РОССИЙСКАЯ ФЕДЕРАЦИЯ

ХАНТЫ-МАНСИЙСКИЙ АВТОНОМНЫЙ ОКРУГ-ЮГРА

АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ

«СУРГУТСКИЙ ИНСТИТУТ ЭКОНОМИКИ, УПРАВЛЕНИЯ И ПРАВА»

«09.02.07 Информационные системы и программирование»

КУРСОВАЯ РАБОТА

по дисциплине «МДК.11.01 Технология разработки и защиты базы данных»

на тему: «Разработка модели реляционной базы данных для студии разработки видеоигр»

Выполнил: Хазипов М.В.
студент группы ИС23/9-1
Руководитель: Преподаватель профессиональных дисциплин
Колмыков М.В. _____
Оценка _____
Подпись _____
Дата _____

Сургут, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ ДЛЯ ИГРОВЫХ СТУДИЙ 6
1.1 Анализ предметной области 6
1.2 Реляционные базы данных как инструмент управления предприятием 11
1.3 Методологии проектирования реляционных баз данных 15
ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ ДЛЯ ИГРОВОЙ СТУДИИ 22
2.1 Проектирование концептуальной модели данных 22
2.2 Описание логической схемы базы данных 26
2.3 Описание физической схемы базы данных 32
ЗАКЛЮЧЕНИЕ 46
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 48
ПРИЛОЖЕНИЯ

ВВЕДЕНИЕ

Разработка компьютерных игр представляет собой сложный и трудоемкий процесс, который требует усилий множества специалистов и тщательного учета разнообразных данных. Этот процесс охватывает различные взаимосвязанные этапы - от первоначальной концепции и проектирования до программирования, тестирования и выпуска готового продукта. Каждый из этих этапов генерирует значительные объемы информации, нуждающейся в эффективном управлении.

Реляционная база данных предназначается для организации, хранения и эффективного управления всеми важными данными. Такая система обеспечивает целостность данных, их безопасность и доступность для всех участников разработки. В структурированной базе данных может храниться информация о сотрудниках и их квалификации, текущих заданиях и проектах, стадиях разработки каждого продукта, используемых технологиях и ресурсах, финансовых показателях и многих других аспектах деятельности студии.

Централизованное хранение и структурирование этих данных не только способствует эффективному управлению студией, но и упрощает координацию работы над отдельными проектами. Это обеспечивает всех участников процесса доступом к актуальной и достоверной информации. В условиях современной игровой индустрии, где требования к качеству и срокам выпуска продуктов постоянно растут, наличие единого источника информации становится важным фактором.

Для хранения и обработки отдельных видов данных могли бы использоваться электронные таблицы, однако их применение сталкивается с рядом ограничений. Наиболее серьезными проблемами являются разрозненность данных в различных файлах и форматах; сложность поддержания целостности и актуальности информации при частых изменениях; ограниченные возможности масштабирования при росте объемов данных. Эти существенные ограничения делают электронные таблицы

непригодными для построения целостной, надежной и масштабируемой системы управления современной игровой студией.

Основной целью данной курсовой работы является разработка реляционной базы данных, предназначенной для автоматизации ключевых бизнес-процессов, управления человеческими ресурсами и материальными активами, эффективного учета оборудования и технических средств, а также хранения данных о всех этапах жизненного цикла игровых проектов. База данных должна обеспечивать поддержку всех стадий разработки - от первоначального концепта до финального релиза.

Достижение этой цели предполагает последовательное решение нескольких задач. Первой задачей является проведение детального анализа предметной области для выявления функциональных требований к системе. Второй важной задачей выступает проектирование ER-диаграммы, которая визуализирует все ключевые сущности предметной области и отношения между ними. Третья задача включает преобразование ER-диаграммы в логическую схему базы данных с определением атрибутов сущностей. Четвертая задача нормализация логической схемы для минимизации избыточности данных. Завершающей задачей является практическая реализация физической модели базы данных с помощью SQL-скриптов.

Объектом исследования выступает процесс управления данными в игровой студии, включающий накопление, хранение, передачу и использование информации. Предметом исследования является реляционная база данных, предназначенная для автоматизации процессов управления данными внутри студии и координации работы над проектами.

В процессе работы применялся комплекс методов и инструментов, включающий методы анализа предметной области, подходы концептуального моделирования данных, а также инструменты для проектирования и реализации. Такой подход обеспечил создание решения, сочетающего теоретическую обоснованность и практическую применимость.

Разработанная база данных может быть использована в работе игровых студий различного масштаба. Особую ценность она представляет для организаций с большим количеством сотрудников, разнообразным парком оборудования и одновременной разработкой нескольких проектов. Внедрение системы позволяет автоматизировать учет сотрудников, оборудования и ресурсов, параметров проекта и взаимодействия с дочерними студиями.

Реализованное решение способствует эффективному стратегическому и оперативному управлению не только студией в целом, но и отдельными проектами, обеспечивая менеджеров своевременной и точной информацией для принятия обоснованных управленческих решений. База данных позволяет отслеживать прогресс разработки, распределять ресурсы и контролировать сроки выполнения задач, что способствует достижению коммерческого успеха и творческих амбиций студии.

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ ДЛЯ ИГРОВЫХ СТУДИЙ

1.1 Анализ предметной области

Разработка видеоигр включает в себя создание основных механик, сюжета, программирование, создание графики и звука, тестирование и выпуск продукта. Индустрия делится на несколько сегментов, что напрямую влияет на подходы к управлению:

- AAA-разработка - крупнобюджетные проекты с огромными командами, требует жесткого финансового контроля и управления рисками;
- Инди-разработка - небольшие студии или разработчики одиночки, которые часто работают без крупного издателя;
- Мобильная разработка - такие игры ориентированы на массового пользователя, важна аналитика и оперативное обновление контента;
- Разработка для VR/AR - новый, быстро развивающийся сегмент, требующий узконаправленных специалистов.

Разработка игр это процесс, состоящий из нескольких этапов. Начинается все с разработки платформ, на которых будет запускаться игра. Затем очередь за разработчиками программного обеспечения. Когда команда выбрала ОС и движок, начинается разработка игры. Подключаются программисты, дизайнеры, сценаристы, художники, аниматоры. Все они должны сложить свои наработки в один проект под видением главного гейм-дизайнера. Для того чтобы об игре начали говорить, в этот процесс включаются издатели. Издатели берут на себя основные затраты за маркетинг и рекламу, а часто являются и спонсорами всего проекта. В конечном счете об игре узнает аудитория, от итогового мнения которой будет зависеть успех проекта.

Над проектом могут работать разные команды: разработчики, которые собираются вокруг игры мечты; издатели - профессиональные организации с

четкими процессам и бюджетами; смешанный тип представляет собой крупную компанию, сочетающую в себе как команду разработчиков, так и команду издательства. При этом в процессе разработки выделяют ключевые роли, которые и определяют, как будет выглядеть, звучать и распространяться проект. К таким ролям относятся:

- Продюсер, управляет ресурсами, сроками, коммуникациями;
- Гейм-дизайнер - отвечает за игровой опыт, проектирует механики, дизайн и сюжет, часто является главным носителем идеи игры;
- Программист создает техническую основу игры;
- Арт-директор - формирует визуальный стиль игры;
- Маркетолог - занимается продвижением проекта.

Часть специалистов не нужна на всех этапах разработки проекта. Поэтому эффективное управление предполагает планирование их загрузки и привлечения. Разработка проекта начинается с создания продукта, в который входят этапы препродакшена и продакшена. На этапе препродакшена формируется геймплейный цикл, сетап - вводная часть проекта, анализируется то на какую аудиторию будет рассчитан проект. Подготавливается документ, который описывает все аспекты игры: от механик и сюжета до арта и звука. После этого создается упрощенная версии игры для проверки ключевой механики. После препродакшена начинается этап продакшена. Продакшен является самым ресурсоемким этапом. Создание всех игровых ресурсов: 3D-модели, анимации, текстуры, уровни, звуковые эффекты, музыка, код. После разработки начинается сборка всех созданных элементов в единую работающую систему. Когда игра уже имеет целостный вид, начинается тестирование: выявляются программные ошибки, настраивается игровой баланс, оценивается удобность пользовательского опыта. Когда проект уже протестирован, начинается подготовка к выпуску, в ходе которой игра переводится на другие языки, проходит процедуру утверждения на платформах и подготавливаются версии игры для распространения.

Таким образом завершается процесс разработки проекта, и начинается продвижение и монетизация, без них даже самый гениальный проект может остаться незамеченным. Создаются сообщества в социальных сетях, на форумах. Команда постоянно коммуницирует с аудиторией. Для анонса выпускаются трейлеры и геймплейные ролики. А ближе к релизу блогерам и журналистам выдаются демо версии игры, для того чтобы они популяризировали ее среди своей аудитории.

Когда аудитория достаточно подготовлена, игру выпускают и начинается процесс поддержки. Выпускаются патчи с исправлением ошибок, добавляются новый контент. В онлайн проектах разработчики поддерживают сервера. Для поддержания здоровой атмосферы в сообществе присутствуют модераторы. Для удержания интереса игроков проводятся внутриигровые события, турниры и конкурсы.

По мере работы все отделы взаимодействуют друг с другом. А координирует всю работу отдел управления. Этот отдел получает данные и отправляет указания всем остальным отделам. Для аналитики того, что в данный момент происходит в компании менеджменту нужно получить данные о том, что сделано на данный момент, присутствуют ли факторы, мешающие работе, а также нужны ли дополнительные ресурсы. В свою очередь отдел управления отправляет общее видение этапов проекта на месяцы вперед, расставляют приоритеты задач, утверждают бюджеты и сроки.

Отдел разработки в этой цепочке является техническим центром, который превращает идеи и дизайн в работающий код. Для своей работы отдел разработчиков получает дизайн-документы, технические задания, готовые ассеты, данные о требованиях к движку и отчеты об ошибках. Результаты работы в виде: рабочих сборок игры, инструментов и плагинов, арт-требований, прототипов механик и тестовых сборок, отдел разработки отправляет остальным.

Художественный-отдел, создает визуальную и атмосферную составляющую игры, поэтому получает от геймдизайнера все, что передает

стиль игры, от разработчиков технические арт-требования, для интеграции ассетов в движок, а от управление утвержденные концепты и приоритеты по задачам. На выходе эти отделы получают концепт-арты, готовые ассеты и обратную связь по ТЗ. Приведу таблицу 1.1, описывающую передачу данных между отделами.

Таблица 1.1 - Входные и выходные данные основных отделов разработки

Отдел	Входные данные	Пример сущности	Выходные данные	Пример сущности
Отдел управления	Состояние готовности проекта; присутствие факторов, мешающих разработке; необходимость в дополнительных ресурсах	Дополнительные ресурсы - сущность Recources с атрибутами id, name, description, amount, reported_by, project_id, date_reported, priority	Общее видение этапов; приоритеты; бюджеты и сроки	Сроки - сущность Deadlines с атрибутами id, project_id, task_id, end_date
Отдел разработки	Дизайн-документы; ТЗ; готовые ассеты; данные о требованиях к движку; отчеты об ошибках	Готовые ассеты - сущность Assets с атрибутами id, name, type, file_size, version, author_id, project_id, storage_path, created_date, status	Рабочие сборки игры; инструменты и плагины; арт-требования; прототипы механик и тестовые сборки	Рабочие сборки - сущность Builds с атрибутами id, name, version, file_size, project_id, publication_date, storage_path, created_date, status
Арт-отдел	Требование геймдизайнера; арт-требования; утвержденные концепты и приоритеты по задачам	Утвержденные концепты - ApprovedConcepts с атрибутами id, name, version, file_size, type, author_id, project_id, storage_path, created_date, status	Концепт-арты; готовые ассеты; обратную связь	Обратная связь - сущность Feedback с атрибутами id, text, author_id, project_id, created_date, asset_id
Тестировщики	Сборка, требования к проекту, документацию	Требования к проекту - сущность ProjectRequirements с атрибутами id, text, author_id, created_date, status, project_id	Отчеты об ошибках, результаты тестов, отчеты о геймдизайне,	Результаты тестов - сущность TestResults с атрибутами id, author_id, text, created_date, project_id, test_type

На этом этапе можно выделить основные проблемы традиционных подходов к хранению данных:

- Отсутствие единой структуры для хранения данных препятствует свободной передачи информации между отделами. Более того, разные отделы

могут хранить данные в разных системах, что создает информационные острова. Последствиями такой ошибки являются несогласованность, дублирование работы и принятие решений на основе неполных данных;

- Кроме того, без единого центра хранения данных возникает проблема путаницы в актуальности данных. В общем доступе может находится три версии одного файла, и разработчики путаются, внося изменения в разные файлы;
- Создаются сложности в коммуникации между отделами, так как общение происходит в более неформальных переписках и не документируется в центре хранения данных. Из-за этого менеджеры тратят много времени на сбор информации от сотрудников, а не на управление студией. Это чревато замедленным принятием решений и быстрым устареванием данных в отчетах из-за невозможности быстро их обновить;
- Особенно остро такая децентрализация ощущается в арт-отделе. Так как нет четкой структуры папок, такая система приводит к потере ассетов, дублированию контента и увеличению размеров проекта за счет не нужных ассетов. Таким образом такая система неэффективна как с точки зрения организации данных, так и с точки зрения организации команды, так как невозможно определить: “Кто?”, “Когда?” и “Что?” изменил.

Таким образом, ключевой проблемой, препятствующей эффективной работе студий любого масштаба, является децентрализованное хранение данных. Это приводит к созданию информационных изоляторов, несогласованности версий файлов, сложностям в коммуникации и значительным операционным издержкам, что в конечном итоге ставит под угрозу успешную и своевременную реализацию даже самого перспективного проекта.

1.2 Реляционные базы данных как инструмент управления предприятием

Реляционные базы данных - это основа для реализации сложной бизнес-логики. А все потому, что у подхода построения бизнес-логики на основе реляционных баз данных есть ряд преимуществ:

- Целостность данных. Это важно, так как потеря, дублирование и искажения критически важных данных приведет к финансовым потерям. Этого позволяют достичь такие механизмы как ACID-транзакции. ACID-транзакции представляют собой атомарные операции, то есть они либо выполняются полностью, либо не выполняются совсем, при этом данные всегда переходят из одного состояния в другое, и если транзакция завершена то данные сохранены и уже не будут потеряны. Первичные ключи в свою очередь гарантируют уникальность записей, а внешние ссылочную целостность;
- Гибкость и мощность запросов через SQL. SQL является декларативным языком и описывает, “что нужно получить”. Таким образом составляются запросы, с помощью которых легко объединяются данные из множества таблиц. Более того в запросах можно использовать группировку и сортировку, что идеально для составления отчетности и аналитики. А если нужно проводить вычисления прямо в таблице, то это можно реализовать в самом запросе с помощью подзапросов и оконных функций;
- Нормализация данных и устранение аномалий. При хранении данных в Реляционной базе данных, они хранятся в одном месте, что устраняет избыточность. Возможность обновить какую-то часть данных только в одном месте, без потребности менять эти же данные в большом количестве записей, устраняет аномалию обновления. Это делает данные структурированными, предсказуемыми и легче поддающимися изменению;

- Безопасность и разграничение доступа. Реляционные СУБД предлагают тонко настроенную систему прав доступа. Можно дать бухгалтеру доступ только к финансовой отчетности, но не к персональным данным;
- Надежность. Реляционные СУБД существуют десятилетиями. Это означает, что они предсказуемы. А их надежность обеспечивается мощными инструментами администрирования, такими как резервное копирование, репликация и мониторинг.

Но на рынке существуют и NoSQL-решения, приведу сравнительную таблицу между реляционными базами данных, NoSQL базами данных и электронными таблицами. В таблице 1.2 приведено сравнение Реляционной СУБД, NoSQL и электронных таблиц.

Таблица 1.2 - Сравнительная таблица Реляционая СУБД, NoSQL и электронные таблицы

Критерий	Реляционные СУБД	NoSQL	Электронные таблицы
Структура данных	Жесткая схема, таблица с строгими типами	Гибкая, схемонезависимая	Свободная структура, в которой ячейки могут содержать что угодно
Целостность данных	Поддержка ACID-транзакций, внешние и внутренние ключи	BASE - подход, ориентированный на доступность в ущерб немедленной согласованности	Легко нарушаемая целостность
Масштабируемость	Вертикальная и горизонтальная	Горизонтальная	Ограничена
Производительность	Оптимизированы для сложных запросов с JOIN	Оптимизированы для простых операций и больших объемов	Медленно работает с большими данными
Бизнес-логика	Хранимые процедуры, триггеры, сложные SQL-запросы	Приложение берет на себя всю логику	Формулы, макросы

Рассмотрим случаи, когда NoSQL решения могут быть лучше чем реляционные базы данных.

Первый случай, когда нам требуется гибкость схемы данных, так как реляционные базы данных требуют предопределенной схемы, добавление нового поля означает миграцию всей системы. В NoSQL можно добавлять поля сразу, что идеально для быстро меняющихся требований.

Второй случай, когда нужно горизонтальное масштабирование. Горизонтальное разделение в реляционных базах данных является процессом, который требует сложной настройки. А NoSQL сразу спроектирован для распределенных систем.

Третий случай, когда наши данные неструктуризированы. Реляционные базы данных плохо подходят для иерархических, сложенных данных. NoSQL идеально подходит для JSON-подобных систем.

Четвертый случай, когда требуется высокая производительность записи. Транзакции и индексы замедляют массовую запись в реляционных базах данных. NoSQL оптимизированы для быстрой записи больших объёмов.

Теперь оратим внимание на электронные таблицы. Выше уже было сказано, что вариант электронных таблиц не выгоден для бизнеса, но рассмотрим ситуации, когда можно их использовать.

При быстром прототипировании допускается использование электронных таблиц, потому что с помощью них можно легко создать прототип отчета, тогда как в реляционной базе данных придется проектировать схемы и писать SQL.

Когда нам нужно провести спонтанный Ad-hoc анализ, таблицы предоставляют для этого понятный визуальный инструментарий, тогда как в реляционных базах данных придется использовать запросы.

На этом заканчиваются плюсы, которые не перекрывают недостатков данного способа хранения данных. Критическими проблемами электронных таблиц являются: сложное масштабирование, трудность в ручных правках, отсутствие целостности, проблемы с совместной работой и безопасность.

Остановившись на реляционной базе данных, приведем основные требования, которые необходимы для удобной работы команды над видеоигрой:

- Производительность и масштабируемость. Эти параметры нужны нашей базе данных, потому что при пиковых нагрузках может обрабатываться до 10000 запросов в секунду и наша база данных должна выдерживать. Кроме того для критичных операций необходима низкая задержка менее 50 миллисекунд при обращении к базе данных, например при возникновении сбоев в онлайн проектах, разработчики должны получать все данные с минимальными задержками, для быстрого реагирования на ситуацию. Масштабирование должно дать возможность обновлять отдельные узлы без простоев;
- Надежность и отказоустойчивость является не менее важными требованиями для базы данных, сервисы не могут позволить себе долгие простои в работе;
- Безопасность является важной составляющей любой базы данных, поэтому в реляционных базах данных реализованы технологии, позволяющие дать доступ к определенным данным, ограниченному числу лиц;
- Операционная эффективность нужна при поддержании проектов, для того чтобы отслеживать состояние системы в реальном времени и получать автоматические оповещения при возникновении каких-либо проблем. А для того, чтобы в случае критических проблем можно было вернуться к прошлой конфигурации системы используются ежедневные бэкапы и планы аварийного восстановления;
- Интеграция с инструментами разработки, например с языком python. Это поможет получать нужные данные, не выходя из своей среды разработки.

Таким образом реляционная база данных является единым центром хранения информации. Общение тоже происходит в самой базе данных, в ней отделы дают обратную связь, которая никуда не теряется и остается в системе.

Таким образом за счет четкой структуры хранения данных и документированию всего происходящего реляционная база данных решает все выделенные в предыдущей подглаве проблемы.

1.3 Методологии проектирования реляционных баз данных

Проектирование базы данных — это комплексная задача, включающая в себя множество этапов, основными из которых являются:

- Концептуальный;
- Логический;
- Физический.

Концептуальный этап является первым этапом в разработке информационных систем. На этом этапе определяется общая структура и концепция будущего продукта. Роль концептуального этапа заключается в создании абстрактной модели, которая будет служить основой для дальнейшей разработки и реализации проекта. На этом этапе выполняется анализ предметной области, специалист изучает бизнес-процессы, потребности пользователей и другую доступную информацию для полного понимания. Основной целью данного анализа является определение основных сущностей и связей, которые будут представлены в базе данных. Сущности представляют собой конкретные объекты или концепции. После создания сущностей, нужно определить виды связи между ними. Они указывают на взаимодействие и зависимости, связи бывают:

- Один-к-одному - каждая сущность одного типа связана с единственной сущностью другого типа;
- Один-ко-многим - каждая сущность одного типа связана с несколькими другого типа;
- Многие-ко-многим - множество сущностей одного типа связано со множеством другого типа.

При определении основных сущностей, их атрибутов и связей между ними вырабатываются основные ограничения и правила, которые должны быть применены к модели. Это могут быть ограничения целостности информации, бизнес-правила или другие условия, которые должны быть учтены при разработке БД.

Результатом концептуального проектирование является простая ER-диаграмма, иллюстрирующая только сущности и связи между ними. Пример такой диаграммы представлен ниже на рисунке 1.1.

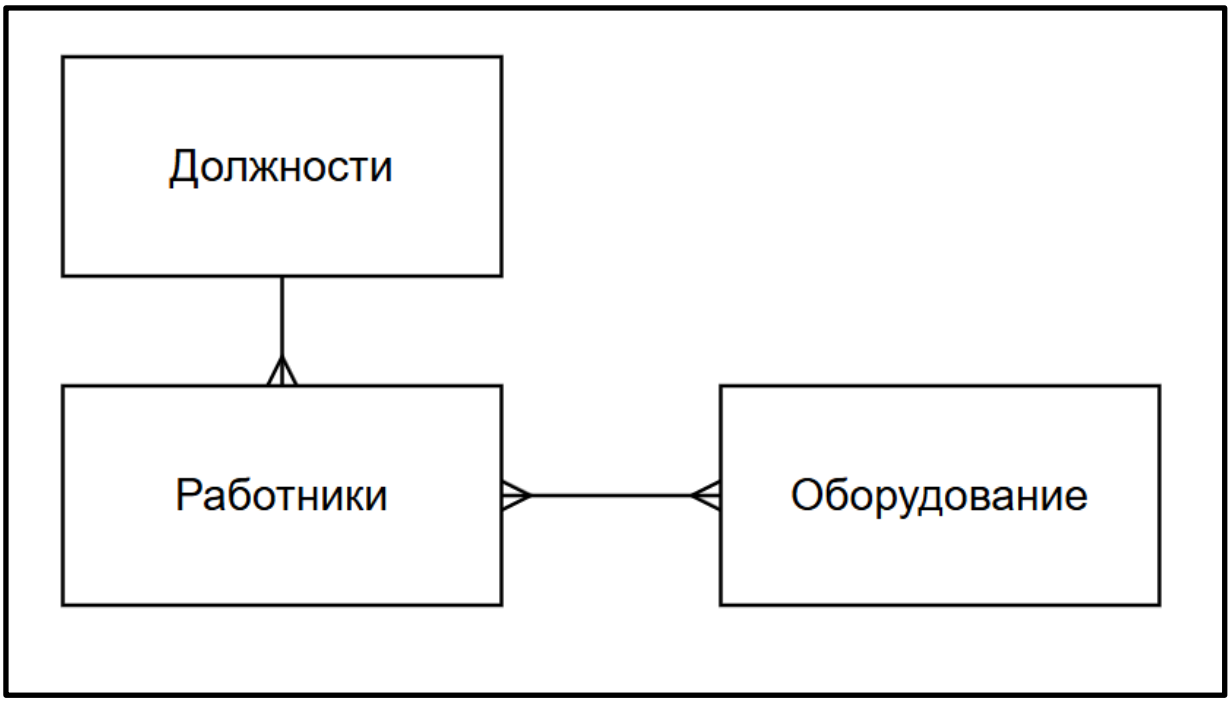


Рисунок 1.1 - пример ER-диаграммы

После создания ER-диаграммы на её основе начинается этап логического проектирования. Конечной целью является преобразование концептуальной модели в схему реляционной модели данных, нормализование структуры и подготовление её для реализации в СУБД.

На этапе логического проектирования концептуальная модель преобразуется в логическую структуру, совместимую с СУБД. Начинается работа с таблицами, полями и ключами. Также проводится нормализация данных, чтобы устранить избыточность и повысить эффективность хранения. При этом логическое проектирование не является хаотичным процессом, оно состоит из следующих взаимосвязанных этапов:

- Нормализация структуры данных - устранение избыточности и аномалий в модели путем применения форм нормализации;
- Определение бизнес-правил и ограничений - формулирование правил целостности, уникальности, обязательности и других ограничений, отражающих бизнес-логику;
- Проектирование процессов обработки данных — разработка логики основных операций создания, чтения, обновления и удаления данных (CRUD);
- Валидация и верификация модели — проверка соответствия логической модели исходным требованиям, устранение противоречий и неоднозначностей;
- Документирование результатов — создание формальной документации логической модели с использованием общепринятых нотаций и стандартов.

Когда у нас уже есть логическая схема, начинается физическое проектирование. На этом этапе проектировщик принимает решения о способах реализации разрабатываемой базы данных. Основной целью физического проектирования является описание способа физической реализации логическйо схемы базы данных. Под этим подразумевается следующее:

- Создание набора реляционных таблиц и ограничений для них на основе информации, представленной в логической модели данных;
- Определение структур хранения данных и методов доступа к ним;
- Разработка средств защиты.

При этом важно понимать, что разработка базы данных это итеративный процесс, в течении которого проектировщик находит ошибки и возвращается на прошлые этапы чтобы их исправить.

После рассмотренных этапов углубимся в инструменты проектирования и начнем с ER-модели и ее нотаций. ER-модель - модель, описывающая изучаемую предметную область с помощью ограниченного набора разнотипных элементов. Основными элементами ER-диаграмм являются:

сущности и связи, про них рассказывалось ранее. На логическом эапе к сущностям и связам добавляются атрибуты, содаржащие в себе данные о сущности.

Существует множество нотаций ER-диаграмм, которые отличаются друг от друга правилами оформления. Выбор нотации зависит от предпочтений проектировщика так как, в сущности, все они представляют одно и то же. Рассмотрим основные нотации.

Нотация Чена является первой созданной нотацией ER-диаграмм. В этой нотации сущность отображается прямоугольником, ее атрибуты овалами, а отношения между сущностями - ромбами. Имена ключевых атрибутов подчеркиваются. На дугах можно поставить кратности связей и ограничения. Ниже, на рисунке 1.2, представлена диаграмма в нотации Чена.

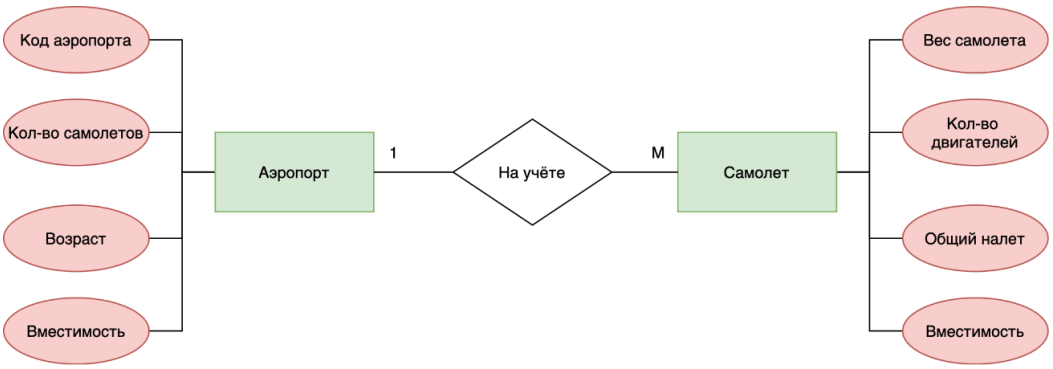


Рисунок 1.2 - ER-диаграмма в нотации Чена

Нотация Мартина. Атрибуты записываются внутри прямоугольника, обозначающего сущность. У атрибутов можно указать типы данных. Отношения не отображаются в виде ромбов, а записываются в виде текста над стрелкой. Нотация предусматривает 4 вида окончания стрелок. В данной работе ER-диаграмма будет построена в нотации Мартина, так как она компактна, позволяет сразу расписать типы данных и просто удобнее для просмотра. Ниже, на рисунке 1.3, приведен пример диаграммы в нотации Мартина.

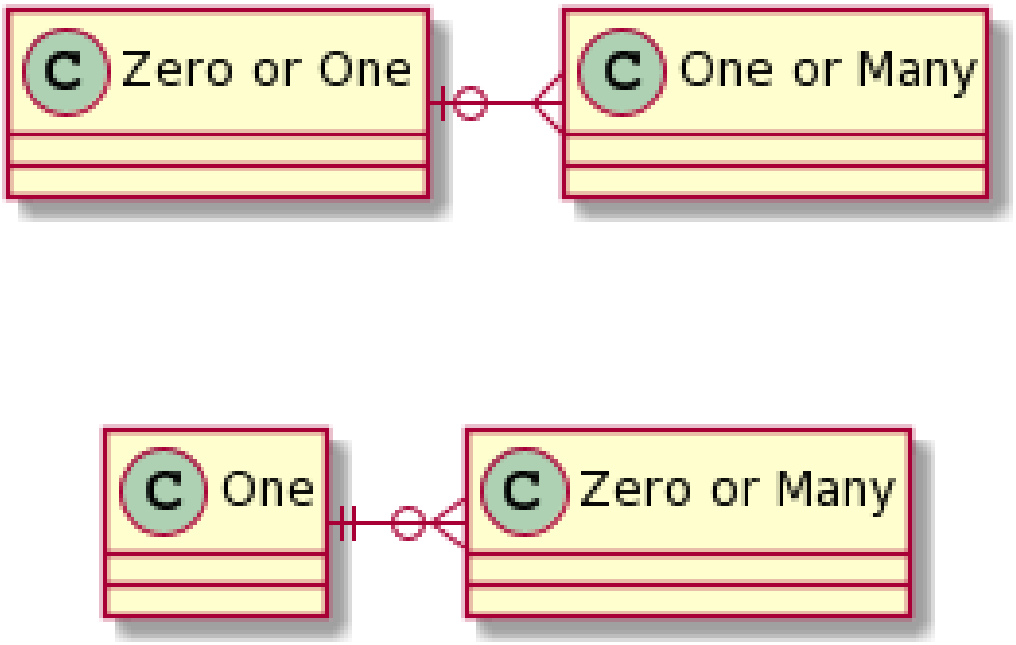


Рисунок 1.3 -ER-диаграмма в нотации Мартина

Нотация диаграммы классов UML. Сущности, называющиеся в этом методе классами, и их атрибуты отображаются так же, как в нотации Мартина, но сами по себе сущности описываются более подробно: расписываются типы данных атрибутов, первичные и внешние ключи, методы сущностей. Связи между сущностями представляются: ассоциацией (отношения между экземплярами классов), агрегацией (связь целое-часть) и наследованием (общее-частное). Эта нотация удобна в разработке на основе объектов, так как современные системы способны обрабатывать такие диаграммы и создавать примитивные шаблоны на их основе. Ниже, на рисунке 1.4, представлен пример диаграммы классов UML.

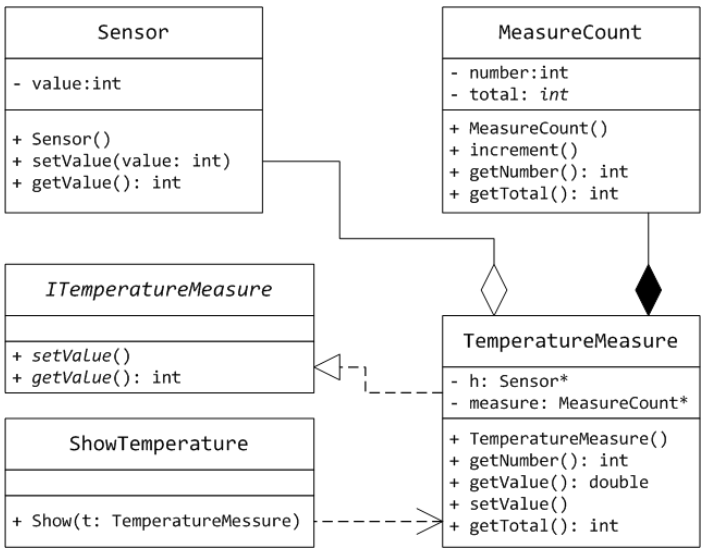


Рисунок 1.4 - ER-диаграмма классов UML

На логическом этапе происходит нормализация данных. Нормализация - это способ организации данных, в нормализованной базе данных нет повторяющейся информации и с ней проще работать. В процессе нормализации данные преобразуют, чтобы они занимали меньше места, а поиск по элементам был быстрым и результативным. Существует семь нормальных форм. Приводить данные к нормальным формам можно только последовательно. Обычно данные нормализуют до третьей нормальной формы. Разберем основные, применяемые формы нормализации:

- Первая нормальная форма - в базе данных не должно быть дубликатов и составных данных. Элементы составных данных лучше разнести по разным полям, иначе в процессе работы могут появиться ошибки и аномалии;
- Вторая нормальная форма - у каждой записи в базе данных должен быть первичный ключ. Первичный ключ — это элемент записи, который не повторяется в других записях;
- Третья нормальная форма - в записи не должно быть столбцов с неключевыми значениями, которые зависят от других неключевых значений;
- Нормальная форма Бойса-Кодда (НФБК) - является частной формой третьей нормальной формы. Чтобы привести к НФБК, необходимо,

чтобы если какое-то поле определяет значение других полей, то это поле само по себе должно быть ключом.

Теперь, когда изложение теории закончено, нужно приступать к практике. В качестве инструмента проектирования базы данных я выберу Draw.io. Draw.io является бесплатным инструментом для создания диаграмм и схем. Ниже приведу таблицу (таблица 1.3) с основными плюсами и причинами выбора Draw.io.

Таблица 1.3 - Ключевые особенности Draw.io

Аспект	Описание	Почему это важно
Модель распространения	Бесплатное программное обеспечение с открытым исходным кодом	Нет затрат на лицензию и все функции доступны без скрытых платежей
Доступность	Онлайн в браузере и как приложение, не требующее регистрации	Можно начать работу быстро без привязки к устройству
Функциональность	Создание ER-диаграм, UML, BPMN, проектирование архитектуры, прототипирование интерфейсов	Универсальный инструмент, покрывающий множество задач аналитика, разработчика и архитектора.
Интеграции и экспорт	Сохранение на устройство, в Google Диск, OneDrive, GitHub, GitLab. Экспорт в PNG, SVG, PDF, HTML и др.	Гибкость в работе и совместной работе, удобство для вставки в документацию.
Интерфейс и удобство	Интуитивно понятный, переведенный на русский язык интерфейс.	Быстрая адаптация, не требует прохождения длительного обучения.

В следующей главе будет показана реализация базы данных для игровой студии поэтапно. Использовать будем Draw.io для реализации ER-диаграммы, а с помощью языка запросов SQL реализуем физический этап.

ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ ДЛЯ ИГРОВОЙ СТУДИИ

2.1 Проектирование концептуальной модели данных

Определим сущности базы данных.

Affiliated_studios, сущность с информацией о дочерних студиях, связывается с сущностью staff через отношение многие-к-одному, где один сотрудник может руководить несколькими студиями, но каждая студия имеет единственного руководителя, что обеспечивает четкую иерархию управления. Affiliated_studios также формируют отношения один-ко-многим с affiliated_studios_on_projects, game_engines и staff_at_the_affiliated_studios, определяя участие в проектах, разработку движков и состав сотрудников, создавая комплексную структуру взаимодействия.

Affiliated_studios_on_projects служит связующей сущностью, реализующей отношение многие-ко-многим между projects и affiliated_studios, где каждый проект может включать несколько студий, а каждая студия может работать над множеством проектов одновременно, обеспечивая гибкость распределения ресурсов.

Budgets_for_project, бюджеты для проектов, относятся к projects через связь многие-к-одному, позволяя каждому проекту содержать несколько бюджетных записей для разных периодов, при этом каждая запись привязана к единственному проекту, что обеспечивает детальный финансовый контроль.

Build_types как справочная сущность связаны с builds отношением один-ко-многим, где каждый тип сборки может характеризовать множество сборок, но каждая сборка принадлежит к одному типу, обеспечивая стандартизацию процесса разработки.

Builds устанавливают связи многие-к-одному с projects и build_types, определяя принадлежность сборки к проекту и её тип, что создает прозрачную

структуру управления версиями. Builds также порождает отношение один-ко-многим с test_reports, фиксируя историю тестирования каждой сборки.

Equipment_maintenance соединяется отношениями многие-к-одному с equipments, maintenance_types и staff, регистрируя обслуживание оборудования определённого типа, выполненное конкретным техником.

Equipment_statuses как справочник связаны с equipments отношением один-ко-многим, где каждый статус может применяться к множеству единиц оборудования.

Equipment_types через отношение один-ко-многим с equipments классифицируют оборудование по типам, позволяя одному типу включать множество экземпляров.

Equipments имеют три связи многие-к-одному с equipment_types, equipment_statuses и staff, определяя тип, статус и ответственного сотрудника. Equipments также формирует отношения один-ко-многим с equipment_maintenance и equipments_for_projects, фиксируя историю обслуживания и использование в проектах.

Equipments_for_projects как реализует отношение многие-ко-многим между projects и equipments, позволяя оборудованию использоваться в нескольких проектах, а проектам задействовать несколько единиц оборудования.

Expense_categories через отношение один-ко-многим с expenses группируют расходные операции по категориям, где каждая категория объединяет множество расходов.

Expenses связаны отношениями многие-к-одному с projects, staff и expense_categories, фиксируя принадлежность расхода к проекту, связь с сотрудником и классификацию.

Game_engines относятся к affiliated_studios через связь многие-к-одному, определяя студию-разработчика движка. Game_engines формируют отношение

один-ко-многим с platforms_and_game_engines_for_projects, определяя использование движков в проектах.

Genres как справочная сущность связаны с genres_for_projects отношением один-ко-многим, позволяя каждому жанру назначаться на несколько проектов.

Genres_for_projects служат связующей таблицей для отношения многие-ко-многим между projects и genres, обеспечивая назначение нескольких жанров на проект и использование жанра в нескольких проектах.

Languages через отношение один-ко-многим с suported_languages определяют языки локализации, поддерживаемые различными проектами.

Libraries связаны с libreries_for_project отношением один-ко-многим, где каждая библиотека может использоваться в нескольких проектах.

Libreries_for_project реализуют отношение многие-ко-многим между projects и libraries, регулируя использование библиотек в проектах.

Maintenance_types как справочник связаны с equipment_maintenance отношением один-ко-многим, классифицируя типы выполняемого обслуживания.

Milestone_statuses через отношение один-ко-многим с milestones определяют возможные статусы вех проекта.

Milestones относятся к projects связью многие-к-одному, фиксируя вехи конкретного проекта. Milestones также формируют отношение один-ко-многим с milestones_statuses, отслеживая историю изменения статусов.

Platforms связаны с platforms_and_game_engines_for_projects отношением один-ко-многим, определяя платформы, поддерживаемые проектами.

Platforms_and_game_engines_for_projects служат сложной связующей таблицей для отношения многие-ко-многим между projects, game_engines и platforms, определяя технологический стек каждого проекта.

Posts как справочник должностей связаны со staff отношением один-ко-многим, где каждая должность может заниматься множеством сотрудников.

Projects имеют связь многие-к-одному с projects_statuses для определения текущего статуса. Projects также формируют несколько отношений один-ко-многим с milestones, tasks, budgets_for_project, builds, expenses и всеми связующими таблицами, выступая центральным элементом системы.

Projects_statuses через отношение один-ко-многим с projects определяют возможные статусы проектов.

Reports_statuses как справочник связан с test_reports отношением один-ко-многим, классифицируя статусы тестовых отчётов.

Roles через отношение один-ко-многим со staff_on_projects определяют роли сотрудников в проектах.

Staff относятся к posts связью многие-к-одному, определяя должность сотрудника. Staff также формируют множество отношения один-ко-многим с affiliated_studios, equipment_maintenance, equipments, expenses, tasks, test_reports и связующими таблицами, отражая роль сотрудников в системе.

Staff_at_the_affiliated_studios реализуют отношение многие-ко-многим между affiliated_studios и staff, формируя кадровый состав дочерних студий.

Staff_on_projects как связующая таблица устанавливают отношение многие-ко-многим между projects и staff с дополнительной связью многие-к-одному с roles, определяя состав проектных команд с назначением ролей.

Suported_languages служат связующей таблицей для отношения многие-ко-многим между languages и projects, определяя языковую поддержку каждого проекта.

Task_statuses как справочник связаны с tasks отношением один-ко-многим, определяя возможные статусы задач.

Tasks имеют три связи многие-к-одному с projects, staff и task_statuses, фиксируя принадлежность задачи к проекту, назначение исполнителя и текущий статус.

Test_reports связаны отношениями многие-к-одному с projects, builds, staff и reports_statuses, регистрируя тестовые отчёты по проектам, сборкам, с указанием тестировщика и статуса отчёта.

На основе концептуальной модели была спроектированна логическая схема базы данных, по которой была спроектированна ER-диаграмма которая представленна в приложении 1.

2.2 Описание логической схемы базы данных

Affiliated_studios — это дочерние студии разработки с атрибутами id, name, birth_date, head_gamedesigner_id, address, email, phone, хранящие полную информацию, включая дату основания и контакты. Они связаны отношением многие-к-одному со staff через head_gamedesigner_id для определения руководства, а также имеют связи один-ко-многим с affiliated_studios_on_projects (для участия в проектах), game_engines (для учёта движков) и staff_at_the_affiliated_studios (для формирования кадрового состава).

Affiliated_studios_on_projects — это связующая таблица с атрибутами id, project_id и affiliated_studio_id, организующая отношение многие-ко-многим между проектами и дочерними студиями.

Таблица budgets_for_project служит для отслеживания бюджетов проектов. Она содержит атрибуты id, project_id, allocated_amount, actual_spent и fiscal_year, которые фиксируют запланированные и фактические расходы в разрезе фискальных лет. Связь многие-к-одному с таблицей projects привязывает финансовые записи к конкретным проектам.

Таблица build_types является справочником типов сборок с атрибутами id и name. Через связь один-ко-многим с таблицей builds каждый тип может характеризовать множество экземпляров сборок.

Таблица builds хранит информацию о сборках проектов, включая атрибуты id, project_id, version, build_date, build_type_id и notes для контроля версионирования и метаданных. Она связана отношениями многие-к-одному с таблицами projects (для определения проекта) и build_types (для классификации типа сборки), а также имеет связь один-ко-многим с test_reports для отслеживания связанных отчётов о тестировании.

Таблица equipment_maintenance регистрирует операции технического обслуживания с атрибутами id, equipment_id, maintenance_date, maintenance_type, cost, technician_id и notes. Она связана отношениями многие-к-одному с таблицами equipments (для привязки к оборудованию), maintenance_types (для определения типа процедуры) и staff (для идентификации технического специалиста).

Таблица equipment_statuses является справочником статусов оборудования с атрибутами id и name и связана отношением один-ко-многим с таблицей equipments, что позволяет присваивать каждый статус множеству единиц оборудования.

Таблица equipment_types служит справочником категорий оборудования с атрибутами id и name и связана отношением один-ко-многим с таблицей equipments, что гарантирует отнесение каждой единицы оборудования к конкретному типу.

Таблица equipments ведет инвентарный учет оборудования с атрибутами id, equipment_type_id, equipment_status_id, date_of_receipt и staff_responsible_for_equipment_id. Она связана отношениями многие-к-одному с таблицами equipment_types (тип), equipment_statuses (статус) и staff (ответственный сотрудник), а также имеет связи один-ко-многим с equipment_maintenance (история обслуживания) и equipments_for_projects (использование в проектах).

Таблица equipments_for_projects является связующей между проектами и оборудованием, содержит атрибуты id, project_id, equipment_id и реализует связь многие-ко-многим через отношения многие-к-одному с таблицами projects и equipments, обеспечивая гибкое распределение ресурсов между проектами.

Таблица expense_categories является справочником категорий финансовых затрат (например, зарплаты или закупки) с атрибутами id и name и связана отношением один-ко-многим с таблицей expenses, что позволяет каждой категории группировать множество расходных операций для детализированного финансового анализа.

Таблица expenses фиксирует финансовые операции компании через атрибуты id, project_id, staff_id, amount, expense_date, category_id и description. Связи многие-к-одному с таблицами projects, staff и expense_categories обеспечивают точную атрибуцию каждого расхода к соответствующему проекту, сотруднику и финансовой категории.

Таблица game_engines хранит данные о игровых движках с атрибутами id, name, version и affiliated_studio_developer_id. Она связана отношением многие-к-одному с affiliated_studios для идентификации студии-разработчика и имеет связь один-ко-многим с platforms_and_game_engines_for_projects для учёта применения движков в различных проектах.

Таблица genres является справочником игровых жанров с атрибутами id и name и связана отношением один-ко-многим с таблицей genres_for_projects, что позволяет присваивать каждый жанр множеству проектов для гибкого жанрового определения продуктов.

Таблица genres_for_projects является связующей для определения жанровой принадлежности проектов, содержит атрибуты id, project_id и genre_id и связана отношениями многие-к-одному с таблицами projects и genres.

Таблица languages является справочником языков локализации с атрибутами id и name и связана отношением один-ко-многим с таблицей

supported_languages, что позволяет каждому языку поддерживаться в нескольких проектах для обеспечения глобального распространения продуктов.

Таблица libraries учитывает сторонние библиотеки с атрибутами id, name и version и связана отношением один-ко-многим с таблицей libraries_for_project, что позволяет каждой библиотеке использоваться в нескольких проектах для оптимизации разработки.

Таблица libraries_for_project служит связующим звеном для управления зависимостями между проектами и библиотеками. Она содержит атрибуты id, library_id и project_id и связана отношениями многие-к-одному с таблицами projects и libraries, реализуя связь многие-ко-многим для эффективного управления программными компонентами.

Таблица maintenance_types является справочником типов технического обслуживания с атрибутами id и name и связана отношением один-ко-многим с таблицей equipment_maintenance, что позволяет каждому типу обслуживания характеризовать множество операций для обеспечения согласованности технических процедур.

Таблица milestone_statuses является справочником возможных состояний вех проекта с атрибутами id и name и связана отношением один-ко-многим с таблицей milestones, что позволяет присваивать каждый статус нескольким вехам для последовательного управления их жизненным циклом.

Таблица milestones фиксирует ключевые этапы разработки проектов через атрибуты id, project_id, name, description, due_date, actual_completion_date и status_id. Она связана отношениями многие-к-одному с таблицами projects (для привязки к конкретному проекту) и milestone_statuses (для отслеживания текущего статуса и истории изменений вех).

Таблица platforms является справочником целевых платформ с атрибутами id и name и связана отношением один-ко-многим с таблицей platforms_and_game_engines_for_projects, что позволяет каждой платформе

поддерживаться в нескольких проектах для реализации стратегии многоплатформенной разработки.

Таблица platforms_and_game_engines_for_projects служит для определения технологического стека проектов и содержит атрибуты id, project_id, game_engine_id и platform_id. Она связана отношениями многие-к-одному с таблицами projects, game_engines и platforms, формируя комплексную систему управления технологическими решениями проектов.

Таблица posts является справочником должностей с атрибутами id, name и salary и связана отношением один-ко-многим с таблицей staff, что позволяет каждой должности заниматься несколькими сотрудниками и формирует организационную структуру компании.

Таблица projects является центральной сущностью системы и содержит полную информацию об игровых проектах через атрибуты id, name, project_start_date, planned_end_date, actual_end_date, version, description и status_id. Она связана отношением многие-к-одному с projects_statuses для определения текущего статуса проекта, а также имеет связи один-ко-многим с таблицами milestones, tasks, budgets_for_project, builds, expenses и всеми связующими таблицами системы, что обеспечивает интеграцию всех компонентов управления проектами.

Таблица projects_statuses является справочником возможных состояний проектов с атрибутами id и name и связана отношением один-ко-многим с таблицей projects, что позволяет присваивать каждый статус нескольким проектам для эффективного отслеживания прогресса разработки.

Таблица reports_statuses является справочником статусов отчетов с атрибутами id и name и связана отношением один-ко-многим с таблицей test_reports, что позволяет каждому статусу характеризовать множество тестовых отчетов для объективной оценки качества сборок.

Таблица roles является справочником ролей в проектах с атрибутами id и name и связана отношением один-ко-многим с таблицей staff_on_projects, что

позволяет назначать каждую роль нескольким сотрудникам в различных проектах для создания гибкой системы управления командами.

Таблица staff содержит полные кадровые данные сотрудников через атрибуты id, first_name, last_name, middle_name, birth_date, address, post_id, hire_date, email, phone и gender. Она связана отношением многие-к-одному с posts для определения должности и имеет множественные связи один-ко-многим с таблицами affiliated_studios (руководство студией), equipment_maintenance (техобслуживание), equipments (ответственность за оборудование), expenses (финансовые операции), tasks (задачи) и test_reports (тестирование), а также со связующими таблицами, отражая многогранное участие сотрудников в процессах компании.

Таблица staff_at_the_affiliated_studios является связующей для формирования кадрового состава студий, содержит атрибуты id, affiliated_studio_id и staff_id и связана отношениями многие-к-одному с таблицами affiliated_studios и staff, реализуя связь многие-ко-многим для оптимального распределения человеческих ресурсов между студиями.

Таблица staff_on_projects является связующей для управления участием сотрудников в проектах и содержит атрибуты id, project_id, staff_id, role_id, start_date, end_date и hourly_rate. Она связана отношениями многие-к-одному с таблицами projects, staff и roles, реализуя связь многие-ко-многим между проектами и сотрудниками с определением ролевых функций и условий работы в каждом проекте.

Таблица supported_languages является связующей для определения языковой поддержки проектов и содержит атрибуты id, language_id и project_id. Она связана отношениями многие-к-одному с таблицами languages и projects, реализуя связь многие-ко-многим между языками и проектами для комплексного управления локализацией.

Таблица task_statuses является справочником статусов задач с атрибутами id и name и связана отношением один-ко-многим с таблицей tasks,

что позволяет присваивать каждый статус нескольким задачам для эффективного мониторинга рабочего процесса.

Таблица tasks управляет рабочими заданиями проектов через атрибуты id, project_id, name, description, assignee_id, status_id, priority, created_date, due_date и estimated_hours, детализируя все аспекты задач. Она связана отношениями многие-к-одному с таблицами projects (для привязки к проекту), staff (для назначения исполнителя) и task_statuses (для отслеживания текущего статуса выполнения).

Таблица test_reports фиксирует результаты тестирования через атрибуты id, project_id, build_id, tester_id, test_date, status_id и note, документируя процесс контроля качества. Она связана отношениями многие-к-одному с таблицами projects, builds, staff и reports_statuses, обеспечивая полную атрибуцию каждого отчета к соответствующему проекту, сборке, тестировщику и статусу.

2.3 Описание физической схемы базы данных

Физическая схема базы данных представленна в приложении 2.

Таблица affiliated_studios содержит атрибуты: id (первичный ключ), name (название студии), birth_date (дата основания), head_gamedesigner_id (внешний ключ на руководителя), address (адрес), email (электронная почта) и phone (телефон). Все атрибуты имеют ограничение NOT NULL, а email и phone также имеют ограничение UNIQUE. Код для создания таблицы affiliated_studios приведен в рисунке 2.1.

```sql
CREATE TABLE affiliated_studios (
  id INTEGER,
  name VARCHAR(50) NOT NULL,
  birth_date DATE NOT NULL,
  head_gamedesigner_id INTEGER NOT NULL,
  address VARCHAR(50) NOT NULL,
  email VARCHAR(50) NOT NULL UNIQUE,
  phone VARCHAR(50) NOT NULL UNIQUE,
  PRIMARY KEY (id),
  FOREIGN KEY (head_gamedesigner_id) REFERENCES staff(id)
);
```

Рисунок 2.1 - Создание таблицы affiliated_studios

Таблица affiliated_studios_on_projects содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект) и affiliated_studio_id (внешний ключ на студию). Все атрибуты имеют ограничение NOT NULL. Код для создания таблицы affiliated_studios_on_projects приведен в рисунке 2.2.

```sql
CREATE TABLE affiliated_studios_on_projects (
  id INTEGER,
  project_id INTEGER NOT NULL,
  affiliated_studio_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (project_id) REFERENCES projects(id),
  FOREIGN KEY (affiliated_studio_id) REFERENCES affiliated_studios(id)
);
```

Рисунок 2.2 - Создание таблицы affiliated_studios_on_projects

Таблица budgets_for_project содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект), allocated_amount (выделенная сумма), actual_spent (фактически потраченная сумма) и fiscal_year (фискальный год). Все атрибуты имеют ограничение NOT NULL. Код для создания таблицы budgets_for_project приведен в рисунке 2.3.

```sql
CREATE TABLE budgets_for_project (
  id INTEGER,
  project_id INTEGER NOT NULL,
  allocated_amount DECIMAL(10, 2) NOT NULL,
  actual_spent DECIMAL(10, 2) NOT NULL,
  fiscal_year VARCHAR(10) NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (project_id) REFERENCES projects(id)
);
```

Рисунок 2.3 - Создание таблицы budgets_for_project

Таблица build_types содержит атрибуты: id (первичный ключ) и name (название типа сборки с ограничением NOT NULL). Код для создания таблицы build_types приведен в рисунке 2.4.

```
CREATE TABLE build_types (
  id INTEGER,
  name VARCHAR(50) NOT NULL,
  PRIMARY KEY (id)
);
```

Рисунок 2.4 - Создание таблицы build_types

Таблица builds содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект), version (версия сборки), build_date (дата сборки), build_type_id (внешний ключ на тип сборки) и notes (заметки). Все атрибуты имеют ограничение NOT NULL, а version также имеет ограничение UNIQUE. Код для создания таблицы builds приведен в рисунке 2.5.

```
CREATE TABLE builds (
  id INTEGER,
  project_id INTEGER NOT NULL,
  version VARCHAR(50) NOT NULL UNIQUE,
  build_date DATE NOT NULL,
  build_type_id INTEGER NOT NULL,
  notes TEXT NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (project_id) REFERENCES projects(id),
  FOREIGN KEY (build_type_id) REFERENCES build_types(id)
);
```

Рисунок 2.5 - Создание таблицы builds

Таблица equipment_maintenance содержит атрибуты: id (первичный ключ), equipment_id (внешний ключ на оборудование), maintenance_date (дата обслуживания), maintenance_type (внешний ключ на тип обслуживания), cost (стоимость), technician_id (внешний ключ на техника) и notes (примечания). Все атрибуты имеют ограничение NOT NULL. Код для создания таблицы equipment_maintenance приведен в рисунке 2.6.

```sql
CREATE TABLE equipment_maintenance (
  id INTEGER,
  equipment_id INTEGER NOT NULL,
  maintenance_date DATE NOT NULL,
  maintenance_type INTEGER NOT NULL,
  cost DECIMAL(10, 2) NOT NULL,
  technician_id INTEGER NOT NULL,
  notes TEXT NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (equipment_id) REFERENCES equipments(id),
  FOREIGN KEY (technician_id) REFERENCES staff(id),
  FOREIGN KEY (maintenance_type) REFERENCES maintenance_types(id)
);
```

Рисунок 2.6 - Создание таблицы equipment_maintenance

Таблица equipment_statuses содержит атрибуты: id (первичный ключ) и name (название статуса оборудования с ограничениями NOT NULL и UNIQUE). Код для создания таблицы equipment_statuses приведен в рисунке 2.7.

```sql
CREATE TABLE equipment_statuses (
  id INTEGER,
  name VARCHAR(50) NOT NULL UNIQUE,
  PRIMARY KEY (id)
);
```

Рисунок 2.7 - Создание таблицы equipment_statuses

Таблица equipment_types содержит атрибуты: id (первичный ключ) и name (название типа оборудования с ограничениями NOT NULL и UNIQUE). Код для создания таблицы equipment_types приведен в рисунке 2.8.

```sql
CREATE TABLE equipment_types (
  id INTEGER,
  name VARCHAR(50) NOT NULL UNIQUE,
  PRIMARY KEY (id)
);
```

Рисунок 2.8 - Создание таблицы equipment_types

Таблица equipments содержит атрибуты: id (первичный ключ), equipment_type_id (внешний ключ на тип оборудования), equipment_status_id (внешний ключ на статус оборудования), date_of_receipt (дата поступления) и staff_responsible_for_equipment_id (внешний ключ на ответственного

сотрудника). Все атрибуты имеют ограничение NOT NULL. Код для создания таблицы equipments приведен в рисунке 2.9.

```
CREATE TABLE equipments (
  id INTEGER,
  equipment_type_id INTEGER NOT NULL,
  equipment_status_id INTEGER NOT NULL,
  date_of_receipt DATE NOT NULL,
  staff_responsible_for_equipment_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (equipment_type_id) REFERENCES equipment_types(id),
  FOREIGN KEY (equipment_status_id) REFERENCES equipment_statuses(id),
  FOREIGN KEY (staff_responsible_for_equipment_id) REFERENCES staff(id)
);
```

Рисунок 2.9 - Создание таблицы equipments

Таблица equipments_for_projects содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект) и equipment_id (внешний ключ на оборудование). Все атрибуты имеют ограничение NOT NULL. Код для создания таблицы equipments_for_projects приведен в рисунке 2.10.

```
CREATE TABLE equipments_for_projects (
  id INTEGER,
  project_id INTEGER NOT NULL,
  equipment_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (project_id) REFERENCES projects(id),
  FOREIGN KEY (equipment_id) REFERENCES equipments(id)
);
```

Рисунок 2.10 - Создание таблицы equipments_for_projects

Таблица expense_categories содержит атрибуты: id (первичный ключ) и name (название категории с ограничениями NOT NULL и UNIQUE). Код для создания таблицы expense_categories приведен в рисунке 2.11.

```
CREATE TABLE expense_categories (
  id INTEGER,
  name VARCHAR(50) NOT NULL UNIQUE,
  PRIMARY KEY (id)
);
```

Рисунок 2.11 - Создание таблицы expense_categories

Таблица expenses содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект), staff_id (внешний ключ на сотрудника), amount (сумма), expense_date (дата расхода), category_id (внешний ключ на категорию) и description (описание). Атрибуты amount, expense_date, category_id и description имеют ограничение NOT NULL, тогда как project_id и staff_id могут

содержать NULL значения. Код для создания таблицы expenses приведен в рисунке 2.12.

```
92 CREATE TABLE expenses (
93   id INTEGER,
94   project_id INTEGER,
95   staff_id INTEGER,
96   amount DECIMAL(10, 2) NOT NULL,
97   expense_date DATE NOT NULL,
98   category_id INTEGER NOT NULL,
99   description TEXT NOT NULL,
100  PRIMARY KEY (id),
101  FOREIGN KEY (project_id) REFERENCES projects(id),
102  FOREIGN KEY (staff_id) REFERENCES staff(id),
103  FOREIGN KEY (category_id) REFERENCES expense_categories(id)
104 );
```

Рисунок 2.12 - Создание таблицы expenses

Таблица game_engines содержит атрибуты: id (первичный ключ), name (название движка), version (версия) и affiliated_studio_developer_id (внешний ключ на студию-разработчика). Все атрибуты имеют ограничение NOT NULL, а name и version также имеют ограничение UNIQUE. Код для создания таблицы game_engines приведен в рисунке 2.13.

```
105 CREATE TABLE game_engines (
106   id INTEGER,
107   name VARCHAR(50) NOT NULL UNIQUE,
108   version VARCHAR(50) NOT NULL UNIQUE,
109   affiliated_studio_developer_id INTEGER NOT NULL,
110   PRIMARY KEY (id),
111   FOREIGN KEY (affiliated_studio_developer_id) REFERENCES affiliated_studios(id)
112 );
```

Рисунок 2.13 - Создание таблицы game_engines

Таблица genres содержит атрибуты: id (первичный ключ) и name (название жанра с ограничениями NOT NULL и UNIQUE). Код для создания таблицы genres приведен в рисунке 2.14.

```
113 CREATE TABLE genres (
114   id INTEGER,
115   name VARCHAR(50) NOT NULL UNIQUE,
116   PRIMARY KEY (id)
117 );
```

Рисунок 2.14 - Создание таблицы genres

Таблица genres_for_projects содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект) и genre_id (внешний ключ на жанр). Все

атрибуты имеют ограничение NOT NULL. Код для создания таблицы genres_for_projects приведен в рисунке 2.15.

```
118 CREATE TABLE genres_for_projects (
119   id INTEGER,
120   project_id INTEGER NOT NULL,
121   genre_id INTEGER NOT NULL,
122   PRIMARY KEY (id),
123   FOREIGN KEY (project_id) REFERENCES projects(id),
124   FOREIGN KEY (genre_id) REFERENCES genres(id)
125 );
```

Рисунок 2.15 - Создание таблицы genres_for_projects

Таблица languages содержит атрибуты: id (первичный ключ) и name (название языка с ограничениями NOT NULL и UNIQUE). Код для создания таблицы languages приведен в рисунке 2.16.

```
126 CREATE TABLE languages (
127   id INTEGER,
128   name VARCHAR(50) NOT NULL UNIQUE,
129   PRIMARY KEY (id)
130 );
```

Рисунок 2.16 - Создание таблицы languages

Таблица libraries содержит атрибуты: id (первичный ключ), name (название библиотеки) и version (версия). Все атрибуты имеют ограничение NOT NULL. Код для создания таблицы libreries приведен в рисунке 2.17.

```
131 CREATE TABLE libraries (
132   id INTEGER,
133   name VARCHAR(50) NOT NULL,
134   version VARCHAR(50) NOT NULL,
135   PRIMARY KEY (id)
136 );
```

Рисунок 2.17 - Создание таблицы libraries

Таблица libreries_for_project содержит атрибуты: id (первичный ключ), librarie_id (внешний ключ на библиотеку) и project_id (внешний ключ на проект). Все атрибуты имеют ограничение NOT NULL. Код для создания таблицы libreries_for_project приведен в рисунке 2.18.

```
CREATE TABLE libreries_for_project (
  id INTEGER,
  librarie_id INTEGER NOT NULL,
  project_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (project_id) REFERENCES projects(id),
  FOREIGN KEY (librarie_id) REFERENCES libraries(id)
);
```

Рисунок 2.18 - Создание таблицы libreries_for_project

Таблица maintenance_types содержит атрибуты: id (первичный ключ) и name (название типа обслуживания с ограничениями NOT NULL и UNIQUE). Код для создания таблицы maintenance_types приведен в рисунке 2.19.

```
CREATE TABLE maintenance_types (
  id INTEGER,
  name VARCHAR(50) NOT NULL UNIQUE,
  PRIMARY KEY (id)
);
```

Рисунок 2.19 - Создание таблицы maintenance_types

Таблица milestone_statuses содержит атрибуты: id (первичный ключ) и name (название статуса вехи с ограничениями NOT NULL и UNIQUE). Код для создания таблицы milestone_statuses приведен в рисунке 2.20.

```
CREATE TABLE milestone_statuses (
  id INTEGER,
  name VARCHAR(50) NOT NULL UNIQUE,
  PRIMARY KEY (id)
);
```

Рисунок 2.20 - Создание таблицы milestone_statuses

Таблица milestones содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект), name (название вехи), description (описание), due_date (плановый срок), actual_complition_date (фактический срок завершения) и status_id (внешний ключ на статус вехи). Все атрибуты кроме actual_complition_date имеют ограничение NOT NULL. Код для создания таблицы milestones приведен в рисунке 2.21.

```
155 CREATE TABLE milestones (
156   id INTEGER,
157   project_id INTEGER NOT NULL,
158   name VARCHAR(50) NOT NULL,
159   description TEXT NOT NULL,
160   due_date DATE NOT NULL,
161   actual_complition_date DATE,
162   status_id INTEGER NOT NULL,
163   PRIMARY KEY (id),
164   FOREIGN KEY (project_id) REFERENCES projects(id),
165   FOREIGN KEY (status_id) REFERENCES milestone_statuses(id)
166 );
```

Рисунок 2.21 - Создание таблицы milestones

Таблица platforms содержит атрибуты: id (первичный ключ) и name (название платформы с ограничениями NOT NULL и UNIQUE). Код для создания таблицы platforms приведен в рисунке 2.22.

```
167 CREATE TABLE platforms (
168   id INTEGER,
169   name VARCHAR(50) NOT NULL UNIQUE,
170   PRIMARY KEY (id)
171 );
```

Рисунок 2.22 - Создание таблицы platforms

Таблица platforms_and_game_engines_for_projects содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект), game_engine_id (внешний ключ на игровой движок) и platform_id (внешний ключ на платформу). Все атрибуты имеют ограничение NOT NULL. Код для создания таблицы platforms_and_game_engines_for_projects приведен в рисунке 2.23.

```
172 CREATE TABLE platforms_and_game_engines_for_projects (
173   id INTEGER,
174   project_id INTEGER NOT NULL,
175   game_engine_id INTEGER NOT NULL,
176   platform_id INTEGER NOT NULL,
177   PRIMARY KEY (id),
178   FOREIGN KEY (project_id) REFERENCES projects(id),
179   FOREIGN KEY (game_engine_id) REFERENCES game_engines(id),
180   FOREIGN KEY (platform_id) REFERENCES platforms(id)
181 );
```

Рисунок 2.23 - Создание таблицы platforms_and_game_engines_for_projects

Таблица posts содержит атрибуты: id (первичный ключ), name (название должности) и salary (оклад). Все атрибуты имеют ограничение NOT NULL, а

name также имеет ограничение UNIQUE. Код для создания таблицы posts приведен в рисунке 2.24.

```
182 CREATE TABLE posts (
183   id INTEGER,
184   name VARCHAR(50) NOT NULL UNIQUE,
185   salary DECIMAL(10, 2) NOT NULL,
186   PRIMARY KEY (id)
187 );
```

Рисунок 2.24 - Создание таблицы posts

Таблица projects содержит атрибуты: id (первичный ключ), name (название проекта), project_start_date (дата начала), planned_end_date (плановая дата завершения), actual_end_date (фактическая дата завершения), version (версия), descripton (описание) и status_id (внешний ключ на статус проекта). Все атрибуты имеют ограничение NOT NULL, а name и version также имеют ограничение UNIQUE. Код для создания таблицы projects приведен в рисунке 2.25.

```
188 CREATE TABLE projects (
189   id INTEGER,
190   name VARCHAR(50) NOT NULL UNIQUE,
191   project_start_date DATE NOT NULL,
192   planned_end_date DATE NOT NULL,
193   actual_end_date DATE NOT NULL,
194   version VARCHAR(50) NOT NULL UNIQUE,
195   descripton TEXT NOT NULL,
196   status_id INTEGER NOT NULL,
197   PRIMARY KEY (id),
198   FOREIGN KEY (status_id) REFERENCES projects_statuses(id)
199 );
```

Рисунок 2.25 - Создание таблицы projects

Таблица projects_statuses содержит атрибуты: id (первичный ключ) и name (название статуса проекта с ограничениями NOT NULL и UNIQUE). Код для создания таблицы projects_statuses приведен в рисунке 2.26.

```
200 CREATE TABLE projects_statuses (
201   id INTEGER NOT NULL,
202   name VARCHAR(50) NOT NULL UNIQUE,
203   PRIMARY KEY (id)
204 );
```

Рисунок 2.26 - Создание таблицы projects_statuses

Таблица reports_statuses содержит атрибуты: id (первичный ключ) и name (название статуса отчета с ограничениями NOT NULL и UNIQUE). Код для создания таблицы reports_statuses приведен в рисунке 2.27.

```
205 CREATE TABLE reports_statuses (
206   id INTEGER,
207   name VARCHAR(50) NOT NULL UNIQUE,
208   PRIMARY KEY (id)
209 );
```

Рисунок 2.27 - Создание таблицы reports_statuses

Таблица roles содержит атрибуты: id (первичный ключ) и name (название роли с ограничениями NOT NULL и UNIQUE). Код для создания таблицы roles приведен в рисунке 2.28.

```
210 CREATE TABLE roles (
211   id INTEGER,
212   name VARCHAR(50) NOT NULL UNIQUE,
213   PRIMARY KEY (id)
214 );
```

Рисунок 2.28 - Создание таблицы roles

Таблица staff содержит атрибуты: id (первичный ключ), first_name (имя), last_name (фамилия), middle_name (отчество), birth_date (дата рождения), addres (адрес), post_id (внешний ключ на должность), hire_date (дата приема), email (электронная почта), phone (телефон) и gender (пол). Все атрибуты кроме middle_name имеют ограничение NOT NULL. Код для создания таблицы staff приведен в рисунке 2.29.

```
215 CREATE TABLE staff (
216   id INTEGER,
217   first_name VARCHAR(50) NOT NULL,
218   last_name VARCHAR(50) NOT NULL,
219   middle_name VARCHAR(50),
220   birth_date DATE NOT NULL,
221   addres VARCHAR(50) NOT NULL,
222   post_id INTEGER NOT NULL,
223   hire_date DATE NOT NULL,
224   email VARCHAR(50) NOT NULL,
225   phone VARCHAR(50) NOT NULL,
226   gender VARCHAR(1) NOT NULL,
227   PRIMARY KEY (id),
228   FOREIGN KEY (post_id) REFERENCES posts(id)
229 );
```

Рисунок 2.29 - Создание таблицы staff

Таблица staff_at_the_affiliated_studios содержит атрибуты: id (первичный ключ), affiliated_studio_id (внешний ключ на студию) и staff_id (внешний ключ на сотрудника). Оба внешних ключа имеют ограничение NOT NULL. Код для создания таблицы staff_at_the_affiliated_studios приведен в рисунке 2.30.

```
CREATE TABLE staff_at_the_affiliated_studios (
  id INTEGER,
  affiliated_studio_id INTEGER NOT NULL,
  staff_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (affiliated_studio_id) REFERENCES affiliated_studios(id),
  FOREIGN KEY (staff_id) REFERENCES staff(id)
);
```

Рисунок 2.30 - Создание таблицы staff_at_the_affiliated_studios

Таблица staff_on_projects содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект), staff_id (внешний ключ на сотрудника), role_id (внешний ключ на роль), start_date (дата начала), end_date (дата окончания) и hourly_rate (почасовая ставка). Атрибуты project_id, staff_id, role_id и start_date имеют ограничение NOT NULL, тогда как end_date и hourly_rate могут содержать NULL значения. Код для создания таблицы staff_on_projects приведен в рисунке 2.31.

```
CREATE TABLE staff_on_projects (
  id INTEGER,
  project_id INTEGER NOT NULL,
  staff_id INTEGER NOT NULL,
  role_id INTEGER NOT NULL,
  start_date DATE NOT NULL,
  end_date DATE,
  hourly_rate DECIMAL(10, 2),
  PRIMARY KEY (id),
  FOREIGN KEY (project_id) REFERENCES projects(id),
  FOREIGN KEY (staff_id) REFERENCES staff(id),
  FOREIGN KEY (role_id) REFERENCES roles(id)
);
```

Рисунок 2.31 - Создание таблицы staff_on_projects

Таблица suported_languages содержит атрибуты: id (первичный ключ), language_id (внешний ключ на язык) и project_id (внешний ключ на проект). Оба внешних ключа имеют ограничение NOT NULL. Код для создания таблицы suported_languages приведен в рисунке 2.32.

```sql
CREATE TABLE suported_languages (
  id INTEGER,
  language_id INTEGER NOT NULL,
  project_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (language_id) REFERENCES languages(id),
  FOREIGN KEY (project_id) REFERENCES projects(id)
);
```

Рисунок 2.32 - Создание таблицы suported_languages

Таблица task_statuses содержит атрибуты: id (первичный ключ) и name (название статуса с ограничениями NOT NULL и UNIQUE). Код для создания таблицы task_statuses приведен в рисунке 2.33.

```sql
CREATE TABLE task_statuses (
  id INTEGER,
  name VARCHAR(50) NOT NULL UNIQUE,
  PRIMARY KEY (id)
);
```

Рисунок 2.33 - Создание таблицы task_statuses

Таблица tasks содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект), name (название), description (описание), assignee_id (внешний ключ на исполнителя), status_id (внешний ключ на статус), priority (приоритет), created_date (дата создания), due_date (срок выполнения) и estimated_hours (оценка часов). Все атрибуты кроме id имеют ограничение NOT NULL для обеспечения полноты информации о задачах. Код для создания таблицы tasks приведен в рисунке 2.34.

```sql
CREATE TABLE tasks (
  id INTEGER,
  project_id INTEGER NOT NULL,
  name VARCHAR(50) NOT NULL,
  description TEXT NOT NULL,
  assignee_id INTEGER NOT NULL,
  status_id INTEGER NOT NULL,
  priority INTEGER NOT NULL,
  created_date DATE NOT NULL,
  due_date DATE NOT NULL,
  estimated_hours INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (project_id) REFERENCES projects(id),
  FOREIGN KEY (assignee_id) REFERENCES staff(id),
  FOREIGN KEY (status_id) REFERENCES task_statuses(id)
);
```

Рисунок 2.34 - Создание таблицы tasks

Таблица test_reports содержит атрибуты: id (первичный ключ), project_id (внешний ключ на проект), build_id (внешний ключ на сборку), tester_id (внешний ключ на тестировщика), test_date (дата тестирования), status_id (внешний ключ на статус отчета) и note (описание результатов). Все атрибуты имеют ограничение NOT NULL для обеспечения полноты документации по тестированию. Код для создания таблицы test_reports приведен в рисунке 2.35.

```sql
CREATE TABLE test_reports (
  id INTEGER,
  project_id INTEGER NOT NULL,
  build_id INTEGER NOT NULL,
  tester_id INTEGER NOT NULL,
  test_date DATE NOT NULL,
  status_id INTEGER NOT NULL,
  note TEXT NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (project_id) REFERENCES projects(id),
  FOREIGN KEY (build_id) REFERENCES builds(id),
  FOREIGN KEY (tester_id) REFERENCES staff(id),
  FOREIGN KEY (status_id) REFERENCES reports_statuses(id)
);
```

Рисунок 2.35 - Создание таблицы test_reports

После создания таблиц их нужно заполнить данными. Для этого используются INSERT запросы. Примиер данных запросов преведен в приложении к курсовой работе.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы была достигнута основная цель – разработана целостная и нормализованная реляционная модель базы данных для управления ключевыми процессами студии разработки видеоигр. Работа продемонстрировала, что централизованная система управления данными на основе реляционных принципов является эффективным решением проблем, характерных для игровой индустрии: разрозненности информации, сложности координации между отделами и отсутствия единого источника достоверных данных.

В рамках исследования были решены все поставленные задачи:

- Проведен анализ предметной области, который позволил выявить ключевые сущности, бизнес-процессы и функциональные требования к системе;
- На основе анализа спроектирована концептуальная модель, которая наглядно отображает основные сущности студии (проекты, сотрудники, оборудование, задачи, сборки и т.д.) и связи между ними;
- Концептуальная модель была преобразована в детализированную логическую схему базы данных, где для каждой сущности определены атрибуты, типы данных и ключи;
- Логическая схема была нормализована до соответствия требованиям третьей нормальной формы, что позволило минимизировать избыточность данных и исключить аномалии вставки, обновления и удаления;
- Завершающим этапом стала реализация физической модели базы данных с помощью SQL DDL-скриптов, готовых к развертыванию в реляционной СУБД.

Разработанная база данных охватывает все аспекты управления современной игровой студией: от учета человеческих ресурсов, оборудования и финансов до планирования проектов, контроля версий сборок и отслеживания процесса тестирования. Использование связующих таблиц

обеспечивает гибкость системы, позволяя моделировать сложные отношения «многие-ко-многим», такие как распределение сотрудников по проектам, назначение нескольких жанров игре или использование одного движка на разных платформах.

Практическая значимость работы заключается в том, что внедрение подобной системы позволит студии:

- Повысить прозрачность и управляемость всех бизнес-процессов;
- Автоматизировать рутинные операции учета и отчетности;
- Улучшить координацию между отделами за счет единого информационного пространства;
- Обеспечить менеджеров актуальными данными для принятия обоснованных решений;
- Эффективно масштабироваться при росте числа сотрудников, проектов и студий.

Таким образом, в рамках курсовой работы не только было реализовано конкретное техническое решение, но и доказана целесообразность применения реляционных баз данных как мощного инструмента для управления сложной и динамичной структурой игровой студии. Дальнейшим развитием проекта может стать создание пользовательского веб-интерфейса для работы с базой данных, реализация хранимых процедур для сложной бизнес-логики и более глубокая проработка модуля аналитики и прогнозирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 2.5 Физическое проектирование бд // StudFiles URL: https://studfile.net/preview/2152676/page:8 (дата обращения: 09.11.2025).
2. ACID Properties in DBMS // GeeksforGeeks URL: https://www.geeksforgeeks.org/dbms/acid-properties-in-dbms (дата обращения: 19.11.2025).
3. Chapter 13 Data Types // MySQL Documentation URL: https://dev.mysql.com/doc/refman/8.0/en/data-types.html (дата обращения: 12.11.2025).
4. Class Diagrams // UML Diagrams URL: https://www.uml-diagrams.org/class-diagrams.html (дата обращения: 05.11.2025).
5. Crow’s Foot Notation // Vertabelo URL: https://www.red-gate.com/blog/crow-s-foot-notation (дата обращения: 03.11.2025).
6. DBMS - Data Models // Tutorialspoint URL: https://www.tutorialspoint.com/dbms/dbms_data_models.htm (дата обращения: 19.11.2025).
7. draw.io Documentation // Draw.io URL: https://www.drawio.com/doc (дата обращения: 10.11.2025).
8. How to Design ER Diagrams for E-commerce Website // GeeksforGeeks URL: https://www.geeksforgeeks.org/dbms/how-to-design-er-diagrams-for-e-commerce-website (дата обращения: 19.11.2025).
9. Introduction of ER Model // GeeksforGeeks URL: https://www.geeksforgeeks.org/dbms/introduction-of-er-model (дата обращения: 12.10.2025).
10. Normal Forms in DBMS // GeeksforGeeks URL: https://www.geeksforgeeks.org/dbms/normal-forms-in-dbms (дата обращения: 18.11.2025).

11. Нормализация базы данных // DecoSystems URL: https://www.decosystems.ru/normalizatsiya-bazy-dannykh (дата обращения: 10.11.2025).

12. Нормализация отношений. Шесть нормальных форм // Habr URL: https://habr.com/ru/articles/254773 (дата обращения: 10.11.2025).

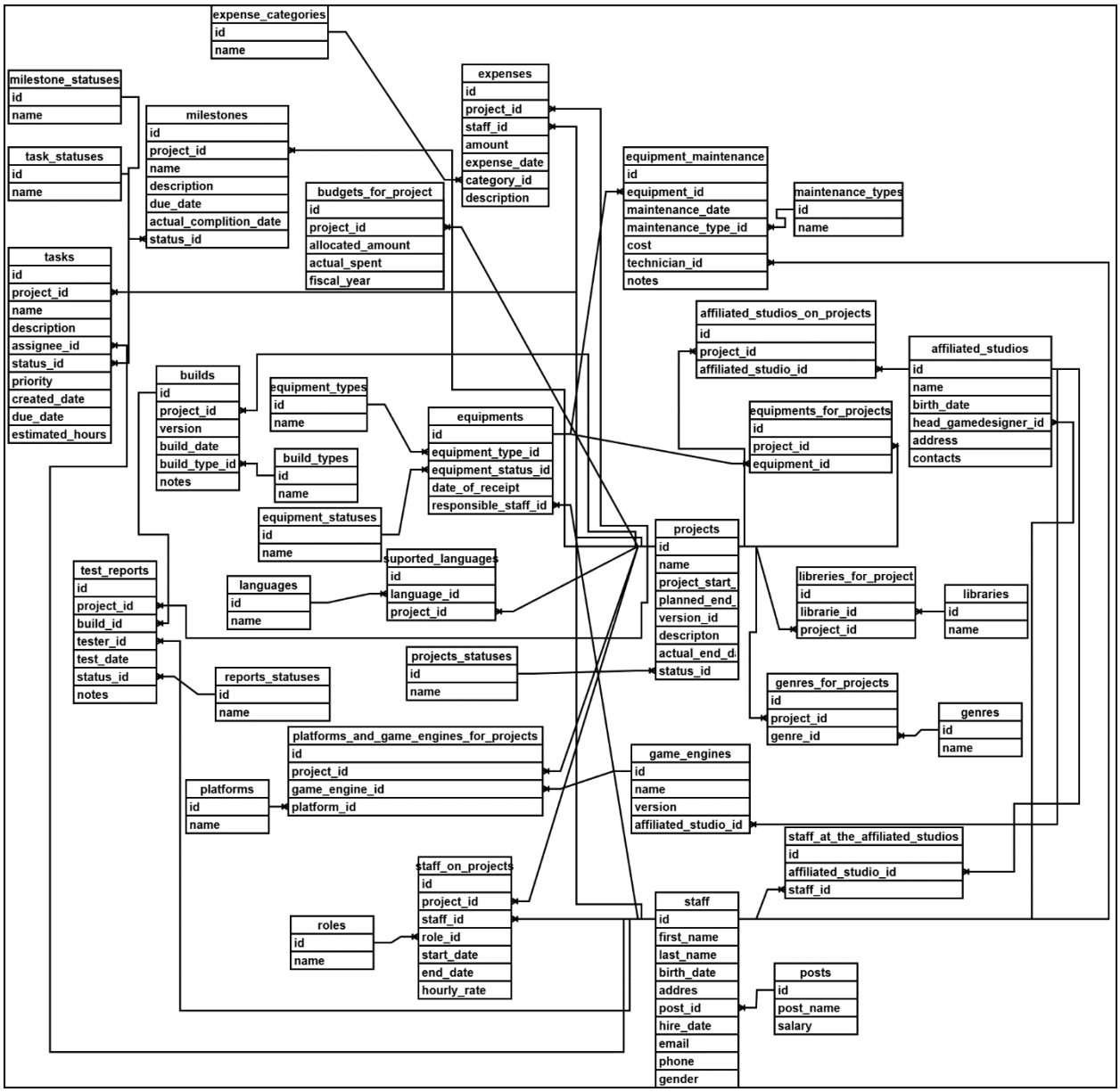
13. Нотации модели сущность-связь (ER диаграммы) // Блог программиста URL: https://pro-prof.com/archives/8126 (дата обращения: 09.11.2025).

14. Построение диаграммы классов // Flexberry URL: https://flexberry.github.io/ru/gpg_class-diagram.html (дата обращения: 09.11.2025).

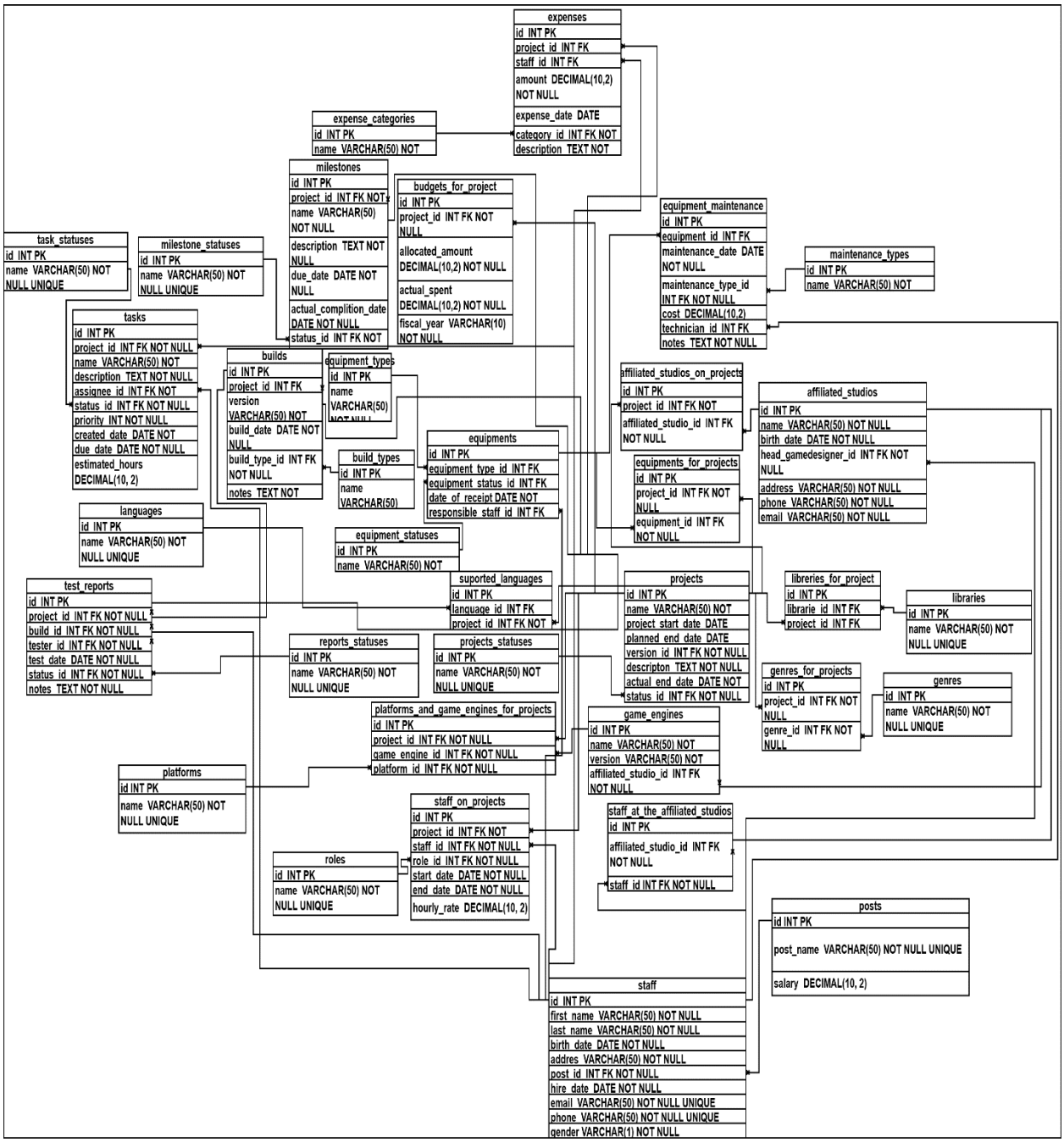
15. Семантическое моделирование. Проектирование БД с помощью ER-модели // Habr URL: https://habr.com/ru/companies/timeweb/articles/916824 (дата обращения: 08.11.2025).

16. Создание базы данных // Microsoft Docs URL: https://learn.microsoft.com/ru-ru/sql/relational-databases/databases/create-a-database?view=sql-server-ver16 (дата обращения: 11.10.2025).

ПРИЛОЖЕНИЕ 1



ПРИЛОЖЕНИЕ 2



ПРИЛОЖЕНИЕ 3

```sql
INSERT INTO posts (id, name, salary) VALUES
(1, 'Ведущий дизайнер', 75000.00),
(2, 'Старший разработчик', 90000.00),
(3, 'Тестировщик', 60000.00),
(4, 'Арт-директор', 80000.00),
(5, 'Системный администратор', 65000.00);

INSERT INTO projects_statuses (id, name) VALUES
(1, 'Планирование'),
(2, 'Активный'),
(3, 'Тестирование'),
(4, 'Завершен'),
(5, 'Приостановлен');

INSERT INTO milestone_statuses (id, name) VALUES
(1, 'Ожидает'),
(2, 'В работе'),
(3, 'Завершен'),
(4, 'Отложен'),
(5, 'Отменен');

INSERT INTO task_statuses (id, name) VALUES
(1, 'Открыта'),
(2, 'В работе'),
(3, 'Решена'),
(4, 'Закрыта'),
(5, 'Переоткрыта');
```