости мультимодельных СУБД в игровой сфере, для создания более приближенных к реальности экспериментов необходимо детально проанализировать потребности игровой индустрии.

2.1 Обзор жанров видеоигр

Стоит начать с того, что игровые проекты делятся на однопользовательские и многопользовательские. Согласно отчёту, собранного компанией Unity в 2022 году [20], 77% всех геймеров предпочитают многопользовательские игры. Рассмотрим несколько наиболее популярных жанров видеоигр с мультиплеерной направленностью:

- 1) Королевская битва — жанр игры, подразумевающий сессионные сражения большого (около 100) количества людей, которых помещают на одну игровую арену и заставляют соревноваться друг с другом. Последний оставшийся игрок объявляется победителем.
- 2) Action-RPG — подвид компьютерных игр, являющиеся ролевыми (RPG, Role Play Games) и сочетающие в себе элементы игр жанра Action (как правило возможность ведения боя в реальном времени).
- 3) MMORPG (Massive Multiplayer Online Role Play Games) — ролевые компьютерные игры, направленные на одновременное нахождение огромного числа пользователей в одном виртуальном мире и основанные на взаимодействии как с самим миром, так и друг с другом.
- 4) Стратегии — игры, которые предполагают от игроков использования стратегического или тактического планирования и, как правило, контроля над несколькими интерактивными объектами.

- 5) Tower Defense — поджанр стратегических игр, где в основе лежит защита определенной территории от врагов, а также постепенное окружение безопасной местности различными структурами, которые могут блокировать, атаковать врагов или выполнять какое-то отличное действие. Цель игры, как правило — простоять определённое время либо уничтожить всех нападающих.
- 6) Шутеры — поджанр Action-игр, в котором главной целью ставится победа над врагами с помощью определённого набора оружия и снаряжения; является очень динамичным видом игр.
- 7) Адвенчуры — приключенческие игры, подразумевающие прохождение игроком сюжета посредством квестов, применений игровых предметов и разговором с неигровыми персонажами.

2.2 Анализ компонентов видеоигр

Согласно отчёту компании Newzoo [21], топ жанров по доходу возглавляют шутеры, адвенчуры и RPG (рисунок 4). За последние 20 лет в рамках этих жанров было создано большое количество игр, что позволяет на их основе выделить наиболее часто используемые компоненты и механики.

В большинстве играх этих жанров возникает необходимость хранить различные данные об игровом мире и отправлять их игрокам по необходимости. Такими данными являются: NPC, ареалы спауна существ и предметов, зарытые сокровища и так далее. Общее название для них – точки интереса (PoI). Они могут быть сгруппированы по внутриигровым городам или регионам для оптимизации запросов на сервер.

В RPG-играх часто имеется возможность сражаться с существами. Каждый вид существ обладает набором одинаковых характеристик: очки здоровья, урон, тип (дружественный, нейтральный или враждебный), а также настроенную таблицу добычи.

Top genres by revenue on the major gaming platforms in 2023

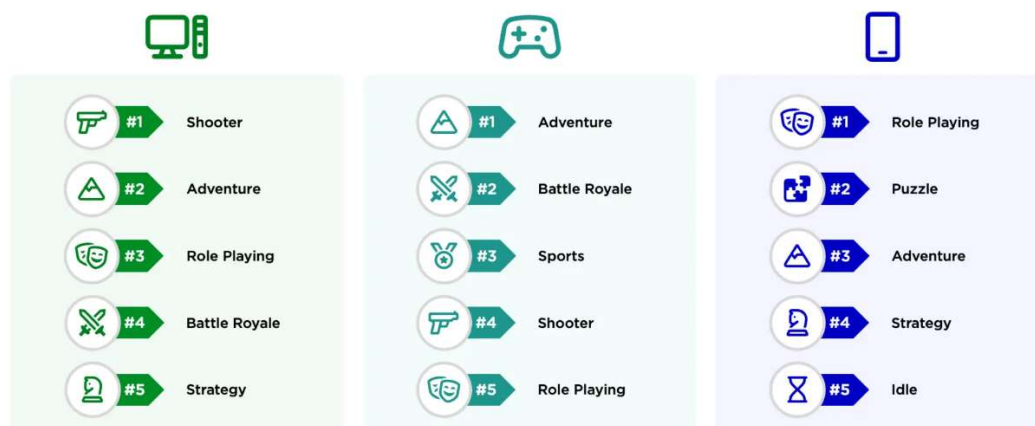


Рисунок 4 – Топ жанров игр 2023 года по доходу

Таблицы добычи (loot tables) определяют, какую награду и с каким шансом игрок получит за определённое достижение, выполнение квеста, убийство существа или любое другое действие. Сама по себе таблица состоит из идентификаторов предметов, которые могут выпасть, а также частоты их выпадения.

Одним из компонентов, широко используемым в RPG, адвенчурах и шутерах является система квестов (рисунок 5). Квесты, а точнее выданные игроку задания, позволяют отслеживать прогресс, вести игрока, а также записывать важные для нарратива детали. Квесты могут содержать как одну, так и несколько задач, например, «выковать меч у кузнеца» и «купить яблоко на рынке». Квесты могут быть повторяющимися или неповторяющимися, они как правило, имеют название, описание, а их выполнение может давать игроку награду и/или опыт. Квесты могут быть объединены в цепочки, таким образом по завершении одного квеста игроку выдаётся новый.

Квесты связаны ещё с одной важной для вышеперечисленных жанров системой – системой диалогов. Диалоги состоят из инструкций – запрограммированных последовательностей из действий, реплик NPC,

катсцен и т.д. Если добавлять в систему ответные фразы игрока, в том числе имея возможность выбрать вариант из списка, то в таком случае диалоги могут расходиться на несколько веток, каждая из которых ведёт к разному результату, создавать циклы или сходиться обратно к одному общему узлу, т.е. будут нелинейными.



Рисунок 5 – Отображение квестов в ММО

Неотъемлемой частью многих современных игр является инвентарь. Он представляет собой список предметов, принадлежащий конкретному игроку, существу или объекту в мире. Предметы состоят из уникального имени и метаданных. При этом стоит различать определение предмета (definition) и его экземпляр. Определения, как правило, хранятся отдельно, и содержат такую информацию как название, описание, характеристики и прочие параметры, одинаковые для всех экземпляров, созданных из этого определения. Инвентарь же состоит из экземпляров, в которых помимо ссылки на определение имеется ещё и уникальные данные – пользовательское название (как в Minecraft), редкость (как в Counter Strike 2).

Часто в игровые проекты также добавляют возможность создавать предметы по рецепту или улучшать их с помощью компонентов, превращая

в другие, как в Fallout 76 [22]. Две эти системы можно объединить, если посчитать, что во второй системе объект улучшения – это всего лишь один из ингредиентов для создания другого. При этом для изготовления предмета могут потребоваться: а) разные типы ингредиентов, б) разное количество ингредиентов. Пример того, как выглядит рецепт в игре Factorio представлен на рисунке 6.



Рисунок 6 – Пример рецепта в Factorio

Системы с множеством компонентов, предполагающие их комбинацию, нечасто, но присутствуют в играх подобных жанров. Это может быть, например, система заклинаний, как в World of Warcraft, или артефактов, как в The Binding of Isaac Rebirth. Действие заклинания или артефакта может происходить в какой-либо области, иметь цель или не иметь её, работать только с определёнными объектами, быть особой формы (луч, сфера и прочее), накладывать определённый эффект и т.д. [23, 24]. При этом каждый компонент может обладать разными по своему составу параметрами. Например, область характеризуется радиусом поражения, эффекты могут иметь разную длительность, мощность. А некоторые комбинации могут провоцировать уникальные эффекты – синергии. Так,

например, комбинация артефактов «The Ludovico Technique» и «Brimstone» из The Binding of Isaac Rebirth превращает луч от «Brimstone» в атакующий полый круг.

В играх с большим количеством контента дизайнеры прибегают к созданию специальной внутриигровой энциклопедии. Её главная задача помочь игроку найти информацию по любой существующей в игре вещи, будь то локация, монстр или объяснение внутриигровой механики. Как правило, она подразделена на категории, а также имеет окно для поиска, позволяющее найти что-либо по ключевым словам, как, например, в серии игр Civilization (рисунок 7).



Рисунок 7 – Внутриигровая энциклопедия игры Civilization

Помимо геймплейных компонентов, игровые проекты зачастую включают в себя системы, повторяющиеся в играх вне зависимости от

жанра, поскольку носят либо утилитарный характер, либо необходимы для внутриигрового социального взаимодействия.

Одной из утилитарных систем является сбор статистических данных. Статистика должна позволять отвечать на такие вопросы как «Какое количество пользователей зашло за прошедшие сутки, месяц?», «В какой период (в часах, в месяцах) вырастает или, наоборот, снижается общий уровень посещаемости?», «Какой процент пользователей и через какое время перестаёт проявлять интерес к игре?», а также выдать аналитику по полу, возрасту, стране проживания и прочим социальным факторам.

В статистику также могут входить данные об игровом процессе. Это могут быть показатели прогресса (количество выполненных квестов, число побед, поражений, уровень персонажа), качество и количество социальных контактов (общительность в чате, число нарушений и прочее).

Статистические данные часто применяются для систем монетизации: персонализированной рекламы, внутриигровых покупок, реферальной системы, уникальных предложений внутриигрового магазина, аналитики, продвижения и прочего [2]. Они также позволяют вовремя среагировать на возможную угрозу, вынести решение о недопуске недобросовестных игроков к определённым режимам, а также для того, чтобы улучшить игровое восприятие.

Помимо этого, в данном разделе хранится информация о внутриигровых ошибках, логах, сбоях, игровых аккаунтах, а также опросах и пользовательских жалобах.

Системами социального взаимодействия в играх могут быть доски лидерства (leaderboard), список друзей, кланы или другие внутриигровые сообщества.

Отдельно стоит отметить возможность подбора игроков схожего уровня для совместного прохождения подземелья или поиск пользователей с тем же рейтингом, рангом в качестве соперников матча. Это важная

деталь, поскольку зачастую столкновение игроков с разным уровнем игрового опыта ухудшает пользовательский опыт менее натренированного игрока. Однако есть специальный жанр игр, а именно королевские битвы, где это является основной идеей.

2.3 Особенности игровой индустрии

Часть систем и компонентов игрового мира можно хранить на клиентской стороне, в особенности, когда речь идёт об однопользовательской игре. Однако в таком случае возникает проблема, что любое изменение этих данных заставит разработчиков заново компилировать файлы игры, а игроков – загружать это обновление, и, как следствие, перезапускать игру и заново подключаться к серверам. Помимо этого, некоторые игры позволяют создавать свои собственные сервера с изменённым или уникальным контентом. В таком случае одним из способов обновления контента является загрузка специальных контент-пакетов. Например, компания Mojang для своей игры Minecraft создала систему «датапаков», благодаря которой сервера могут настраивать такие компоненты как таблицы добычи (loot tables), рецепты, достижения, а также создавать собственные измерения [25]. Игра на клиентской стороне при подключении к серверу загружает все изменения по специальному протоколу.

Необходимо учитывать особенности некоторых жанров игры, таких как MMORPG. Для того чтобы поддерживать большое количество (от нескольких тысяч до 2-3 миллионов) одновременно играющих пользователей, игры с таким количеством игроков делят общее пространство на отдельные области, каждая из которых находится на собственном сервере. При этом деление может происходить, например, по реальной или внутриигровой позиции пользователя [26].

Как подчёркивает автор статьи [27] реляционные БД в их стандартном виде неудобны при хранении игровых объектов в их стандартном виде, поскольку для выстраивания связей необходимо будет создавать транзакцию на несколько таблиц. При этом таблиц, которые потребуются в запросе, может быть больше десяти, что накладывает отпечаток на скорость и усложняет поддержку правильной синхронизации данных. Автор предлагает использовать BLOB-объекты (сериализованные бинарные объекты), которые содержат сильно связанные свойства, и хранить их в одной из колонок реляционных СУБД.

Для хранения утилитарной информации, такой как, например, данные аккаунтов наилучшим образом будет с точки зрения Abdullah Alqwbani и др. [28] использование реляционной модели, поскольку данных не так много, все они должны храниться наиболее безопасным образом и не требуют большой аналитики. Для данных же игрового мира было предложено использовать NoSQL-подход, поскольку реляционная модель менее производительна при масштабировании.

2.4 Существующие решения по хранению игровых данных

Несмотря на потенциальные преимущества мультимодельных СУБД над одномодельными базами данных, не было найдено публичных примеров их использования в игровых проектах. Компании в своих продуктах как правило используют одномодельные базы данных. Большая часть проект работает с реляционной моделью [29], например, такие проекты как Eve Online [30] или World of Warcraft [31].

Стоит отметить, что Activion Blizzard, компания, создавшая World of Warcraft, выпустила блог, в котором отметила недостатки своей архитектуры системы заклинаний [31]. По итогу они разделили одну таблицу с большим количеством потенциально пустых полей на несколько новых таблиц. С одной стороны, таким образом они нормализовали часть

своей базы данных. С другой стороны, из-за сложности самой системы, где у каждого заклинания может быть несколько отличающихся друг от друга эффектов, усложнился способ их получения. Одной из проблем реляционных СУБД является сильное снижение производительности при соединении большого количества таблиц.

При этом некоторые игровые проекты успешно работают и с NoSQL-хранилищами, например, компания Epic Games для хранения данных в игре Fortnite использует MongoDB [32].

Другим вариантом хранения информации в игровой индустрии, является использование частично мультимодельного подхода, за счёт возможностей одномодельных баз данных. При рефакторинге проектов Аллоды и Skyforge, разработчики решили поменять реляционную базу данных, сменив MySQL на PostgreSQL. Они отмечали, что реляционная модель имеет ряд значимых недостатков, которые можно закрыть только используя связку реляционных возможностей с нереляционными. Таким образом было решено создать несколько столбцов, работающих в качестве хранилищ для данных в документном стиле, поскольку эти данные сильно связаны друг с другом и редко меняются по отдельности [33]. Например, координаты, направление взгляда, а также несколько дополнительных параметров теперь хранятся в столбце position в виде JSON-объекта.

Таким образом можно заметить, что в игровой отрасли существует запрос на использование нескольких моделей одновременно, однако использование одной модели данных внутри другой не позволяет по максимуму использовать обе, а также затрудняет обработку (как минимум, сериализацию/десериализацию) данных, поскольку она производится в двух местах – внутри базы данных и в отдельном приложении.

3 ПРОЕКТИРОВАНИЕ МУЛЬТИМОДЕЛЬНОГО ХРАНИЛИЩА

Главными гипотезами в пользу использования мультимодельных СУБД являются два тезиса:

1. Использование мультимодельных СУБД ускоряет работу инфраструктуры из-за того, что они позволяют представлять данные в том виде, в котором они будут доступны более быстрым способом.
2. Поддержка большого количество разных СУБД очень сложна, так как требует большое количество специалистов.

Для того чтобы продемонстрировать возможности мультимодельной СУБД в рамках игровой отрасли, создадим пример возможной базы данных вкупе с часто используемыми запросами к ней. Предлагается рассмотреть также примеры запросов к спроектированной базе данных, чтобы было ясно преимущество использования мультимодельного подхода.

Возможности создания схемы данных по аналогии с привычной реляционной схемой данных в ArangoDB не предусмотрено по причине отсутствия нотации для моделирования мультимодельных БД. Объекты разработанной базы данных представлены в приложении А.

В качестве примера подробно изучим компоненты игр, типичные для жанров RPG и MMORPG.

3.1 Проектирование основных компонентов хранилища

Одними из основополагающих в базе данных будут являться коллекции `items` и `players`.

Коллекция `items` необходима для хранения определений предметов. Определение предмета состоит из уникального идентификатора, его названия либо ключа перевода, описания, а также любой метаинформации, которая будет полезна для создания экземпляров этого предмета, например,

качество предмета или прочность у инструментов и оружия (рисунок 8). Экземпляр предмета – это конкретный объект, который имеет все свойства, указанные в определении предмета, но представляет собой его воплощение.

```
1 {  
2   "name": "Iron Sword",  
3   "desc": "The Sword to make enemies cry!",  
4   "meta": {  
5     "durability": 260  
6   }  
7 }
```

Рисунок 8 – Пример документа коллекции items

Ключевым звеном в структуре будет являться коллекция players, поскольку большая часть данных в шутерах, адвенчурах и RPG центрированы на игроке. Ключом будет никнейм, поскольку он уникален, а сам документ в этой коллекции может включать в себя инвентарь (список предметов, который носит игрок и использует по мере надобности), параметры (уровень, количество опыта, здоровья, позиция и т.д.), открытые навыки, активные эффекты и прочее (рисунок 9).

```
_id:   players/Time_Conqueror  
_rev:  _h2Pxxz----  
_key:  Time_Conqueror  
  
1 {  
2   "inventory": [  
3     {  
4       "item": "items/stone",  
5       "count": 5,  
6       "data": {}  
7     },  
8     {  
9       "item": "items/wood",  
10      "count": 6,  
11      "data": {}  
12     }  
13   ],  
14   "params": {  
15     "pos": [  
16       100,  
17       200,  
18       300  
19     ],  
20     "level": 54,  
21     "experience": 58000  
22   },  
23   "online": true  
24 }
```

Рисунок 9 – Документ коллекции players

3.2 Проектирование системы рецептов с помощью графовой модели

Теперь воспроизведём одну из популярных игровых механик, предложенных в одной из предыдущих глав: систему рецептов. Для этого создадим коллекцию рёбер (edge collection) с названием `recipes`, которая будет содержать в себе связи между предметами из коллекции `items` с направлением от ингредиентов к результату рецепта. Поскольку некоторые рецепты могут требовать не единственный экземпляр предмета, а несколько, необходимо также сохранять их количество внутри этого связующего ребра. Поскольку ингредиенты рецепта могут отличаться не только количеством, но и типом (определением предмета), для каждого ингредиента нужно создавать связывающее ребро.

ArangoDB имеет уникальные возможности визуального представления объектов БД в различных форматах. Способ визуализации объекта можно выбрать в зависимости от текущей задачи, стоящей перед разработчиком. Так, например, объект `recipes` можно представить как в виде графа, так и в виде, характерном для документной модели (рисунки 10 и 11).

Content

```
{ "_from": "items/flint", "_id": "recipes/2361", "_key": "2361", "_rev": "_h1h5P_W---", "_to": "items/workbench", "count": 3 }

{ "_from": "items/planks", "_id": "recipes/5286", "_key": "5286", "_rev": "_h1h5x1G---", "_to": "items/slab", "count": 2 }

{ "_from": "items/wood", "_id": "recipes/2307", "_key": "2307", "_rev": "_h1h53jC---", "_to": "items/planks", "count": 1 }

{ "_from": "items/planks", "_id": "recipes/2335", "_key": "2335", "_rev": "_h1h58M0---", "_to": "items/workbench", "count": 6 }

{ "_from": "items/workbench", "_id": "recipes/8984", "_key": "8984", "_rev": "_h1iuyEC---", "_to": "items/superbench", "count": 1 }
```

Рисунок 10 – Документное представление объекта `recipes`

Наиболее часто будут использоваться следующие запросы: «Получить все предметы, которые можно создать из имеющегося набора ингредиентов, учитывая их количество», «Получить все рецепты,

содержащие указанный ингредиент», «Получить все ингредиенты, требуемые для создания конкретного предмета». Вышеупомянутые запросы для своего функционирования потребуют использование графовой модели, а также создание дополнительного графа recipes, совпадающем по названию с исходным объектом recipes (рисунок 12).

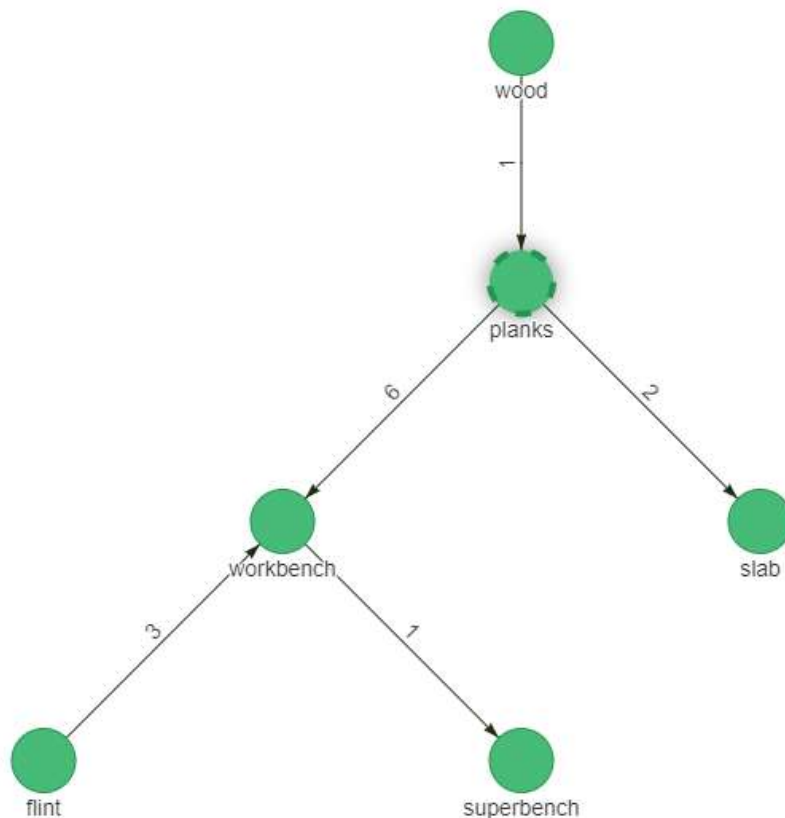


Рисунок 11 – Графовое представление объекта recipes

Name: ⓘ

Relation Add relation

Edge definition*: ⓘ

fromCollections*: ⓘ

toCollections*: ⓘ

Рисунок 12 – Структура графа recipes

На вход запроса для получения доступных рецептов из имеющихся ингредиентов (приложение В) необходимо подавать массив из объектов, состоящих из поля «item» – id предмета и «count» – его количества. На выходе мы ожидаем получить массив найденных рецептов, которые можно сделать из всех указанных ингредиентов. Пример параметров запроса, а также его результат представлены на рисунке 13 и 14 соответственно.

```
1 {  
2   "items": [  
3     {  
4       "item": "items/planks",  
5       "count": 7  
6     },  
7     {  
8       "item": "items/flint",  
9       "count": 5  
10    }  
11  ]  
12 }
```

Рисунок 13 – Параметры запроса на поиск подходящих рецептов

```
1 [  
2   [  
3     {  
4       "result": "items/workbench",  
5       "ingredients": [  
6         {  
7           "item": "items/planks",  
8           "count": 6  
9         },  
10        {  
11          "item": "items/flint",  
12          "count": 3  
13        }  
14      ]  
15    },  
16    {  
17      "result": "items/slab",  
18      "ingredients": [  
19        {  
20          "item": "items/planks",  
21          "count": 2  
22        }  
23      ]  
24    }  
25  ]  
26 ]
```

Рисунок 14 – Пример результата запроса на поиск подходящих рецептов

Второй и третий запрос из списка выше также используют поиск по графу, только в качестве параметра запроса там необходимо предоставить идентификатор предмета (приложение С).

3.3 Проектирование системы таблиц добычи с помощью документной модели

Ещё один компонент, который стоит рассмотреть в системе ArangoDB – таблицы добычи. Для их хранения требуется документная модель без использования графовой. Благодаря языку запросов AQL, мы можем полностью переложить ответственность за расчёт добычи на СУБД, без необходимости вытаскивания таблиц добычи целиком и их расчёта за пределами базы данных.

Таблицы добычи состоят из пулов (pool), то есть множеств, из которых будут выделены случайные предметы. Пулы никак не связаны друг с другом и высчитываются отдельно, а результат расчёта (итоговые предметы) затем объединяется с остальными из других пулов. При этом пул имеет свойство «rolls», означающее количество «бросков кубика», то есть сколько раз будет выбран предмет из этого пула. Пул также содержит свойство «entries», представляющее собой массив предметов, которые могут выпасть. Каждый элемент содержит тип предмета («item»), количество («count»), вес («weight»). Изначально каждый предмет рассчитывает префиксную сумму, таким образом занимая интервал значений по формуле:

$$w_{pr} = w_i \sum_{j=0}^{i-1} w_j \quad (1)$$

где w_i – это вес элемента i .

Затем из результата формулы (1) берётся случайное число от 0 до w_{pr} последнего элемента и ищется интервал, в котором это случайное число окажется, таким образом определяя выбранный предмет. Пример одной из таблиц добычи указан на рисунке 15.

```

1  {
2    "pools": [
3      {
4        "rolls": 2,
5        "entries": [
6          {
7            "item": "items/planks",
8            "count": 3,
9            "weight": 10
10         },
11         {
12           "item": "items/flint",
13           "count": 1,
14           "weight": 5
15         }
16       ]
17     },
18     {
19       "rolls": 1,
20       "entries": [
21         {
22           "item": "items/wood",
23           "count": 3,
24           "weight": 20
25         },
26         {
27           "item": "items/tree_heart",
28           "count": 1,
29           "weight": 5
30         }
31       ]
32     }
33   ]
34 }

```

Рисунок 15 – Пример результата запроса на поиск подходящих рецептов

Стоит отметить, что в играх не всегда используется строго конкретное число в свойствах «count» и «rolls», довольно часто разработчики указывают минимально и максимально возможное число, однако здесь эти свойства были строго ограничены для упрощения и наглядности.

Наиболее популярными к этой системе будут следующие запросы: «Получить случайный набор ресурсов по настроенной таблице добычи», «Показать шанс выпадения определённого предмета и его количества».

Разберём первый запрос (приложение D): в качестве входных параметр ему необходимо передать ключ таблицы добычи (рисунок 16).

Примеры результата представлены на рисунке 17. Как мы можем заметить на последнем рисунке, из первого пула всегда приходит 2 предмета, а из последнего 1.

Key	Value
loot_table	firewood_ent

JSON

Рисунок 16 – Параметры запроса для получения случайного предмета

```

1  [
2  [
3  {
4    "item": "items/flint",
5    "count": 1
6  },
7  {
8    "item": "items/flint",
9    "count": 1
10 },
11 {
12  "item": "items/tree_heart",
13  "count": 1
14 }
15 ]
16 ]

```

```

1  [
2  [
3  {
4    "item": "items/flint",
5    "count": 1
6  },
7  {
8    "item": "items/planks",
9    "count": 3
10 },
11 {
12  "item": "items/wood",
13  "count": 3
14 }
15 ]
16 ]

```

Рисунок 17 – Примеры результата запроса для получения случайного предмета

Запрос на показ шанса выпадения предмета также часто пользуется популярностью у игроков и может быть частью какой-то внутриигровой энциклопедии или любого другого информационного интерфейса. Тело запроса будет похоже на предыдущий, однако вместо выбора случайного предмета для каждого пула мы будем получать вес конкретного предмета, поделённого на суммарный вес пула, а затем просуммируем результирующие вероятности (листинг 1).

На примере луттейбла на рисунке 18 попробуем получить вероятность выпадения. На входе запроса помимо указания идентификатора таблицы добычи необходимо будет приложить

идентификатор предмета, результатом будет являться дробное число. В случае примера на рисунке 18 при запуске запроса с указанием предмета «items/snow_heart» в качестве параметра, итоговая вероятность будет приблизительно равна 0.11.

```
for table in loot_tables
  filter table._key == @loot_table

let poolWeights = (for pool in table.pools
  let sumWeight = sum(pool.entries[*].weight)

  for e in pool.entries
    FILTER e.item == @item_id
    return e.weight / sumWeight)
return sum(poolWeights)
```

Листинг 1 – Запрос получения случайного предмета

```
1  {
2    "pools": [
3      {
4        "rolls": 2,
5        "entries": [
6          {
7            "item": "items/ice",
8            "count": 1,
9            "weight": 5
10         },
11         {
12           "item": "items/snow",
13           "count": 3,
14           "weight": 10
15         },
16         {
17           "item": "items/snow_heart",
18           "count": 1,
19           "weight": 1
20         }
21       ]
22     },
23     {
24       "rolls": 1,
25       "entries": [
26         {
27           "item": "items/stick",
28           "count": 3,
29           "weight": 20
30         },
31         {
32           "item": "items/snow_heart",
33           "count": 1,
34           "weight": 1
35         }
36       ]
37     }
38   ]
39 }
```

Рисунок 18 – Пример таблицы добычи с дублирующимся выпадающим предметом

3.4 Проектирование системы внутриигровой энциклопедии

Одним из преимуществ ArangoDB является возможность создания специальных анализаторов текста, что может сильно помочь при разработке внутриигровой энциклопедии: ArangoSearch.

Для начала необходимо определиться со структурой: понадобится коллекция `wiki`, которая будет состоять из документов, представляющих собой игровые статьи со свойствами «title» (название), «description» (описание), а также «tags» (теги). Поиск будет производиться по названию и тегам. Теги не являются обязательным свойством, но позволяют привязывать определённые слова или фразы к статье, которая не содержит прямого названия, но является единственным источником по нужной теме. Пример статьи указан на рисунке 19.

```
1 {  
2   "title": "Basics",  
3   "tags": [  
4     "Draconium Ore"  
5   ],  
6   "description": "The very basis of Draconic  
   Research. This ore can be found in the overworld  
   below y-level 8 as well as in The Nether at any  
   y-level, although it is extremely rare in both.  
   You can also find it in much larger quantities  
   in The End, both in the main island and in the  
   comets that randomly spawn throughout The End.  
   Drops 1 - 3 Draconium Dust when mined. This can  
   be increased up to 4 - 12 with the Fortune III  
   enchantment."  
7 }
```

Рисунок 19 – Пример внутриигровой статьи

Для правильного функционирования движка ArangoSearch необходимо также создать представление коллекции: «`wiki_token_view`». В качестве ссылок укажем «`wiki`», а также отметим нужные поля – «`title`» и «`tags`» (рисунок 20).

```

32  "links": {
33    "wiki": {
34      "analyzers": [
35        "identity"
36      ],
37      "fields": {
38        "tags": {
39          "analyzers": [
40            "text_en"
41          ]
42        },
43        "title": {
44          "analyzers": [
45            "text_en"
46          ]
47        }
48      },
49      "includeAllFields": false,
50      "storeValues": "none",
51      "trackListPositions": false
52    }

```

Рисунок 20 – Настройка представления «wiki_token_view»

Теперь можно запрашивать статью, используя фразовый анализатор (листинг 2). Сам запрос принимает любое словосочетание, и если находит совпадение по названию или тегу, то выдаёт нужную статью. Например, по названию «Draconium» показываются все статьи, так или иначе с ним связанные (рисунок 21). Однако если ввести «Ore», то результатом будет статья с рисунка 19.

```

LET p = (
  FOR word IN TOKENS(@text, "text_en")
  RETURN word
)

```

```

FOR doc IN wiki_token_view
  SEARCH PHRASE(doc.title, p, "text_en")
  OR
  PHRASE(doc.tags, p, "text_en")

RETURN {
  title: doc.title,
  description: doc.description
}

```

Листинг 2 – Запрос получения случайного предмета

title	description
Basics	The very basis of Draconic Research. This ore can be found in the overworld below y-level 8 as well as in The Nether at any y-level, although it is extremely rare in both. You can also find it in much larger quantities in The End, both in the main island and in the comets that randomly spawn throughout The End. Drops 1 - 3 Draconium Dust when mined. This can be increased up to 4 - 12 with the Fortune III enchantment.
Draconium Dust	This is the dust dropped when mining Draconium Ore. It can be smelted into Draconium Ingots.
Draconium Ingot	The product of smelting Draconium Dust in a furnace. Used in most Draconic Evolution recipes.

Рисунок 21 – Вывод статей по запросу ArangoSearch

3.5 Проектирование системы подбора игроков

С помощью систем поиска ArangoDB и возможности настраивать индексированные представления коллекций можно реализовать простую (для наглядности) систему подбора игроков по их уровню. Для этого хорошо подойдут запросы в диапазоне, которые также работают с ключевым словом SEARCH, как и в случае с системой энциклопедии. Перед написанием запроса предварительно создадим индексированное представление `player_level_view` (рисунок 22).

Запросом к данной системе будет получение списка игроков, чей уровень будет находиться в диапазоне 5 в обе стороны по отношению к уровню запрашивающего игрока. Стоит учитывать важный критерий – игрок должен быть «онлайн», то есть быть в игре в данный момент. На вход подаётся никнейм игрока, для которого нужно произвести подбор, а на выходе – список подходящих игроков. Тело запроса представлено в листинге 3, а пример результата – на рисунке 23.

```

28 ▾ "links": {
29 ▾   "players": {
30 ▾     "analyzers": [
31 ▾       "identity"
32 ▾     ],
33 ▾     "fields": {
34 ▾       "online": {},
35 ▾       "params": {
36 ▾         "fields": {
37 ▾           "level": {}
38 ▾         }
39 ▾       }
40 ▾     },
41     "includeAllFields": false,
42     "storeValues": "none",
43     "trackListPositions": false
44   }
45 }
46 }

```

Рисунок 22 – Структура представления player_level_view

```

LET level = (
  FOR player IN players
  FILTER player._key == @player_name
  LIMIT 1

  RETURN player.params.level
)

FOR player IN player_level_view
SEARCH player.online == true AND player.params.level IN level-5..level+5
FILTER player._key != @player_name

RETURN {
  name: player._key,
  level: player.params.level
}

```

Листинг 3 – Запрос на подбор игроков

```

1 ▾ [
2 ▾   {
3     "name": "player3000",
4     "level": 53
5   }
6 ]

```

Рисунок 23 – Результат запроса системы подбора

3.6 Выводы

В этой главе была сформирована структура СУБД для наглядности использования мультимодельного подхода при решении комплексных специфических задач игровой отрасли.

Проделанная работа ясно показывает, что возможности мультимодельного подхода и, особенно, ArangoDB, хорошо накладываются на потребности различных рассмотренных механик.

4 ПРОВЕДЕНИЕ ЭКСПЕРИМЕНТОВ

Чтобы понять преимущества мультимодельных СУБД над одномодельными, необходимо также рассмотреть примеры, в которых имеются непохожие друг на друга критерии работы с данными.

4.1 Выбор одномодельных СУБД для проведения анализа

В качестве реляционной СУБД для проведения экспериментов выбрана PostgreSQL, поскольку она является одной из наиболее популярных [34] среди реляционных СУБД. Она обладает высокой мощностью и достаточно широким функционалом для визуализации данных и профилирования, рассчитана на использование с огромными массивами информации, а также имеет библиотеки для подключения в различные языки программирования, в особенности Python, что поможет при заполнении БД синтетическими данными.

Для тестирования документной модели была выбрана MongoDB, поскольку она также имеет удобную библиотеку для языка Python, а также является кроссплатформенной БД.

4.2 Сравнение производительности системы квестов

Первый эксперимент будет проведён на квестовой системе. Стоит выделить тот факт, что будет рассматриваться не предварительная подготовка базы данных, например, добавление новых квестов без привязки к игроку или добавление нового игрока, поскольку эти действия производятся в период формирования или миграции БД.

Основные запросы, которые необходимы и будут запрашиваться игроками или системой достаточно часто – это добавление игроку нового активного квеста, завершение квеста, получение всех активных и завершённых квестов игрока для отображения на его стороне, а также проверка, выполнен ли квест у игрока.

Выбранные для тестирования структуры одномодельных СУБД представлены на рисунке 24 и 25. Для MongoDB и ArangoDB были созданы коллекции `players` и `quests`. В случае с документной моделью был выбран подход хранить активные и завершенные квесты внутри документа игрока (рисунок 26). Такой же подход можно было бы использовать и в мультимодельной БД, однако было решено попробовать испытать связующую коллекцию `players_quests`.

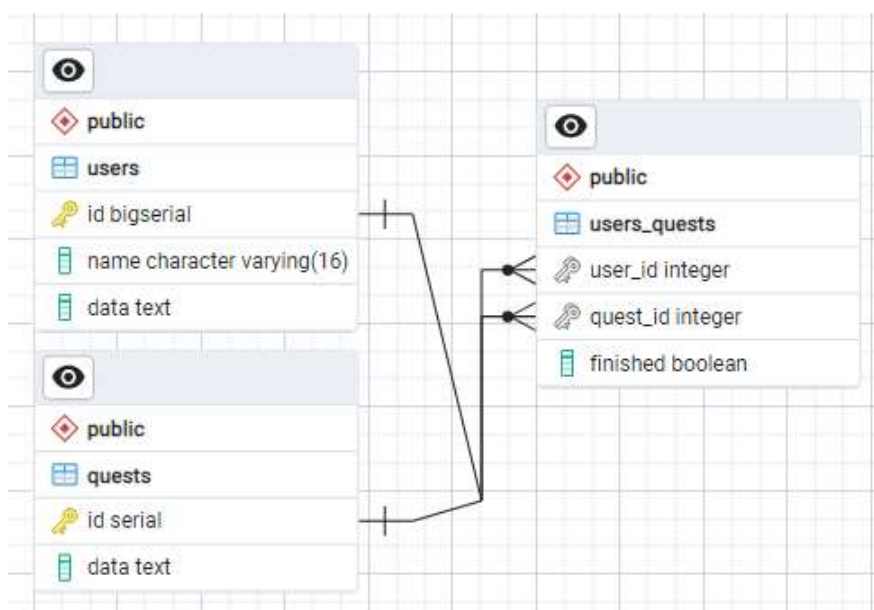


Рисунок 24 – Система квестов реляционной БД

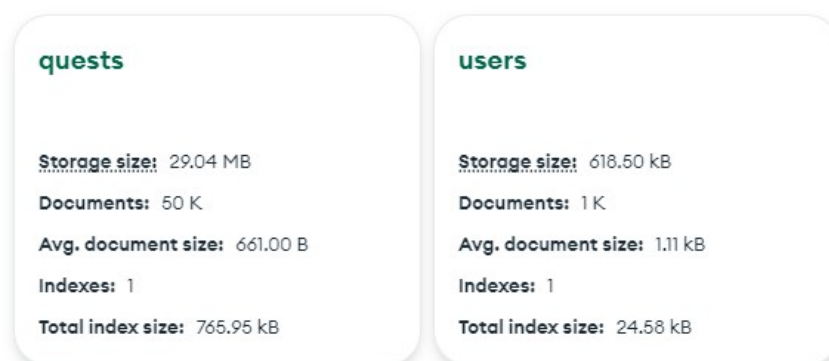


Рисунок 25 – Система квестов документной БД

Эксперимент проводился на выборке из 20000 игроков, 50000 квестов, у каждого игрока в среднем 3000 завершенных квестов, 50 активных. Результаты приведены в таблице 3.

```

    _id: ObjectId('662ad91aed9bb29fcffaacc6')
    name : "ONXVISJYHKPROZTT"
    ▶ active_quests : Array (70)
    ▶ finished_quests : Array (3050)

```

Рисунок 26 – Примера документа players

Таблица 3 – Сравнение средней скорости запросов для разных моделей

	Реляционная	Документная	Мультимодельная
Добавление активного квеста игроку	0,56ms	2,76ms	1,4ms
Перевод квеста в статус «Завершён»	0,66ms (с индексом по user_id и quest_id)	4,084ms	4ms
Получить все активные квесты игрока	1,09ms	1,874ms	4,69ms
Получить все завершённые квесты игрока	1,917ms	1,76ms	8,3ms
Завершён ли квест у игрока	0,5ms	1,573ms	1,45ms

Эксперимент показал, что реляционная база данных справляется с задачей извлечения квестов для конкретного игрока приблизительно с той же скоростью, что и документная база данных, однако добавление и изменение информации происходит значительно быстрее. Это также

является следствием того, что нет необходимости извлекать квесты частями, поскольку для их отображения на клиенте нужна вся информация целиком.

Несмотря на возможность ArangoDB работать с графовыми представлениями, при проведении тестов возникла проблема с запрашиваем квестов у конкретного игрока, причём чем больше было квестов, тем дольше длился запрос. По сравнению с реляционной и документной базами данных разница существенная. Были в том числе предприняты попытки добавить индексы, однако в виду того, что квесты, хранимые в связях, по статистике почти всегда будут в несколько раз преобладать над активными квестами, индексы не исправят проблему.

Правильным способом запрашивать квесты с точки зрения бизнес-логики будет во время захода пользователя в игру. Пока квест активен, нам нужна вся его информация, часть – для отображения, часть – для серверной логики, а значит не требуется запрашивать поля отдельно, реляционная СУБД в этом случае будет иметь преимущество. Мульти модельная же СУБД при той же структуре как в документной базе данных показывает преимущество в скорости при добавлении и изменении данных, что значит, что при необходимости переноса или изначального создания структуры в мульти модельной СУБД сильного ухудшения производительности не ожидается (таблица 4).

Таблица 4 – Средняя скорость запроса для мульти модельной СУБД со структурой из документной СУБД

	Мульти модельная
1	2
Добавление активного квеста игроку	1,4ms

Продолжение таблицы 4

1	2
Перевод квеста в статус «Завершён»	1,8ms
Получить все активные квесты игрока	1,4ms
Получить все завершённые квесты игрока	1,4ms
Завершён ли квест у игрока	1,3ms

4.3 Сравнение производительности системы кланов

Рассмотрим следующий компонент игровых проектов – возможность создавать кланы. Кланом владеет какой-либо игрок, который может приглашать в него других, а также выдавать им специальные роли. Помимо этого, кланы могут враждовать друг с другом. Игроки в таком случае могут атаковать других игроков из враждебного клана, ввиду чего необходимо отмечать определённым цветом игроков в зоне видимости, если он из враждебного клана.

Для хранения таких данных в реляционной СУБД потребуется четыре таблицы: `players`, `clans`, `players_clans`, `clans_relations` (рисунок 27). Ответ на вопрос о враждебности конкретного игрока в таком случае можно будет получить с помощью запроса из приложения Е.

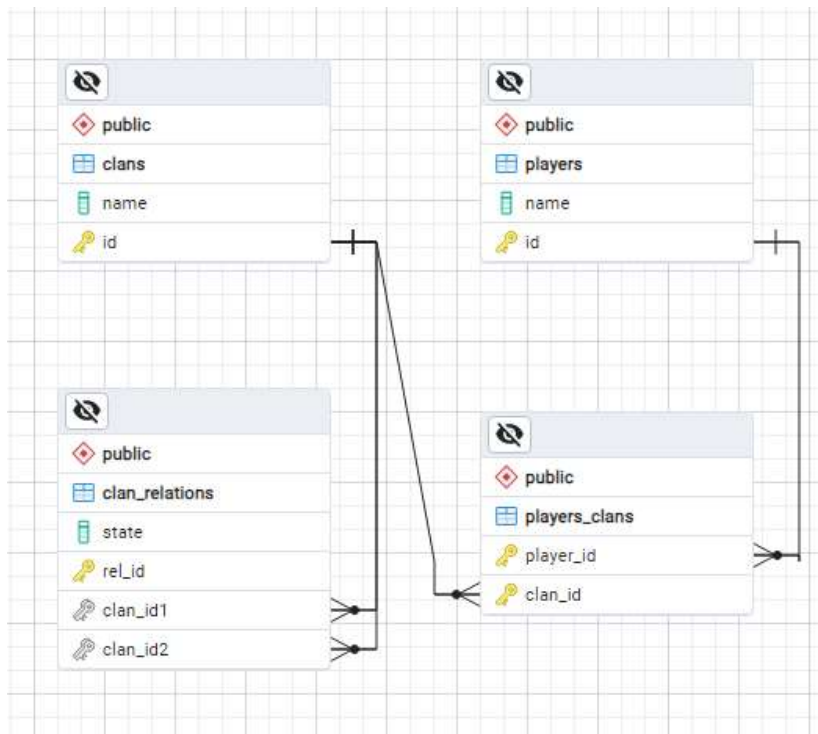


Рисунок 27 – Схема PostgreSQL для системы кланов

Для того чтобы хранить такую информацию в мультимодельной СУБД необходимо иметь коллекции `players` и `clans`, а также коллекцию связей `players_clans`, в которой будут храниться принадлежность игроков к кланам, а также отношения между самими кланами. При этом последние будут также хранить параметр связи `state`, который указывает, находятся ли кланы в состоянии мира или войны (рисунок 28). Теперь чтобы найти, враждуют ли два игрока, нужно пройти по графу на 3 вершины вперёд (игрок принадлежит клану, клан относится к клану, клану принадлежит игрок) (листинг 3).

```
for player in players
filter player._id == "players/@me"
FOR v,e,p in 3..3 ANY
player players_clans
filter v._id == "players/@enemy"
filter p.edges[*].state ANY == "war"
return p
```

Листинг 3 – Запрос в ArangoDB для поиска враждебных игроков

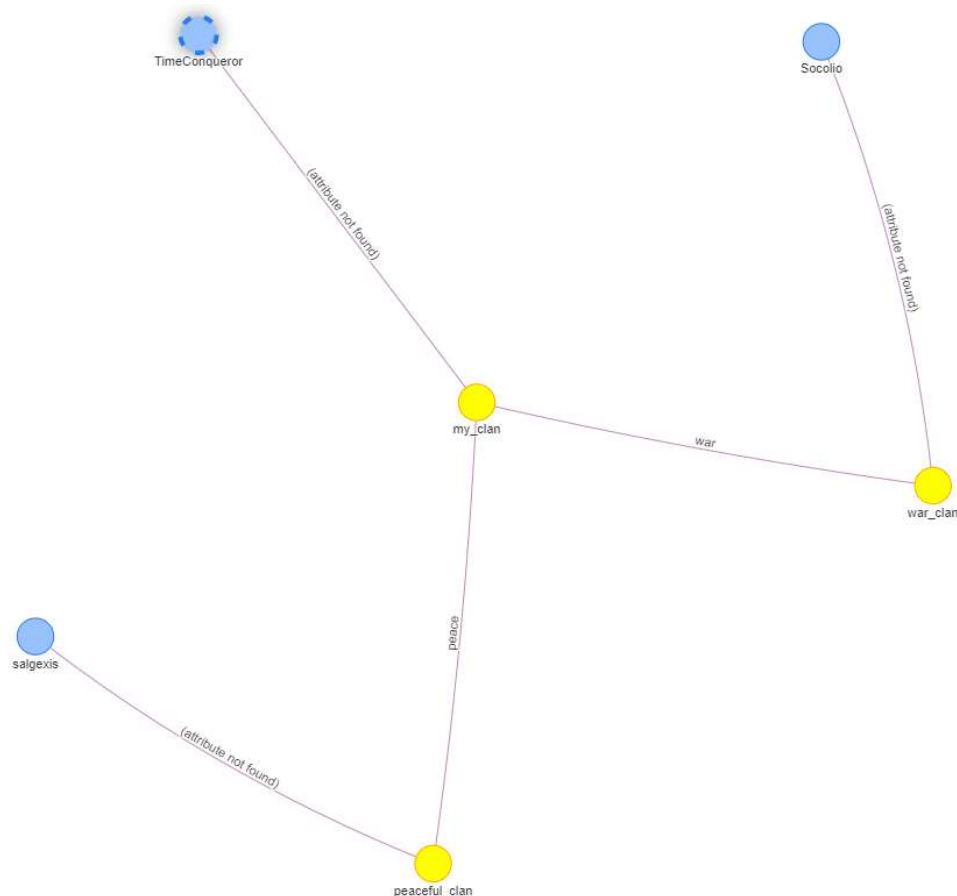


Рисунок 28 – Структура графа для поиска враждебных игроков

Эксперимент по сравнению средней скорости ответа проведён на выборке из 230000 игроков и 7000 кланов и представлен в таблице 5.

Таблица 5 – Средняя скорость запроса реляционной и мультимодельной СУБД для поиска враждебных игроков

	Реляционная	Мультимодельная
Поиск враждебных игроков	57.6ms	5ms

Как мы можем заметить, реляционная СУБД многократно уступает мультимодельной, когда необходимо сравнить данные одного типа по каким-либо связям, то есть совмещать несколько больших таблиц. Графовая модель, чьими возможностями обладает мультимодельная СУБД, позволяет решать такие задачи и делать это очень эффективно.

4.4 Сравнение производительности системы диалогов

Третьей рассмотрим систему диалогов, которая встречается в преобладающем большинстве игр. Диалоги – это последовательность инструкций. Каждая инструкция обладает уникальным идентификатором, набором ссылок на идентификаторы следующих инструкций вместе с предикатами, указывающими, может ли следующая инструкция быть запущена, учитывая некоторые параметры. Инструкция может принадлежать только одному диалогу.

Различают несколько типов инструкций. Первый тип – это реплика NPC, которая включает в себя сам текст либо ключ на локализованный вариант текста, а также идентификатор, имя или ссылку на NPC, который произносит этот текст. Вторым типом инструкции является список ответов игрока после определённой реплики NPC. Третий типом является катсцена, которая содержит ссылку на расположение файла на стороне пользователя.

Если рассматривать структуру диалогов в рамках реляционной СУБД, то необходимо создать две таблицы: `instructions`, которая содержит в себе сами инструкции и `instruction_links`, позволяющая связь сущности из `instructions` (рисунок 29).

Существует два варианта того, как запрашивать данные диалога из базы данных. Первый из них – отправить в запросе идентификатор стартовой инструкции и от неё найти все следующие инструкции, принадлежащие одной последовательности до тех пор, пока ссылка на следующую инструкцию не будет равна `null`. Для реляционной базы данных задача обхода всех связанных ветвящихся инструкций – крайне сложная и трудоёмкая задача. Поэтому если используется реляционная БД, то все данные в рамках одного диалога проще хранить в упакованном виде, то есть в одном столбце. Однако такой способ предполагает накладки по производительности, так как частая выгрузка больших данных будет

довольно медленной. В таких случаях данные можно закешировать и хранить в памяти вызывающего приложения, однако это также даст дополнительные накладные расходы в случае большого количества игроков и приведёт к большому количеству занимаемой памяти.

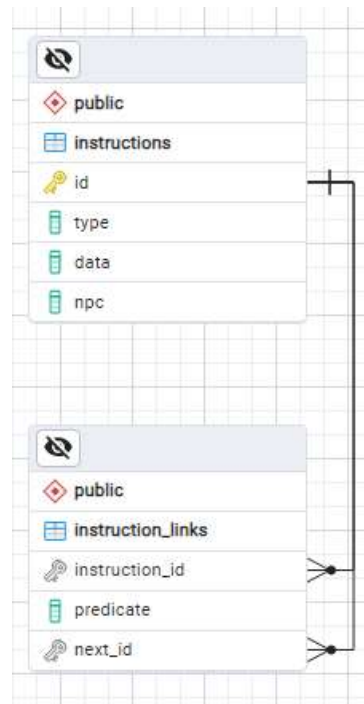


Рисунок 29 – Структура реляционной СУБД для диалоговой системы

Если диалоговую систему рассматривать в мультимодельной СУБД, а именно в ArangoDB, то в таком случае имеет смысл хранить её в виде документов, где каждый документ содержит в себе все инструкции (рисунок 30). При этом решается вопрос с тем, что реплики NPC ограничиваются единственной строкой текста, в то время как ответов игрока может быть несколько – компонент `data` мы можем хранить как в виде строки, так и в виде массива строк. Компонент `next`, отвечающий за выбор следующих инструкций, также является массивом, где каждый элемент может хранить в себе условие для перехода – `predicate`. Если поле условия отсутствует, то следующий идентификатор инструкции считается как путь по умолчанию.


```

1  {
2    "instructions": [
3      {
4        "id": 0,
5        "type": "npc_text",
6        "data": "Привет, путник!",
7        "next": [
8          {
9            "id": 1
10         }
11       ]
12     },
13     {
14       "id": 1,
15       "type": "player_answer",
16       "data": ["И тебе привет!", "Промолчать"],
17       "next": [
18         {
19           "id": 2
20         },
21         {
22           "predicate": ["silence_potion", "choosen_2"],
23           "id": null
24         }
25       ]
26     },
27     {
28       "id": 2,
29       "type": "action",
30       "data": "npc_go_away",
31       "next": [
32         {
33           "id": null
34         }
35       ]
36     }
37   ]
38 }

```

Рисунок 30 – Пример хранения в документном стиле для диалоговой системы

Такой формат хранения данных вкупе с мультимодельной СУБД позволяет удобно и в тоже время эффективно доставать нужную информацию. Это также убирает необходимость дополнительного кеширования.

4.5 Анализ результатов

Экспериментальная часть наглядно показала, что в случае, когда данные необходимо часто добавлять и изменять, а структура этих данных, если рассматривать их в виде таблиц близка к линейной и не требует сложных слияний, мультимодельная СУБД проигрывает реляционной по производительности, в особенности это касается запроса извне больших массивов данных. Несмотря на это, для таких сущностей как квесты, запросы на большой объём данных крайне редки, и поэтому, хоть мультимодельные СУБД в этом проигрывают, это не является препятствием к их использованию в случае, если при этом они используются для решения иных задач.

Игровым проектам также для некоторых случаев необходимо использовать графовую модель. Реляционная СУБД частично может покрыть эту необходимость за счёт резкого снижения производительности вкупе с очень комплексными запросами, но мультимодельная СУБД показывает лучшие результаты и позволяет покрыть большее количество потребностей.

При этом мультимодельные СУБД показывают хорошие результаты и при обращении к данным, хранящимся в документной модели. Часть игровой информации очень удобно хранить в этом формате, поэтому это несомненно добавляет мультимодельным БД преимущество.

ЗАКЛЮЧЕНИЕ

В рамках работы было проведено исследование применимости мультимодельных СУБД в рамках игровой индустрии и разработана структура мультимодельного хранилища для различных игровых компонентов.

Были проанализированы существующие мультимодельные подходы, их преимущества и недостатки. Помимо этого, было также проведено исследование области игровой индустрии, а именно выделены наиболее часто используемые механики, системы и компоненты.

На основе этих исследований было спроектировано мультимодельное хранилище на базе ArangoDB, основываясь на потребностях RPG-игр. Оно показало, что разные компоненты системы ввиду специфики области зачастую требуют как запросов к графовой модели, так и при необходимости хранения данных в виде документов.

Кроме того, был проведён анализ производительности и удобства использования мультимодельными СУБД по сравнению с одномодельными. Он показал, что мультимодельный подход имеет значительные преимущества в игровой области на большем количестве компонентов. Ими являются: а) хранение и поиск внутриигровых социальных взаимодействий, б) эффективный доступ к связанным данным с разной моделью хранения, в) использование единого языка запросов под все поддерживаемые модели (например, одновременный поиск по графовым и документным объектам). Помимо этого, не исключена необходимость использования модели «ключ-значение» для хранения таких данных, как игровые логи, что также реализуемо в рамках мультимодельной СУБД. Отдельно стоит отметить удобство и универсальность языка запросов AQL, который является неотъемлемым

компонентом ArangoDB, а также специализированный поисковый движок Arangosearch, упрощающий написание запросов.

Были также найдены случаи, когда мультимодельная БД проигрывает в производительности другим моделям, например, при выстраивании связей между элементами коллекции разных типов через специальную коллекцию ребёр, что оказалось менее эффективно, чем использовать изначально документный подход. Несмотря на меньшее удобство, такой способ всё также применим и в мультимодельной СУБД.

Из недостатков использования ArangoDB стоит отметить сложность просмотра графовых данных, поскольку встроенный визуализатор имеет довольно скудное количество возможностей: нельзя выделять и передвигать группы компонентов и связей, а также быстро повторно генерировать получившийся граф. Сами связи в графе также визуализируются неоптимальным образом, оставляя большое количество неиспользуемого пространства.

Помимо этого, мультимодельный подход может не быть наилучшим решением в тех случаях, когда игровые проекты минималистичны и узконаправлены, то есть обладают малой вариативностью данных, как например, в игре Contexto.

Существует также несколько вариантов направления дальнейших исследований:

1. Сравнение нескольких мультимодельных СУБД друг с другом с учётом особенностей предметной области
2. Анализ применимости мультимодельных СУБД с учётом особенностей конкретного жанра

Таким образом, мультимодельные СУБД могут с успехом применяться в игровых проектах.

Все запросы, рисунки, а также дополнительная информация опубликована в репозитории на GitHub:
<https://github.com/TimeConqueror/gamedev-multimodal-dbms>

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Листьев Д. С., Савина А. Г., Малявкина Л. И. Тенденции развития игровой индустрии //Цифровые инструменты обеспечения устойчивого развития экономики и образования: новые подходы и актуальные проблемы. – 2022. – С. 69-75.
2. Кишкевич Д. В., Нестеренков С. Н., Марков А. Н. Применение технологии Big Data в игровой индустрии. – 2022.
3. Khine P. P., Wang Z. A review of polyglot persistence in the big data world //Information. – 2019. – Т. 10. – №. 4. – С. 141.
4. Лапуценко А. В., Ключкова М. В. Особенности различных систем управления базами данных //Потенциал российской экономики и инновационные пути его реализации. – 2022. – С. 337-342.
5. Jowan, S. A., Faraj Swese, R., Yousf Aldabrzi, A., & Saad Shertil, M. (2016). Traditional RDBMS to NoSQL Database: New Era of databases for Big Data. Journal of Humanities and Applied Science, 29.
6. Алексеев А. М., Борозна В. С., Суружий Н. А. Системы управления базами данных. Классификация систем управления базами данных. – 2023.
7. Карпюк, А. Д. Какие виды СУБД и их реализации существует и что подходит лучшим образом для разработки автоматизированных информационных систем? / А. Д. Карпюк, Д. Г. Власенкова // Приоритетные направления инновационной деятельности в промышленности: Сборник научных статей по итогам четвертой международной научной конференции., Казань, 29–30 апреля 2020 года. Том Часть 2. – 2020. – С. 62-64. – EDN UPKNFL.
8. Stack Overflow Developer Survey 2023. [Электронный ресурс] URL: <https://survey.stackoverflow.co/2023> (Дата обращения 21.01.2024)

9. Соколов К. К. Сравнение реляционных и нереляционных моделей СУБД //Научное обеспечение технического и технологического прогресса. – 2019. – С. 39-41
- 10.Guo, Qingsong & Zhang, Chao & Zhang, Shuxun & Lu, Jiaheng. (2023). Multi-model query languages: taming the variety of big data. Distributed and Parallel Databases. 1-41.
- 11.Lu, Jiaheng & Holubova, Irena. (2019). Multi-model Databases: A New Journey to Handle the Variety of Data. ACM Computing Surveys. 52. 1-38.
- 12.Rai, P. K., & Singh, P. (2015). International Journal of Computer Science and Mobile Computing Studies and Analysis of Popular Database Models. International Journal of Computer Science and Mobile Computing, 4(5), 834–838.
- 13.Куприянич Е. М. Сравнительный анализ подходов к разработке приложений NoSQL и SQL СУБД //XI Конгресс молодых ученых. – 2022. – С. 72-75.
- 14.Pokorný J. Graph databases: their power and limitations //Computer Information Systems and Industrial Management: 14th IFIP TC 8 International Conference, CISIM 2015, Warsaw, Poland, September 24-26, 2015, Proceedings 14. – Springer International Publishing, 2015. – С. 58-69.
- 15.Шарипова, Н. Н. Об использовании NOSQL-хранилищ данных / Н. Н. Шарипова // Восточно-Европейский научный журнал. – 2016. – Т. 9, № 3. – С. 73-76. – EDN XRXLFF.
- 16.Галигузова Е. В., Илларионова Ю. Е. Сравнение реляционных и нереляционных СУБД //Символ науки. – 2023. – №. 1-2. – С. 14-17.
17. Microsoft Learn [Электронный ресурс]. Azure Cosmos DB Documentation URL: <https://learn.microsoft.com/en-us/azure/cosmos-db/> (Дата обращения 04.03.2024)

18. [Электронный ресурс]. NoSQL Performance Benchmark 2018 – MongoDB, PostgreSQL, OrientDB, Neo4j and ArangoDB URL: <https://arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/> (Дата обращения 05.03.2024)
19. Ye, Feng & Sheng, Xinjun & Nedjah, Nadia & Sun, Jun & Zhang, Peng. (2023). A Benchmark for Performance Evaluation of a Multi-Model Database vs. Polyglot Persistence. Journal of Database Management. 34. 1-20.
20. Unity 2022 Multiplayer Report [Электронный ресурс] URL: <https://create.unity.com/multiplayer-report-2022> (Дата обращения 10.01.2024)
21. Newzoo Global Games Market Report [Электронный ресурс] URL: <https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2023-free-version> (Дата обращения 10.04.2024). – Режим доступа: по запросу.
22. Fallout 76 Wiki: [Электронный ресурс]. Fallout 76 Crafting URL: https://fallout.fandom.com/wiki/Fallout_76_crafting (Дата обращения 29.03.2024).
23. Официальный сайт компании Activision Blizzard: [Электронный ресурс]. Дневники разработчиков URL: <https://news.blizzard.com/ru-ru/world-of-warcraft/21881587/za-chashkoj-kofe-s-razrabotchikami-klassicheskaya-versiya-world-of-warcraft> (Дата обращения: 10.03.2024).
24. The Binding of Isaac Wiki: [Электронный ресурс]. Предметы URL: <https://bindingofisaacrebirth.fandom.com/wiki/Items> (Дата обращения: 10.03.2024).
25. Minecraft Wiki: [Электронный ресурс]. Data pack URL: https://minecraft.fandom.com/wiki/Data_pack (Дата обращения: 19.03.2024).

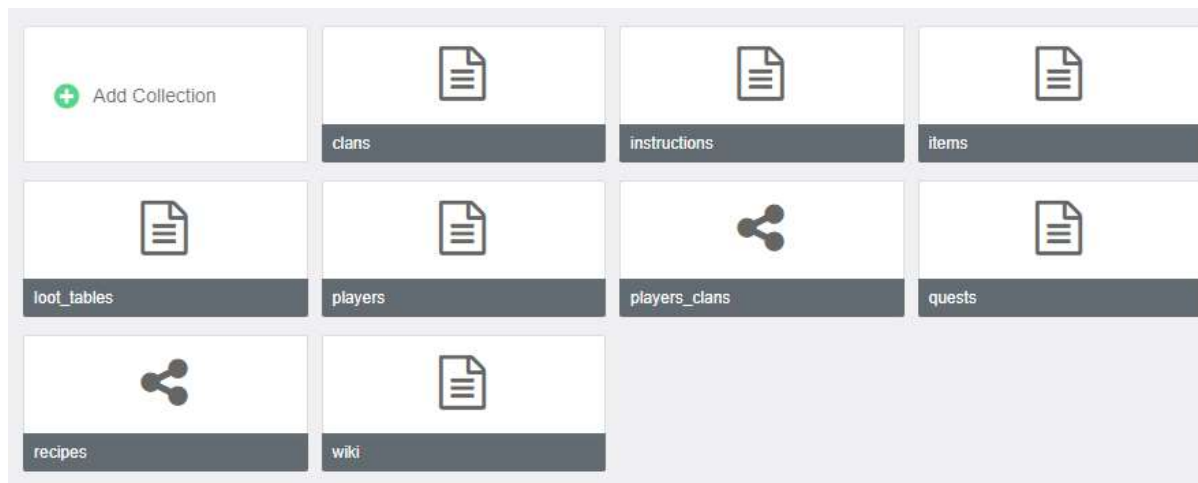
26. Joshua Butcher. (2012). Databases in Online (Social) Gaming Designing a scalable data infrastructure. (n.d.).
27. MMORPG Data Storage (Part 1): [Электронный ресурс]. Plant Based Games. URL: <https://plantbasedgames.io/blog/posts/01-mmorpg-data-storage-part-one/> (Дата обращения: 25.01.2024)
28. Abdullah Alqwbani, B., Zuping, Z., Aqlan, F., Alqwbani α, A., Zuping σ, Z., & Aqlan ρ, F. (2014). Big Data Management for MMO Games and Integrated Website Implementation. Type: Double Blind Peer Reviewed International Research Journal Publisher: Global Journals Inc, 14.
29. Официальный сайт игры Eve Online: [Электронный ресурс]. News. Making Our Backside Bigger. URL: <https://www.eveonline.com/news/view/making-our-backside-bigger> (Дата обращения: 02.02.2024).
30. Котенко В. Н., Елисеев В. О. Инновационный метод хранения данных в играх в жанре Role-Playing Game //Донецкие чтения 2021: образование, наука, инновации, культура и вызовы современности. – 2021. – С. 243-246.
31. Официальный сайт компании Activision Blizzard: [Электронный ресурс]. Дневники разработчиков URL: <https://news.blizzard.com/ru-ru/world-of-warcraft/21881587/za-chashkoj-kofe-s-razrabotchikami-klassicheskaya-versiya-world-of-warcraft> (Дата обращения: 29.12.2023).
32. Официальный сайт игры Fortnite: [Электронный ресурс]. News. Postmortem of service outage at 3.4M CCU. URL: <https://www.fortnite.com/news/postmortem-of-service-outage-at-3-4m-ccu> (Дата обращения: 02.02.2024).
33. Блог компании VK: [Электронный ресурс]. Базы данных в онлайн-играх. От Аллодов Онлайн до Skyforge. URL: <https://habr.com/ru/companies/vk/articles/182088/> (Дата обращения: 29.12.2023).

34. DB-Engines Ranking [Электронный ресурс] URL: <https://db-engines.com/en/ranking> (Дата обращения 04.01.2024)
35. ArangoDB Documentation [Электронный ресурс] URL: <https://docs.arangodb.com/stable/about-arangodb/> (Дата обращения 26.03.2024)
36. MongoDB Documentation [Электронный ресурс] URL: <https://www.mongodb.com/docs/> (Дата обращения 26.03.2024)
37. MongoDB Resources [Электронный ресурс] How to use Python with MongoDB URL: <https://www.mongodb.com/resources/languages/python> (Дата обращения 28.03.2024)
38. PostgreSQL: Documentation [Электронный ресурс] URL: <https://www.postgresql.org/docs/> (Дата обращения 26.03.2024)

ПРИЛОЖЕНИЕ А

Объекты разработанной базы данных

Общий вид объектов:



Документы коллекции clans:

Content
<pre>{"_id": "clans/RVJEJL", "_key": "RVJEJL", "_rev": "_h2Q0CvS---"}</pre>
<pre>{"_id": "clans/UXFJCB", "_key": "UXFJCB", "_rev": "_h2Q0C2m---"}</pre>
<pre>{"_id": "clans/EVSKTP", "_key": "EVSKTP", "_rev": "_h2Q0C96---"}</pre>
<pre>{"_id": "clans/SFNOUG", "_key": "SFNOUG", "_rev": "_h2Q0DN6---"}</pre>
<pre>{"_id": "clans/FDYKRI", "_key": "FDYKRI", "_rev": "_h2Q0D1e---"}</pre>
<pre>{"_id": "clans/PBGJVP", "_key": "PBGJVP", "_rev": "_h2Q0D2e---"}</pre>
<pre>{"_id": "clans/PKATKB", "_key": "PKATKB", "_rev": "_h2Q0D9K---"}</pre>
<pre>{"_id": "clans/KCBLKQ", "_key": "KCBLKQ", "_rev": "_h2Q0ELq---"}</pre>
<pre>{"_id": "clans/ACOUIO", "_key": "ACOUIO", "_rev": "_h2Q0EaS---"}</pre>
<pre>{"_id": "clans/OCMHYD", "_key": "OCMHYD", "_rev": "_h2Q0EhW---"}</pre>

Документы коллекции instructions:

Content	_key
<pre>{ "_id": "instructions/56854", "_key": "56854", "_rev": "h19dCoe---", "instructions": [{ "id": 0, "type": "npc_text", "data": "Привет, путник!", "next": [{ "id": 1 }] }, { "id": 1, "type": "player_answ...</pre>	56854
<pre>{ "_id": "instructions/414074", "_key": "414074", "_rev": "h2QR3AQ--", "instructions": [{ "id": 0, "type": "npc_text", "data": "Это кто пожалован?", "next": [{ "id": 1 }] }, { "id": 1, "type": "player_...</pre>	414074
<pre>{ "_id": "instructions/459936", "_key": "459936", "_rev": "h2Q59L----", "instructions": [{ "id": 0, "type": "player_answer", "data": ["Ну как, готово?", "Уйти"], "next": [{ "id": 1 }] }, { "id": 1, "ty...</pre>	459936

Документы коллекции items:

Content	_key
<pre>{"_id":"items/stone", "_key":"stone", "_rev":"_h1f0zeC---", "name":"Stone"}</pre>	stone
<pre>{"_id":"items/flint", "_key":"flint", "_rev":"_h1f1Q92---", "name":"Flint"}</pre>	flint
<pre>{"_id":"items/wood", "_key":"wood", "_rev":"_h1f1cbW---", "name":"Wood"}</pre>	wood
<pre>{"_id":"items/planks", "_key":"planks", "_rev":"_h1f1mkS---", "name":"Planks"}</pre>	planks
<pre>{"_id":"items/workbench", "_key":"workbench", "_rev":"_h1f2Rj6---", "name":"Workbench"}</pre>	workbench
<pre>{"_id":"items/slab", "_key":"slab", "_rev":"_h1hXvpm---", "name":"Slab"}</pre>	slab
<pre>{"_id":"items/iron_sword", "_key":"iron_sword", "_rev":"_h1hm-d---", "desc":"The Sword to make enemies cry!", "meta":{"durability":260}, "name":"Iron _</pre>	iron_sword
<pre>{"_id":"items/superbench", "_key":"superbench", "_rev":"_h1itZjG---", "name":"Superbench"}</pre>	superbench

Документы коллекции loot tables:

```
Content

{"_id":"loot_tables/firewood_ent","key":"firewood_ent","rev":"h1lbWm---","pools":[{"rolls":2,"entries":[{"item":"items/planks","count":3,"weight":1}]}]}

{"_id":"loot_tables/frosty","key":"frosty","rev":"h1mZl9---","pools":[{"rolls":2,"entries":[{"item":"items/ice","count":1,"weight":5}, {"item":"items/snow_globe","count":1,"weight":5}]}]}
```

Документы коллекции players:

Content
<code>{"_id":"players/Time_Conqueror","_key":"Time_Conqueror","_rev":"_h2Pxxz---","inventory":[{"item":"items/stone","count":5,"data":{}},{ "item":"item...</code>
<code>{"_id":"players/player3000","_key":"player3000","_rev":"_h2Px8y2---","inventory":[],"online":true,"params":{"pos":[5000,10,100],"level":53,"experi...</code>
<code>{"_id":"players/master","_key":"master","_rev":"_h2PyMai---","inventory":[],"online":true,"params":{"pos":[5000,10,2100],"level":83,"experience":6...</code>
<code>{"_id":"players/Socolio","_key":"Socolio","_rev":"_h2PyRE0---","inventory":[],"online":true,"params":{"pos":[200,300,100],"level":21,"experience":...</code>
<code>{"_id":"players/algorithmX","_key":"algorithmX","_rev":"_h2Pye4---","inventory":[],"online":false,"params":{"pos":[5000,10,100],"level":51,"exper...</code>
<code>{"_id":"players/EPSJKAZRWU","_key":"EPSJKAZRWU","_rev":"_h2QOCve---"}</code>
<code>{"_id":"players/TXDPTGYVIU","_key":"TXDPTGYVIU","_rev":"_h2QOCvq---"}</code>
<code>{"_id":"players/CJZSHCYAZR","_key":"CJZSHCYAZR","_rev":"_h2QOCvy---"}</code>

Документы коллекции players_clans:

Content

```
{ "_from": "players/TUFCTPNSRJ", "_id": "players_clans/1719552", "_key": "1719552", "_rev": "_h2QyZcW---", "_to": "clans/QSNQLZ" }

{ "_from": "players/URAULZWQSI", "_id": "players_clans/1719555", "_key": "1719555", "_rev": "_h2QyZci---", "_to": "clans/QSNQLZ" }

{ "_from": "players/ONIAPSJBHF", "_id": "players_clans/1719558", "_key": "1719558", "_rev": "_h2QyZcq---", "_to": "clans/QSNQLZ" }

{ "_from": "players/HTLXYMMUYA", "_id": "players_clans/1719562", "_key": "1719562", "_rev": "_h2QyZc6---", "_to": "clans/BBUNXQ" }

{ "_from": "players/QZUPBHTTJQ", "_id": "players_clans/1719565", "_key": "1719565", "_rev": "_h2QyZdC---", "_to": "clans/BBUNXQ" }

{ "_from": "players/FEPTZDNOZI", "_id": "players_clans/1719568", "_key": "1719568", "_rev": "_h2QyZdK---", "_to": "clans/BBUNXQ" }

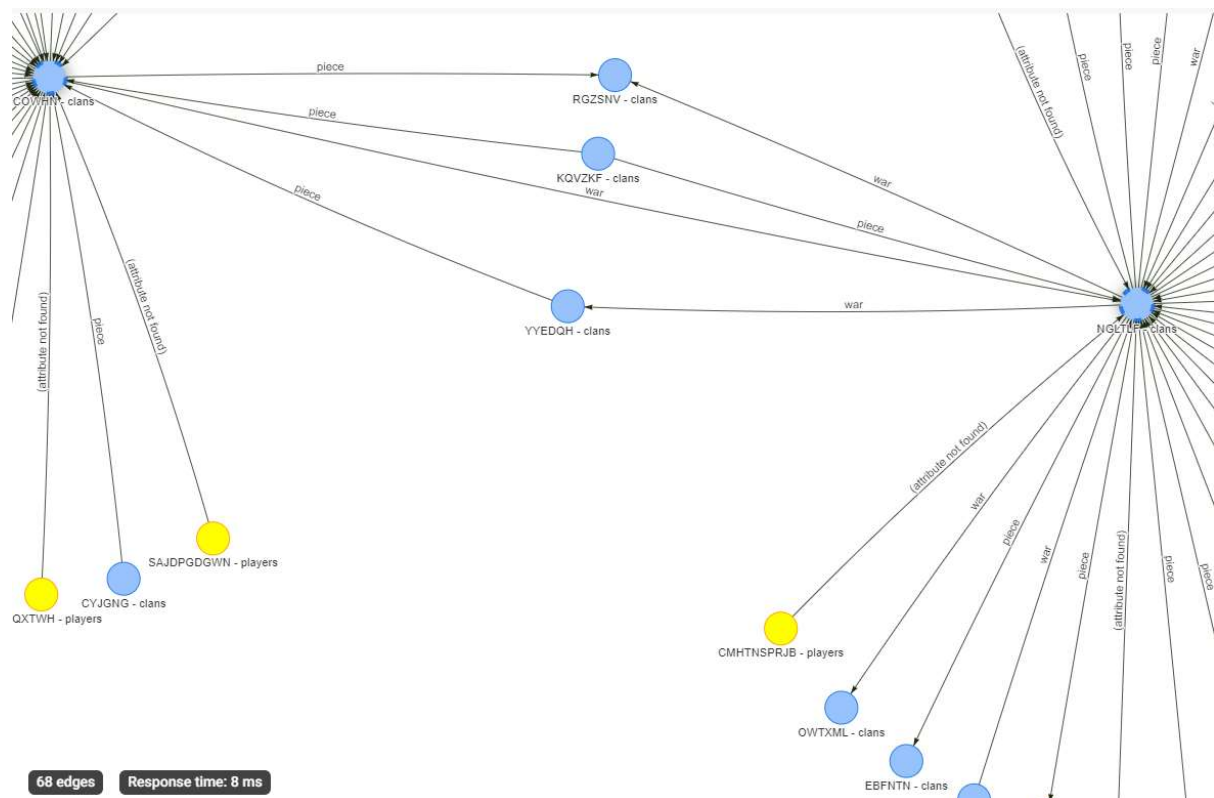
{ "_from": "players/KVBVPGKQQ", "_id": "players_clans/1719571", "_key": "1719571", "_rev": "_h2QyZd5---", "_to": "clans/BBUNXQ" }

{ "_from": "players/CLPZQHKTkt", "_id": "players_clans/1719574", "_key": "1719574", "_rev": "_h2QyZda---", "_to": "clans/BBUNXQ" }

{ "_from": "players/HBLGLNYUHH", "_id": "players_clans/1719577", "_key": "1719577", "_rev": "_h2QyZdm---", "_to": "clans/BBUNXQ" }

{ "_from": "players/NESCLPEAIW", "_id": "players_clans/1719580", "_key": "1719580", "_rev": "_h2QyZdu---", "_to": "clans/BBUNXQ" }
```

Графовое представление `players_clans` (жёлтым – игроки, синим – кланы):



Документы коллекции quests:

Content
<pre>{ "_id": "quests/855325", "_key": "855325", "_rev": "_h2QW9my---", "desc": "PHADREJDKJVQFAHMKPRQOFFMERDUGZWUTDUGUFFCKQDKTVYOHNOEHVEDGPPUWUYRNKHXRSYAIJZ...</pre>
<pre>{ "_id": "quests/855327", "_key": "855327", "_rev": "_h2QW9m6---", "desc": "DETEHYINJDUSHENEVNHTIIHDRXFEUMSPHCYOSUCHSZNFGYQEQPGTKTZHPFCFNIALBRRHAIZSUUXO...</pre>
<pre>{ "_id": "quests/855329", "_key": "855329", "_rev": "_h2QW9n---", "desc": "UOXJZMDXZLIQSFULOSSKIXOMXLASPODDRHDXSYIHZTNEATNKPVZHEFKHSRTRPLXPEKXKCGMNYA...</pre>
<pre>{ "_id": "quests/855331", "_key": "855331", "_rev": "_h2QW9nC---", "desc": "OTYVUGVIYTPGLWRCODKJGCZEATEDICYSYVGEKJWPMXTFINAAYNKHYZFYKCNMFGXMLOEHPAAXDYITHL...</pre>
<pre>{ "_id": "quests/855333", "_key": "855333", "_rev": "_h2QW9n6---", "desc": "EVNNJVVYXZDCQXCLIWRLCAXLAXIJUSFRROEFHQXDZISBJITYCOCBBDHMKXSCVIVMFHOCBEHMKR...</pre>
<pre>{ "_id": "quests/855335", "_key": "855335", "_rev": "_h2QW9n0---", "desc": "CJOKRABBMBSRTIROREWFIFHMERMUGATMPJWAOTYQTZNCQFYVDIPUDQJUNEYZBLMESGSZLQOBXNJR...</pre>
<pre>{ "_id": "quests/855337", "_key": "855337", "_rev": "_h2QW9n5---", "desc": "OKZHTBOKEGMWTFVNXUNWYISZRDYKJPPWCBAHOOFWUTGKAIPUKQNTWYUODDURBTHUDNXLKCHCT...</pre>
<pre>{ "_id": "quests/855339", "_key": "855339", "_rev": "_h2QW9nW---", "desc": "WFCESPAXNZPKDTYWFHTQQCUDYCMJQXEAHMBVTYHJNZMZQLBXTGYBZPFEAHMIATFURKOHUZNECES...</pre>

Документы коллекции recipes:

Content
<pre>{ "_from": "items/flint", "_id": "recipes/2361", "_key": "2361", "_rev": "_h1h5P_W---", "_to": "items/workbench", "count": 3 }</pre>
<pre>{ "_from": "items/planks", "_id": "recipes/5286", "_key": "5286", "_rev": "_h1h5x1G---", "_to": "items/slab", "count": 2 }</pre>
<pre>{ "_from": "items/wood", "_id": "recipes/2307", "_key": "2307", "_rev": "_h1h53jC---", "_to": "items/planks", "count": 1 }</pre>
<pre>{ "_from": "items/planks", "_id": "recipes/2335", "_key": "2335", "_rev": "_h1h58M0---", "_to": "items/workbench", "count": 6 }</pre>
<pre>{ "_from": "items/workbench", "_id": "recipes/8984", "_key": "8984", "_rev": "_h1iuyEC---", "_to": "items/superbench", "count": 1 }</pre>

Документы коллекции wiki:

Content
<pre>{ "_id": "wiki/draconium_dust", "_key": "draconium_dust", "_rev": "_h1n1FrK---", "description": "This is the dust dropped when mining Draconium Ore. It ca..." }</pre>
<pre>{ "_id": "wiki/basics", "_key": "basics", "_rev": "_h1oRlW0G---", "description": "The very basis of Draconic Research. This ore can be found in the overwor..." }</pre>
<pre>{ "_id": "wiki/draconium_ingot", "_key": "draconium_ingot", "_rev": "_h2QbDm5---", "description": "The product of smelting Draconium Dust in a furnace. Us..." }</pre>

ПРИЛОЖЕНИЕ В

Запрос для поиска рецепта из конкретного списка ингредиентов

```
LET startingNodes = @items

LET nodesResults = (
  FOR snode in startingNodes
  RETURN (
    FOR v, e, p in 1
    OUTBOUND snode.item
    GRAPH recipes
    OPTIONS {uniqueVertices: 'path'}
    RETURN {id: v._id, count: e.count, ingredient: snode.item}
  )
)

LET nodeIdResults = UNIQUE(
  (FOR arr in nodesResults
    RETURN (FOR v in arr RETURN v.id)
  )**)

LET out = (
  FOR result in nodeIdResults
  LET ingredients = (
    FOR v, e, p in 1
    INBOUND result
    GRAPH recipes
    RETURN {item: v._id, count: e.count}
  )

  FILTER (FOR i in ingredients
    RETURN startingNodes[? ANY FILTER CURRENT.item == i.item AND
CURRENT.count >= i.count]
  )[*] ALL == true

  return {result: result, ingredients: ingredients}
)

return out
```

ПРИЛОЖЕНИЕ С

Дополнительные запросы системы рецептов

1. «Получить все рецепты, содержащие указанный ингредиент»

Тело запроса:

```
LET startingNode = @item
```

```
FOR v, e, p in 1
```

```
  OUTBOUND startingNode
```

```
  GRAPH recipes
```

```
  OPTIONS {uniqueVertices: 'path'}
```

```
  RETURN {result: v._id, ingredient_count: e.count}
```

Входные параметры:

Key	Value	JSON
item	items/planks	

Результат запроса:

```
1  [
2  {
3    "result": "items/workbench",
4    "ingredient_count": 6
5  },
6  {
7    "result": "items/slab",
8    "ingredient_count": 2
9  }
10 ]
```

2. «Получить все ингредиенты, требуемые для создания конкретного предмета»

Тело запроса:

```
LET startingNode = @item
```

```
FOR v, e, p in 1
```

```
  INBOUND startingNode
```

```
  GRAPH recipes
```

```
  OPTIONS {uniqueVertices: 'path'}
```

```
  RETURN {result: v._id, ingredient_count: e.count}
```


Входные параметры:

Key	Value	JSON
item	<input type="text" value="items/planks"/>	

Результат запроса:

```
1  [
2    {
3      "result": "items/wood",
4      "ingredient_count": 1
5    }
6  ]
```

ПРИЛОЖЕНИЕ D

Запрос на получение случайного набора ресурсов

```
for table in loot_tables
filter table._key == @loot_table

let out = (for pool in table.pools
  let sumWeight = sum(pool.entries[*].weight)

  let summedEntries = (FOR i IN 0..LENGTH(pool.entries)-1
    LET entry = pool.entries[i]
    LET sum = pool.entries[i].weight + SUM(SLICE(pool.entries, 0, i)[*].weight)
    RETURN {entry:entry,sum:sum})

  let items = (for roll in 0..pool.rolls-1
    let randFloat = (RAND() + DATE_MILLISECOND(DATE_NOW()) / 1000.0) /
2.0
    let randWeight = MIN([sumWeight, CEIL(randFloat * sumWeight)])

    return (for e in summedEntries
      SORT e.sum ASC
      FILTER e.sum >= randWeight
      LIMIT 1
      RETURN {item: e.entry.item, count: e.entry.count})
  )

return items)
return flatten(out, 2)
```

ПРИЛОЖЕНИЕ Е

Запрос для поиска враждующих игроков в реляционной СУБД

```
SELECT
  CASE WHEN EXISTS
    (
      select * from clan_relations
    where state = 'war' and ((
      clan_id1 in (
        select clan_id from players_clans
        left join players on player_id = players.id
        where players.name = @name2
      ) and clan_id2 in (
        select clan_id from players_clans
        left join players on player_id = players.id
        where players.name = @name1
      )
    ) or (
      clan_id1 in (
        select clan_id from players_clans
        left join players on player_id = players.id
        where players.name = @name1
      ) and clan_id2 in (
        select clan_id from players_clans
        left join players on player_id = players.id
        where players.name = @name2
      )
    )
  )
  THEN 'TRUE'
  ELSE 'FALSE'
END;
```