СОДЕРЖАНИЕ

ВВЕДЕНИЕ………………………………………………………………………………………. 5

1 ПОСТРОЕНИЕ ИНФОЛОГИЧЕСКОЙ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ………………. 6

1.1 Анализ предметной области и выявление необходимого набора сущностей………. 6

1.2 Обоснование требуемого набора атрибутов для каждой сущности и выделение идентифицирующих атрибутов……………………………………………………………... 7

1.3 Определение связей между объектами……………………………………………….. 10

1.4 Описание полученной модели на языке инфологического проектирования………... 11

2 ПОСТРОЕНИЕ СХЕМЫ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ…………………………... 12

2.1 Построение набора необходимых отношений базы данных…………………………. 12

2.2 Задание первичных и внешних ключей определенных отношений…………………. 13

2.3 Приведение отношения БД к третьей нормальной форме…………………………... 14

2.4 Определение ограничений целостности для внешних ключей отношений и для отношений в целом………………………………………………………………………….. 15

2.5 Графическое представление связей между внешними и первичными ключами…… 15

3 СОЗДАНИЕ СПРОЕКТИРОВАННОЙ БАЗЫ ДАННЫХ………………………………... 16

4 ЗАПИСЬ ВЫРАЖЕНИЙ, УКАЗАННЫХ В ВАРИАНТЕ ЗАДАНИЯ ТИПОВ ЗАПРОСОВ НА ЯЗЫКЕ SQL……………………………………………………………… 26

5 ВЫБОР И ОБОСНОВАНИЕ СРЕДСТВ РАЗРАБОТКИ ПРИЛОЖЕНИЯ……………… 36

6 РЕАЛИЗАЦИЯ ЗАКОНЧЕННОГО ПРИЛОЖЕНИЯ, РАБОТАЮЩЕГО С СОЗДАННОЙ БАЗОЙ ДАННЫХ…………………………………………………………………………... 37

6.1 Разработка и построение интерфейса главной и рабочей форм……………………... 37

6.2 Построение главного меню и кнопок панели инструментов………………………… 37

6.3 Выполнение программного кода в среде Microsoft Visual C#...................................... 38

ЗАКЛЮЧЕНИЕ………………………………………………………………………………… 39

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ………………………………………….. 40

ПРИЛОЖЕНИЕ А – КОНЦЕПТУАЛЬНАЯ СХЕМА БД…………………………………... .41

ПРИЛОЖЕНИЕ В – СХЕМА РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ………………………… 43

ПРИЛОЖЕНИЕ С – ГЛАВНАЯ И РАБОЧИЕ ФОРМЫ ПРИЛОЖЕНИЯ…………………. 44

ПРИЛОЖЕНИЕ D – ЛИСТИНГ ПРОГРАММЫ…………………………………………….. 46

ВВЕДЕНИЕ

Компьютерные игры в двадцать первом веке заняли место среди популярных средств получения досуга. Каждая новая игра позваляет получить своему пользователю какой-то новый опыт, которого ранее он не получал, при этом не создавая опасности для жизни. В 2018 году количество выпусков новых игр составило более сотни уникальных идей. С каждым годом количество выпускаемых игру увеличивается, а борьба за своего потенциального клиента увеличивается ещё быстрее. Уже достаточно давно существуют площадки для разработчиков, которые предоставляют услуги по размещению, хранению, обновлению и распространению компьютерных игр среди пользователей. Такие системы позволяют намного быстрее продвигать свой продукт, однако сегодня такой подход исчерпал себя из-за наплыва огроимного количества индивидуальных рпазработчиков и их низкокачественных проектов. К тому же отсутствие конкуренции на рынке площадок распространение компьютерных игр приводит только к негативным последствиям.

В 2018 году крупные игроки рынка компьютерных игр приняли решенние о переходи с централизованной площадки реализации игр на свои закрытые локальные площадки, в которых они смогут реализовывать только свою продукцию, забирая всю прибыль и внимание только себе. Для этих целей данным компаниям необходимо реализовать своё уникальное хранилище данных, в котором они будут хранить всю информацию о своих продуктах, а также о пользовательской активности в данной системе. В рамках курсовой работы будет выполнено проектирование такой системы, однако воприки тенденциям направленной на широкий круг разработчиков. Также будет реализовано графическое приложение для получения доступа в эту систему и управление ею.

Курсовая работа имеет своей целью:

− систематизацию знаний и накопление первоначального опыта в области проектирования баз данных и автоматизированных рабочих мест, закрепление, углубление и обобщение теоретических знаний по дисциплине «Базы данных» и применение их для решения научно-технических задач в соответствии с выбранной темой;

− формирование навыков самостоятельной работы и овладение методикой и технологией проектирования баз данных в составе корпоративных информационных систем;

− приобретение навыков обобщения и анализа результатов, полученных другими разработчиками по выбранной предметной области.

1 ПОСТРОЕНИЕ ИНФОЛОГИЧЕСКОЙ  
КОНЦЕПТУАЛЬНОЙ МОДЕЛИ

Инфологическая модель данных – описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных.

Цель инфологического моделирования – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическую модель данных пытаются строить по аналогии с естественным языком. Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).

1.1 Анализ предметной области и   
выявление необходимого набора сущностей

Чтобы правильно спроектировать базу данных, перед началом её проектирования необходимо определить основные задачи, которые будет решать вся система, после чего приступать к проектированию. Таким образом далее будет составлено краткое описание задачи.

**Задача системы** *–* построение автономной системы продажи и покупки компьютерных игр пользователям системы, предоставляя часть наибольшую заработанных средств разработчику.

**Цель существования системы** *–* накопление и продажа компьютерных игр пользователям.

**:**

* Учет учащихся учреждения образования.
* Систематизация накопленной информации.
* Составление документов и отчетов.
* Управление перемещением учащихся по структуре учебного заведения.
* Анализ накопленной информации.

**Основные бизнес-правила:**

* Информация об учащихся храниться до её явного удаления из системы.
* После составления управляющих документов, необходимо изменять соответствующие свойства учащихся на вновь определенные.
* Учащийся может иметь одну национальность и несколько гражданств.
* Каждый документ должен иметь уникальный номер, не повторяющийся в пределах одного типа содержания.
* У студента может быть один или несколько адресов, но только один из них может быть адресом регистрации.
* Академический отпуск может оформляться один раз по болезни и несколько раз по беременности и родам.

**Перечень печатных отчетов:**

* Приказ на отчисление за академическую неуспеваемость.
* Распоряжение о переводе в другую группу.
* Справка о том, что студент действительно обучается в ВУЗе по месту требования.

Для построения информационной системы требуется для начала выделить необходимы набор сущностей, которые описывают эту систему. Данный набор должен удовлетворять всем условиям на проектирование системы.

Определим минимальный набор сущностей, необходимый для проектирования информационной системы для учета контингента студентов университета. Для определения первичного набора сущностей будет проведён анализ технического задания и предметной области.

Для учета студентов в университете необходимо описание такой сущности как *студент*. В данной сущности должны быть заложены описательные характеристики студента, однозначно идентифицирующие его.

Все студенты обучаются в группах, поэтому следует выделить так же сущность – *группа*.

Студенты, организованные в группы, учатся на одном из факультетов, возглавляемом деканатом. Отсюда следует необходимость выделения сущности – *факультет*.

Структура факультета выглядит следующим образом: каждый факультет состоит из *кафедр*, за кафедрами числятся *специальности*. В свою очередь специальности имеют одну или несколько *специализаций*. На каждой специальности есть действующие *формы обучения* (одна или несколько). Следовательно, информация о данных структурах будет храниться в соответствующих сущностях.

Личные данные о студентах должны включать информацию о его *проживании*, *образовании*, *гражданстве и национальности,* *паспортные данные*, информацию о *работе* и *родственниках, семейном положении*. Всю данную информацию будем хранить в отдельных соответствующих сущностях.

Деканат оформляет на студентов различные документы, следовательно можно выделить сущность – *документ*. Каждый документ имеет свою *форму*, *назначение* и *содержание*, которые так же являются отдельными сущностями. К документу может прилагаться дополнительная информация, поэтому так же необходимо определить сущность «*Дополнительная информация к документу*».

1.2 Обоснование требуемого набора атрибутов для каждой сущности  
и выделение идентифицирующих атрибутов

Атрибут – поименованная характеристика сущности.

Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей.

Для каждой сущности, выделенной в пункте 1.1. необходимо определить атрибуты.

1. Сущность – «Игра»:

**Атрибуты:** название игры, дата выхода, стоимость, описание, скидка.

1. Сущность – «Разработчик»:

**Атрибуты:** название разработчика, описание, полученые средства.

1. Сущность – «Пользователь»:

**Атрибуты:** *логин, пароль, псевдоним, роль, статус.*

1. Сущность – «Покупка»:

**Атрибуты:** *пользователь, время, описание, сумма.*

1. Сущность – «Новость»:

**Атрибуты:** автор, время создания, время редактирования, заголовок, текст.

1. Сущность – «Комментарий»:

**Атрибуты:** *автор, время создания, время редактирования, текст.*

1. Сущность – «Дополнение к игре»:

**Атрибуты:** *название, описание, стоимость.*

1. Сущность – «Устройства»:

**Атрибуты:** *название устройства*

1. Сущность – «Тэги»:

**Атрибуты:** *название тэга.*

1. Сущность – «Жанры»:

**Атрибуты:** *название жанра.*

1. Сущность – «Статистика игры»:

**Атрибуты:** *год сбора статистики, количество покупок, количество просмотров, количество желающих купить игру, количество отметок «нравится», количество отметок «не нравится», количество комментариев.*

1. Сущность – «Кошелёк пользователя»:

**Атрибуты:** *счёт, кэшбэк, персональная скидка.*

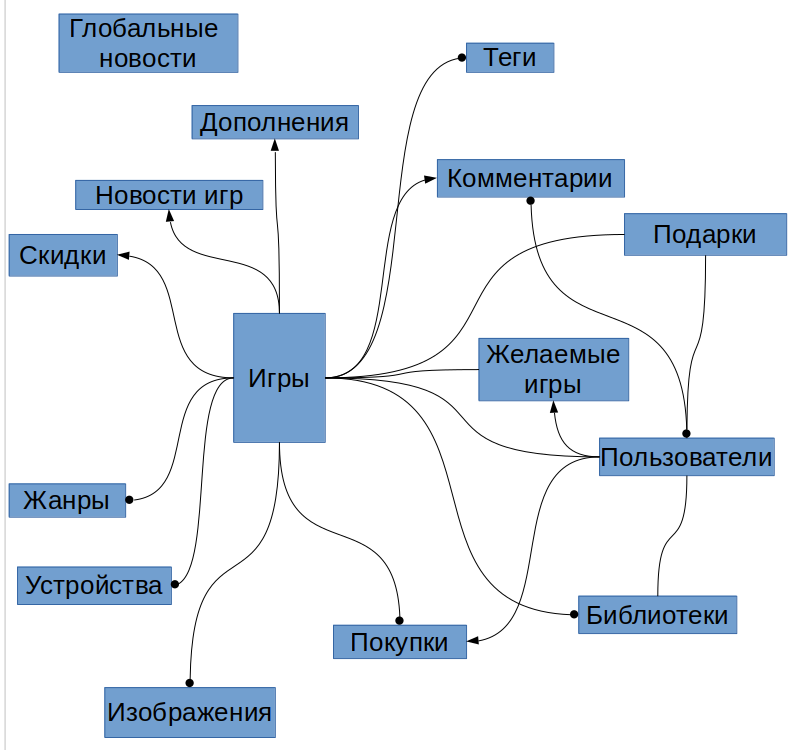
1.3 Определение связей между объектами

Следующим этапом в проектировании инфологической модели является установление связи между сущностями.

Связь – это ассоциирование двух или более сущностей. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой экземпляр данной сущности должен участвовать в данной связи).

Для реализации информационной системы компьютерных игр необходимо установить все связи между объектами. А именно, нужно рассмотреть всю информационную систему в совокупности и определить взаимное влияние объектов, составляющих систему.

Этот процесс изображен на рис. 1.1



**Рисунок 1.1 – Концептуальная схема базы данных**

Проследить отношения, в которых состоят таблицы базы данных можно по схеме, изображенной в приложении A на рис. А.1.

1.4 Описание полученной модели на языке  
инфологического проектирования

Проектирование инфологической модели предметной области – частично формализованное описание объектов предметной области в терминах некоторой семантической модели, например, в терминах ER-модели (*англ.* entity-relationship model).

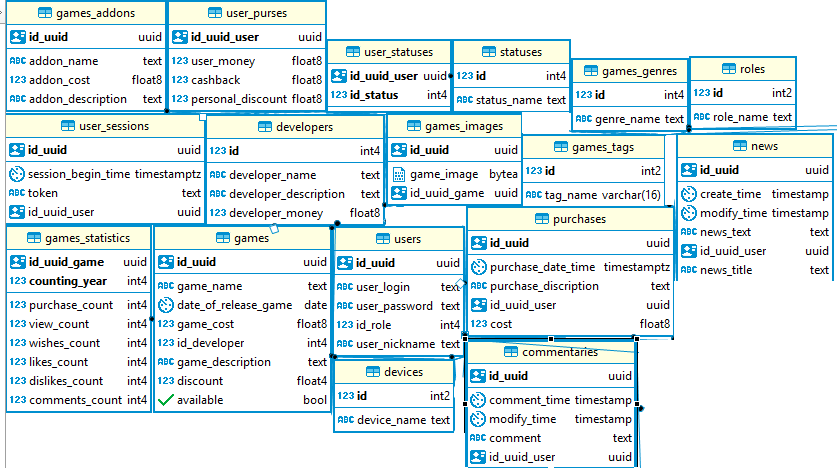
По правилам построения ER-диаграмм в нотации Crow's Foot(рус. «воронья лапка») сущность изображается в виде прямоугольника. Связь изображается линией, которая связывает две сущности, участвующие в отношении. Степень конца связи указывается графически, множественность связи изображается в виде «вилки» на конце связи. Модальность связи так же изображается графически — необязательность связи помечается кружком на конце связи. Атрибуты сущности записываются внутри прямоугольника, изображающего сущность.

2 ПОСТРОЕНИЕ СХЕМЫ РЕЛЯЦИОННОЙ  
БАЗЫ ДАННЫХ

2.1 Построение набора необходимых отношений базы данных

Для построения схемы реляционной базы данных необходимо определить совокупность отношений, составляющих базу данных. Эта совокупность отношений будет содержать всю информацию, которая должна храниться в базе данных.

На основе полученной в первом пункте концептуальной модели можно определить набор необходимых отношений базы данных. На рисунке 2.1 представлены отношения для базы данных информационной системы игровых приложений.



**Рисунок 2.1 – Набор необходимых отношений базы данных**

Схема реляционной базы данных представлена в приложении В.

2.2 Задание первичных и внешних ключей  
определенных отношений

В реляционной базе данных каждому объекту и сущности реального мира соответствуют кортежи отношений. И любое отношение должно обладать первичным ключом. Ключ – это минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся атрибутам. Каждое отношение должно обладать хотя бы одним ключом. В таблице 2.1 определены первичные и внешние ключи для отношений.[4]

**Таблица 2.1 – Первичные и внешние ключи отношений**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название таблицы | Первичный ключ | Внешние ключи |
|  | games | id\_uuid | id\_developer |
|  | users | id\_uuid | id\_role |
|  | games\_addons | id\_uuid |  |
|  | user\_purse | id\_uuid\_user | id\_uuid\_user |
|  | statuses | id |  |
|  | games\_genres | id |  |
|  | games\_tags | id |  |
|  | user\_sessions | id\_uuid | id\_uuid\_user |
|  | developers | id |  |
|  | roles | id |  |
|  | news | id\_uuid |  |
|  | games\_statistics | id\_uuid\_game  countin\_year | id\_uuid\_game |
|  | devices | id |  |
|  | commentaries | id\_uuid | id\_uuid\_user |
|  | purchases | id\_uuid | id\_uuid\_user |

2.3 Приведение отношения БД к третьей нормальной форме

Процесс преобразования базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры базы данных к виду, обеспечивающему минимальную избыточность, то есть нормализация не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение объёма БД. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в БД информации.[4]

Для реляционных баз данных необходимо, чтобы все отношения базы данных обязательно находились в 1НФ. Нормальные формы более высокого порядка могут использоваться разработчиками по своему усмотрению. Однако грамотный специалист стремится к тому, чтобы довести уровень нормализации базы данных хотя бы до 3НФ, тем самым, исключив из базы данных избыточность и аномалии обновления.

Определение 3НФ – неключевые атрибуты не должны определять другие неключевые атрибуты.

Для приведения базы данных в третью нормальную форму необходимо создать ряд агригирующих таблиц, которые будут выполнять роль хранения информации об отношениях сущностей. Большенство таких таблиц будут хранить пары вторичных ключей для установления связи между таблицами.

База данных в третьей нормальной форме представлена на схеме базы данных в приложение B.

2.4 Определение ограничений целостности для внешних  
ключей отношений и для отношений в целом

Ограничение целостности отношений заключается в том, что в любом отношении должны отсутствовать записи с одним и тем же значением первичного ключа. Конкретно требование состоит в том, что любая запись любого отношения должна быть отличной от любой другой записи этого отношения. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Ограничение целостности для внешних ключей состоит в том, что значение внешнего ключа должно быть равным значению первичного ключа цели; либо быть полностью неопределенным, т.е. каждое значение атрибута, участвующего во внешнем ключе должно быть неопределенным.

Условиями целостности называется набор правил, используемых для поддержания допустимых межтабличных связей и запрета на случайное изменение или удаление связанных данных. Следует устанавливать целостность данных только при выполнении следующих условий: связываемое поле из главной таблицы является полем первичного ключа и имеет уникальный индекс, связанные поля имеют один и тот же тип данных.

Для автоматического обновления связанных полей (удаления записей) при обновлении (удалении) в главной таблице, следует устанавливать обеспечение целостности данных и каскадное обновление связанных полей (каскадное удаление связанных записей).

Ограничение целостности, накладываемые на разрабатываемую систему:

* ключевое поле отношения должно быть уникальным;
* внешний ключ должен быть повторяющимся, то есть соответствовать уникальному ключу в своем отношении.

Для удовлетворения требования ограничения целостности для внешних ключей отношений и для отношений в целом необходимо, чтобы выполнялось соответствие между типами вводимых данных и типами столбцов в таблицах, а также чтобы были заполнены все обязательные поля в таблицах, т.е. те поля которые не могут содержать значения NULL.

2.5 Графическое представление связей между  
внешними и первичными ключами

По результатам нормализации, определении первичных и внешних ключей, связей между сущностями, была получена схема реляционной базы данных, представленная в приложении B. Полученная ER-диаграмма построена автоматически средством управления базами данных. На ней изображаются все отношения базы данных, а также связей между внешними и первичными ключами. Первичные ключи обозначаются жирным текстом в верхней части таблицы.

3 СОЗДАНИЕ СПРОЕКТИРОВАННОЙ БАЗЫ ДАННЫХ

Для реализации спроектированной базы данных была выбрана система управления базами данных PostgreSQL 10. Это обусловлено тем, что, данная СУБД имеет большую функциональность, она надёжна и имеет огромный запас производительности с отличной документацией.

В создаваемой базе данных будут использоваться следующие типы данных:

* integer – Целочисленный тип. Размер – 4 байта.
* int2 – Цлочисленный тип. Размер – 2 байта
* text – Строковый тип переменной длины.
* serial – Целочисленный тип размером в 4 байт с автоматической привязкой к персональному счётчику.
* boolean – Логический тип данных.
* date – Тип, определяющий дату.
* timestamp – Тип, определяющий дату и время.
* uuid – Кастомный тип данных для уникального идентифицирования– 16 байт.
* float – Число с плавающей точкой. Размер – 8 байт.
* float4 – Число с плавающей точкой. Размер – 4 байт.
* bytea – Двоичная строка переменной длины.

Опишем все таблицы, которые будут созданы в базе данных. Для удобства использования в базе данных все имена написаны в нижнем регистре, так как такое решение позволяет избавиться от потенциальных ошибок.

Таблица *games* содержит список всех игр, которые доступны и недоступны в магазине. Ее структура приведена в таблице 3.1.

**Таблица 3.1 – Характеристика атрибутов таблицы games**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid | uuid | Идентификатор игры, ключевое поле |
| game\_name | text | Название игры |
| game\_cost | float | Стоимость игры |
| date\_of\_release\_game | date | Дата релиза игры |
| id\_developer | integer | Идентификатор разработчика игры |
| game\_description | text | Описание игры |
| discount | float4 | Скидка на игру |
| available | boolean | Доступность игры пользователям |

Таблица *commentaries* содержит информацию о оставленном пользователем комментарии. Ее структура приведена в таблице 3.2.

**Таблица 3.2 – Характеристика атрибутов таблицы *commentaries***

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid | uuid | Идентификатор комментария, ключевое поле |
| id\_uuid\_user | uuid | Идентификатор пользователя |
| comment\_time | timestamp | Время добавление комментария |
| modify\_time | timestamp | Время редактирования комментария |
| comment | text | Текст комментария |

Таблица *desired\_games* хранит уникальные пары игра-пользователь и определяет, какие игры хочет преобрести пользователь. Ее структура приведена в таблице 3.3.

**Таблица 3.3 – Характеристика атрибутов таблицы desired\_games**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_game | uuid | Идентификатор желаемой игры |
| id\_uuid\_user | uuid | Идентификатор желающего пользователя |

Таблица *developer\_status\_purchases* хранит уникальные пары разработчик-покупка для хранения информации о всех покупках статусов разработчиков. Ее структура приведена в таблице 3.4.

**Таблица 3.4 – Характеристика атрибутов таблицы developer\_status\_purchases**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_developer | integer | Идентификатор разработчика |
| id\_uuid\_purchase | uuid | Идентификатор покупки |

Таблица *developers* содержит информацию о разработчиках. Ее структура приведена в таблице 3.5.

**Таблица 3.5 – Характеристика атрибутов таблицы developers**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | integer | Идентификатор разработчика, первичный ключ |
| developer\_name | text | Название разработчика |
| developer\_description | text | Описание разработчика |
| developer\_money | float | Денежное состояние разработчика |

Таблица *devices* хранит список устройств, на которых можно играть в игры. В отдельную таблицу были вынесены для возможного будущего расширения таблицы. Ее структура приведена в таблице 3.6.

**Таблица 3.6 – Характеристика атрибутов таблицы devices**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | int2 | Идентификатор устройства, перввичный ключ |
| device\_name | text | Наименование устройства |

Таблица *game\_addon\_purchases* связывает данные о покупке с купленным дополнением к игре. Ее структура приведена в таблице 3.7.

**Таблица 3.7 – Характеристика атрибутов таблицы game\_addon\_purchases**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_game\_addon | uuid | Идентификатор купленного дополнения к игре |
| id\_uuid\_purchase | uuid | Идентификатор покупки |

Таблица *game\_purchases* связывает данные о покупке с купленной игрой. Ее структура приведена в таблице 3.8.

**Таблица 3.8 – Характеристика атрибутов таблицы game\_purchases**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_game | uuid | Идентификатор купленной игры |
| id\_uuid\_purchase | uuid | Идентификатор покупки |

Таблица *games\_addons* хранит информацию о всех дополнениях к играм. Ее структура приведена в таблице 3.9.

**Таблица 3.9 – Характеристика атрибутов таблицы games\_addons**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid | uuid | Идентификатор дополнения, первичный ключ |
| id\_uuid\_game | uuid | Идентификатор игры, которой принадлежит дополнение |
| addon\_name | text | Название дополнения |
| addon\_cost | float | Идентификатор родственника |
| addon\_description | text | Степень родства |

Таблица *games\_commentaries* хранит данные о отношение комментариев к играм. Ее структура приведена в таблице 3.10.

**Таблица 3.10 – Характеристика атрибутов таблицы games\_commentaries**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_game | uuid | Идентификатор прокомментированной игры |
| id\_uuid\_comment | uuid | Идентификатор комментария |

Таблица *games\_genres* служит для хранения списка всех возможных жанров. Ее структура приведена в таблице 3.11.

**Таблица 3.11 – Характеристика атрибутов таблицы games\_genres**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | integer | Идентификатор жанра |
| genre\_name | text | Название жанра |

Таблица *games\_genres\_list* служит для связи игрового жанра с игрой. Ее структура приведена в таблице 3.12.

**Таблица 3.12 – Характеристика атрибутов таблицы games\_genres\_list**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_genre | integer | Идентификатор жанра |
| id\_uuid\_game | uuid | Идентификатор игры |

Таблица *games\_images* служит для хранения изображений, поставляемых с игрой. Эта таблица была создана отдельно для того, чтобы при выполнении запросов не нужно было загружать большой объем данных, который представляет собой изображение, а выполнять загрузку только по требованию. Ее структура приведена в таблице 3.13.

**Таблица 3.13 – Характеристика атрибутов таблицы games\_images**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid | uuid | Идентификатор изображения |
| id\_uuid\_game | uuid | Идентификатор игры |
| game\_image | bytea | Двоичные данные изображения |

Таблица *games\_news* служит для связи новости с конкретной игрой. Ее структура приведена в таблице 3.14.

**Таблица 3.14 – Характеристика атрибутов таблицы games\_news**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_games\_news | uuid | Идентификатор новости |
| id\_uuid\_game | uuid | Идентификатор игры, к которой относится новость |

Таблица *games\_ratings* хранит информацию об пользовательских отметках «нравится» или «не нравится». Ее структура приведена в таблице 3.15.

**Таблица 3.15 – Характеристика атрибутов таблицы games\_ratings**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_game | uuid | Идентификатор игры |
| id\_uuid\_user | uuid | Идентификатор пользователя |
| like\_game | boolean | Понравилась ли пользователю игра |

Таблица *games\_statistics* служит для хранения статистики игры за определённый год. Ее структура приведена в таблице 3.16.

**Таблица 3.16 – Характеристика атрибутов таблицы games\_statistics**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_game | uuid | Идентификатор игры, первичный ключ |
| counting\_year | integer | Год сбора статистики |
| purchase\_count | integer | Количество покупок игры |
| view\_count | integer | Количество просмотров игры |
| wishes\_count | integer | Количество пожеланий купить игру |
| likes\_count | integer | Количество отметок «нравится» |
| dislikes\_count | integer | Количество отметок «не нравится» |
| comments\_count | integer | Количество комментариев |

Таблица *games\_supported\_devices* связывает игры с устройствами, которые поддерживают эти игры. Ее структура приведена в таблице 3.17.

**Таблица 3.17 – Характеристика атрибутов таблицы games\_supported\_devices**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_game | uuid | Идентификатор игры |
| id\_device | integer | Идентификатор устройства |

Таблица *games\_tags* служит для хранения списка всех доступных тэгов. Ее структура приведена в таблице 3.18.

**Таблица 3.18 – Характеристика атрибутов таблицы games\_tags**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | integer | Идентификатор тэга, первичный ключ |
| tag\_name | text | Имя тэга |

Таблица *games\_tags\_list* связывает игры с их тэгами. Ее структура приведена в таблице 3.19.

**Таблица 3.19 – Характеристика атрибутов таблицы games\_tags\_list**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_game | uuid | Идентификатор игры |
| id\_tag | integer | Идентификатор тэга |

Таблица *global\_news* служит для хранения списка идентификаторов новостей, которые относятся к глобальным новостям. Ее структура приведена в таблице 3.20.

**Таблица 3.20 – Характеристика атрибутов таблицы global\_news**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_global\_news | uuid | Идентификатор глобальной новости, первичный ключ |

Таблица *global\_news\_commentaries* хранит идентификаторы комментариев, которые относятся к глобальной новости. Ее структура приведена в таблице 3.21.

**Таблица 3.21 – Характеристика атрибутов таблицы global\_news\_commentaries**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_comment | uuid | Идентификатор комментария к глобальной новости |
| id\_uuid\_global\_news | uuid | Идентификатор глобальной новости |

Таблица *news* служит для хранения всех когда-либо опубликованных новостей. Ее структура приведена в таблице 3.22.

**Таблица 3.22 – Характеристика атрибутов таблицы news**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid | uuid | Идентификатор новости, первичный ключ |
| id\_uuid\_user | uuid | Идентификатор автора новости |
| create\_time | timestamp | Время публикации новости |
| modify\_time | timestamp | Время редактирования новости |
| news\_title | text | Заголовок новости |
| news\_text | text | Текст новости |

Таблица *purchases* хранит информацию о всех пользовательских покупках. Ее структура приведена в таблице 3.23.

**Таблица 3.23 – Характеристика атрибутов таблицы purchases**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid | uuid | Идентификатор покупки, первичный ключ |
| id\_uuid\_user | uuid | Идентификатор пользователя |
| purchase\_date\_time | timestamp | Дата и время совершения покупки |
| cost | float | Стоимость покупки |
| purchase\_discription | text | Описание покупки |

Таблица *roles* хранит все доступные пользовательские роли. Важность роли увеличивается с каждым новым идентификатором, который используется для проверки на возможность выполнять конкретное действие. Ее структура приведена в таблице 3.24.

**Таблица 3.24 – Характеристика атрибутов таблицы roles**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | integer | Идентификатор роли, первичный ключ |
| role\_name | text | Название роли |

Таблица *statuses* определяет количество игр, которые должны быть куплены пользователем, чтобы заработать определённый статус. Ее структура приведена в таблице 3.25.

**Таблица 3.25 – Характеристика атрибутов таблицы statuses**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | integer | Идентификатор статуса, первичный ключ |
| status\_name | text | Название статуса |

Таблица *user\_game\_addons* хранит информацию об дополнениях к играм, которые купил пользователь. Ее структура приведена в таблице 3.26.

**Таблица 3.26 – Характеристика атрибутов таблицы user\_game\_addons**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_user | uuid | Идентификатор пользователя |
| id\_uuid\_game\_addon | uuid | Идентификатор дополнения к игре |

Таблица *user\_games* хранит информацию о купленных пользователем играх. Ее структура приведена в таблице 3.27.

**Таблица 3.27 – Характеристика атрибутов таблицы user\_games**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_user | uuid | Идентификатор пользователя |
| id\_uuid\_game | uuid | Идентификатор игры |

Таблица *user\_purses* хранит данные о средствах каждого пользователя. Ее структура приведена в таблице 3.28.

**Таблица 3.28 – Характеристика атрибутов таблицы user\_purses**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_user | uuid | Идентификатор пользователя, первичный ключ |
| user\_money | float | Счёт пользователя |
| cashback | float | Кэшюэк пользователя |
| personal\_discount | float | Персональная скидка пользователя |

Таблица *user\_sessions* хранит информацию о пользовательских сессиях. Ее структура приведена в таблице 3.29.

**Таблица 3.29 – Характеристика атрибутов таблицы user\_sessions**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid | uuid | Идентификатор пользовательской сессии, первичный ключ |
| id\_uuid\_user | uuid | Идентификатор пользователя |
| session\_begin\_time | timestamp | Дата и время начала сессии |
| token | text | Уникальный ключ доступа пользователя |

Таблица *user\_statuses* связывает пользователей с их текущими статусами. Ее структура приведена в таблице 3.30.

**Таблица 3.30 – Характеристика атрибутов таблицы user\_statuses**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_user | uuid | Идентификатор пользователя |
| id\_status | integer | Идентификатор статуса |

Таблица *users* служит для соединения форм и назначений документов, т.к. каждая форма документа может иметь свои определенные назначения. Ее структура приведена в таблице 3.31.

**Таблица 3.31 – Характеристика атрибутов таблицы users**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid | uuid | Идентификатор пользователя, первичный ключ |
| user\_login | text | Логин пользователя |
| user\_password | text | Пароль пользователя |
| user\_nickname | text | Псевдоним пользователя |
| id\_role | integer | Идентифкатор роли пользователя |

Таблица *users\_developers* хранит информацию о составе команды разработчиков с информацией о главенстве их пользователей. Ее структура приведена в таблице 3.32.

**Таблица 3.32 – Характеристика атрибутов таблицы users\_developers**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_user | uuid | Идентификатор пользователя |
| id\_developer | integer | Идентифкатор разработчика |
| main\_developer | boolean | Является ли пользователь главным разработчиком |

Таблица *games\_news\_commentaries* хранит данные о пренадлежности комментариев к игровым новостям. Ее структура приведена в таблице 3.33.

**Таблица 3.33 – Характеристика атрибутов таблицы games\_news\_commentaries**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id\_uuid\_comment | uuid | Идентификатор комментария |
| id\_uuid\_games\_news | uuid | Идентифкатор игровой новости |

Для поддержания целостности и защиты от прямого доступа к данным со стороны пользователя введён ряд хранимых процедур и триггеров, основная задача которых – инкапсуляция данных и предоставления интерфейса для работы с базой данных со стороны простого пользователя. Для удобства каждое имя процедуры начинается с «public\_function\_» или с «private\_function\_», чтобы наглядно разделить уровни доступа функций. Так как пользователь user не имеет прав на выполнение каких-либо запросов к таблицам, необходимо все интерфейсы публичных функций, составляющих интерфейс для работы с базой данных, дополнительно делегировать параметр «SECURITY DEFINER», который будет выполнять все функции не от имени вызывающего пользователя, а от имени автора функции.

Чтобы обеспечить разделение уровней доступа к данным, необходимо реализовать процедуру, которая будет проверять пользовательский токен на его правильность и владение определённым уровнем доступа. Возвращать эта функция должна булево значение, определяющее, имеет ли пользователь право на выполнение данной операции или нет. Данную функцию необходимо интегрировать во все функции, которые требуют какой-то определённый уровень доступа, начиная от простой авторизованности и заканчивая правами администратора.

Так же наобходимо реализовать триггеры, которые будут проверять корректность выполняемых операций и, в случае неудачи, сообщать пользователю о его ошибочных действиях. Для этой цели в PostgreSQL plpgsql существует функция raise, которая генерирует сообщение определённого уровня. Для сообщения об ошибки подойдёт уровень EXCEPTION.

4 ЗАПИСЬ ВЫРАЖЕНИЙ, УКАЗАННЫХ В ВАРИАНТЕ  
ЗАДАНИЯ ТИПОВ ЗАПРОСОВ НА ЯЗЫКЕ SQL

В задачу по проектированию базы данных так же входит написание определённых запросов, которые будут возвращать определённый набор данных. Список запросов и их реализация на языках SQL и PLPGSQL представлены в таблицах 4.1-4.9.

**Таблица 4.1 – задача и реализация первого запроса**

|  |
| --- |
| **Задача** |
| Вывести список комментариев к указанной игре с информацией об авторе |
| **Реализация** |
| **create** **or** **replace** **function** public\_function\_get\_cooments\_for\_game(  name\_of\_game **text**, commemts\_amount **integer**, list\_offset **integer**)  **returns** **table** (id\_uuid uuid, nickname **text**, “role” **text**, status **text**, add\_comment\_time **timestamp**, modify\_comment\_time **timestamp**, “comment” **text**)  **as** **$body$**  **select** commentaries.id\_uuid,  users.user\_nickname,  roles.role\_name,  statuses.status\_name,  commentaries.comment\_time,  commentaries.modify\_time,  commentaries.”comment”  **from** games  **inner** **join** games\_commentaries  **on** games\_commentaries.id\_uuid\_game = games.id\_uuid  **inner** **join** commentaries  **on** games\_commentaries.id\_uuid\_comment = commentaries.id\_uuid  **inner** **join** users  **on** commentaries.id\_uuid\_user = users.id\_uuid  **left** **join** user\_statuses  **on** users.id\_uuid = user\_statuses.id\_uuid\_user  **left** **join** statuses  **on** user\_statuses.id\_status = statuses.id  **inner** **join** roles  **on** users.id\_role = roles.id  **where** games.game\_name = name\_of\_game  **order** **by** commentaries.comment\_time **asc**  **limit** commemts\_amount  **offset** list\_offset;  **$body$**  **SECURITY** **DEFINER**  **language** **sql**; |
| **Тестирование** |
| select \* from public\_function\_get\_cooments\_for\_game(‘Minecraft’, 99, 0); |

**Таблица 4.2 – задача и реализация второго запроса**

|  |
| --- |
| **Задача** |
| Вывести сумарную стоимость стоимость покупок, связанных с игрой для определённого пользователя |
| **Реализация** |
| **create** **or** **replace** **function** public\_function\_get\_summary\_game\_cost\_with\_addons(user\_token **text**, name\_of\_game **text**)  **returns** **table** (game\_cost **float**, summary\_addons\_cost **float**)  **as** **$body$**  **declare**  user\_id uuid;  game\_id uuid;  **begin**  **if** ( private\_function\_check\_permissions(user\_token, 0) = **false** ) **then**  **raise** ‘Permission denied!’;  **end** **if**;  user\_id := private\_function\_get\_user\_id\_by\_token(user\_token);  **return** query  **select** games.game\_cost, **sum**(games\_addons.addon\_cost::**float**)  **as** summary\_addons\_cost  **from** games  **inner** **join** user\_games  **on** user\_games.id\_uuid\_game = games.id\_uuid  **left** **join** games\_addons  **on** games\_addons.id\_uuid\_game = games.id\_uuid  **left** **join** user\_game\_addons  **on** user\_game\_addons.id\_uuid\_game\_addon = games\_addons.id\_uuid  **where** games.game\_name = name\_of\_game **and**  user\_games.id\_uuid\_user = user\_id **and**  ( (games\_addons.id\_uuid **is** **not** **null** **and**  user\_game\_addons.id\_uuid\_game\_addon **is** **not** **null** **and**  user\_game\_addons.id\_uuid\_user = user\_id) **or**  (games\_addons.id\_uuid **is** **null** **and** user\_game\_addons.id\_uuid\_game\_addon **is** **null**) )  **group** **by** games.game\_cost;  **end**  **$body$**  **SECURITY** **DEFINER**  **language** plpgsql; |
| **Тестирование** |
| **select** \* **from** public\_function\_get\_summary\_game\_cost\_with\_addons(  ‘b9712af0060b44837757426afbba21b123332446’, ‘Terraria’)  **union**  **select** \* **from** public\_function\_get\_summary\_game\_cost\_with\_addons(  ‘b9712af0060b44837757426afbba21b123332446’, ‘Super test game’); |

**Таблица 4.3 – задача и реализация третьего запроса**

|  |
| --- |
| **Задача** |
| Вывести разработчиков, чьи игры имеют максимальное количество комментариев |
| **Реализация** |
| **create** **or** **replace** **function** public\_function\_get\_developer\_of\_most\_discussed\_game()  **returns** **table** (developer\_name **text**)  **as** **$body$**  **select** developers.developer\_name  **from** developers  **inner** **join** games  **on** games.id\_developer = developers.id  **inner** **join** games\_commentaries  **on** games\_commentaries.id\_uuid\_game = games.id\_uuid  **inner** **join** commentaries  **on** games\_commentaries.id\_uuid\_comment = commentaries.id\_uuid  **group** **by** developers.developer\_name  **order** **by** **count**(commentaries.id\_uuid) **desc**,  **max**(commentaries.comment\_time) **desc**  **limit** 5;  **$body$**  **SECURITY** **DEFINER**  **language** **sql**; |
| **Тестирование** |
| **select** \*  **from** public\_function\_get\_developer\_of\_most\_discussed\_game(); |

**Таблица 4.4 – задача и реализация четвёртого запроса**

|  |
| --- |
| **Задача** |
| Вывести все дополнения по указанной игре |
| **Реализация** |
| **create** **or** **replace** **function** public\_function\_get\_game\_addons(target\_game\_name **text**)  **returns** **table** (addon\_name **text**, addon\_cost **float**, addon\_description **text**)  **as** **$body$**  **select** games\_addons.addon\_name,  games\_addons.addon\_cost,  games\_addons.addon\_description  **from** games\_addons  **inner** **join** games  **on** games.id\_uuid = games\_addons.id\_uuid\_game  **where** games.game\_name = target\_game\_name;  **$body$**  **SECURITY** **DEFINER**  **language** **sql**; |

**Продолжение таблицы 4.4**

|  |
| --- |
| **Тестирование** |
| **select** \* **from** public\_function\_get\_game\_addons(‘Super test game’); |

**Таблица 4.5 – задача и реализация пятого запроса**

|  |
| --- |
| **Задача** |
| Вывести пользователя, который за указанный период потратил больше всего денег на покупку игр |
| **Реализация** |
| **create** **or** **replace** **function** public\_function\_get\_top\_spending\_money(  begin\_date **date**, end\_date **date**)  **returns** **table** (user\_nickname **text**)  **as** **$body$**  **select** users.user\_nickname  **from** users  **inner** **join** purchases  **on** users.id\_uuid = purchases.id\_uuid\_user  **where** purchases.purchase\_date\_time::**date** >= begin\_date **and**  purchases.purchase\_date\_time::**date** <= end\_date  **group** **by** users.user\_nickname  **order** **by** **sum**(purchases.”cost”::**float**) **desc**  **limit** 1;  **$body$**  **SECURITY** **DEFINER**  **language** **sql**; |
| **Тестирование** |
| **select** \* **from** public\_function\_get\_top\_spending\_money(‘2018-12-24’, ‘2018-12-26’); |

**Таблица 4.6 – задача и реализация шестого запроса**

|  |
| --- |
| **Задача** |
| Присвоить признак постоянного клиента пользователю, который пробрёл определённое количество игр |
| **Реализация** |
| **create** **or** **replace** **function** trigger\_function\_on\_buy\_new\_game() **returns** **trigger**  **as** **$body$**  **declare**  user\_games\_amount **integer**;  exists\_on\_statuses **oolean**;    **begin**  **select** **count**(user\_games.id\_uuid\_user) **into** user\_games\_amount  **from** user\_games  **where** user\_games.id\_uuid\_user = **new**.id\_uuid\_user  **group** **by** user\_games.id\_uuid\_user;    **select** **exists** (  **select** id **from** statuses **where** statuses.id = user\_games\_amount  ) **into** exists\_on\_statuses;    **if** ( exists\_on\_statuses ) **then**  **update** user\_statuses  **set** id\_status = user\_games\_amount  **where** id\_uuid\_user = **new**.id\_uuid\_user;  **end** **if**;    **return** **new**;  **end**  **$body$**  **language** plpgsql;  **create** **trigger** trigger\_on\_buy\_new\_game  **after** **insert** **on** user\_games **for** **each** **row**  **execute** **procedure** trigger\_function\_on\_buy\_new\_game(); |
| **Тестирование** |
|  |

**Таблица 4.7 – задача и реализация седьмого запроса**

|  |
| --- |
| **Задача** |
| Вывести список дополнений, которые принадлежат указанному пользователю |
| **Реализация** |
| **create** **or** **replace** **function** public\_function\_get\_user\_game\_addons(  user\_token **text**, target\_game\_name **text**)  **returns** **table** (addon\_name **text**, addon\_description **text**)  **as** **$body$**  **declare**  user\_id uuid;  game\_id uuid;    **begin**  **if** ( private\_function\_check\_permissions(user\_token, 0) = **false** ) **then**  **raise** ‘Permission denied!’;  **end** **if**;    user\_id := private\_function\_get\_user\_id\_by\_token(user\_token);  game\_id := private\_function\_get\_game\_id\_by\_name(target\_game\_name);    **return** query  **select** games\_addons.addon\_name,  games\_addons.addon\_description  **from** games\_addons  **inner** **join** user\_game\_addons  **on** games\_addons.id\_uuid = user\_game\_addons.id\_uuid\_game\_addon  **inner** **join** games  **on** games.id\_uuid = games\_addons.id\_uuid\_game  **where** user\_game\_addons.id\_uuid\_user = user\_id **and**  games.id\_uuid = game\_id;  **end**  **$body$**  **SECURITY** **DEFINER**  **language** plpgsql; |
| **Тестирование** |
| **select** \*  **from** public\_function\_get\_user\_game\_addons(  ‘b9712af0060b44837757426afbba21b123332446’, ‘Super test game’); |

**Таблица 4.8 – задача и реализация восьмого запроса**

|  |
| --- |
| **Задача** |
| Вывести список игр, которые ниразу небыли куплены пользователями |
| **Реализация** |
| **create** **or** **replace** **function** public\_function\_get\_games\_nobody\_bought(  game\_amount **integer**, list\_offset **integer**)  **returns** **table** (  game\_name **text**, game\_cost **float**, game\_discount **float4**,  date\_of\_release\_game **date**)  **as** **$body$**  **select** games.game\_name,  games.game\_cost,  games.discount,  games.date\_of\_release\_game  **from** games  **where** games.available **and**  games.id\_uuid **not** **in** (  **select** user\_games.id\_uuid\_game **from** user\_games  )  **limit** game\_amount  **offset** list\_offset;  **$body$**  **SECURITY** **DEFINER**  **language** **sql**; |
| **Тестирование** |
| **select** \* **from** public\_function\_get\_games\_nobody\_bought(99, 0); |

**Таблица 4.9 – задача и реализация девятого запроса**

|  |
| --- |
| **Задача** |
| Вывести список игр, которые хочет приобрести пользователь |
| **Реализация** |
| **create** **or** **replace** **function** public\_function\_get\_user\_desired\_games(nickname **text**)  **returns** **table** (desired\_game **text**)  **as** **$body$**  **select** games.game\_name  **from** games  **inner** **join** desired\_games  **on** games.id\_uuid = desired\_games.id\_uuid\_game  **inner** **join** users  **on** desired\_games.id\_uuid\_user = users.id\_uuid  **where** users.user\_nickname = nickname;  **$body$**  **SECURITY** **DEFINER**  **language** **sql**; |
| **Тестирование** |
| **select** \* **from** public\_function\_get\_user\_desired\_games('Test\_user'); |

5 ВЫБОР И ОБОСНОВАНИЕ СРЕДСТВ  
РАЗРАБОТКИ ПРИЛОЖЕНИЯ

Для реализации задачи по разработке информационной системы учета контингента студентов университета была выбрана СУБД MS SQL Server 2008. Данная СУБД является одной из наиболее популярных систем управления и обслуживанию баз данных. В ней применяются новейшие разработки и последние достижения в области проектирования, построения и обслуживания баз данных. Так же MS SQL Server стал де факто стандартом источника данных для всех решений Microsoft, что облегчает его интеграцию с различными программными пакетами.

Для создания приложения используется IDE Visual Studio 2010, а я зыком написания приложения был выбран – C#. Данное сочетание позволяет реализовать самые смелые идеи, т.к. предоставляет мощные средства для написания приложений.

Для соединения с базой данных было принято использовать ADO.NET Entity Framework – объектно-ориентированную технологию доступа к данным, являющуюся object-relational mapping (ORM) решением для .NET Framework от Microsoft.

В Entity Framework реляционные таблицы, столбцы и ограничения внешнего ключа логических моделей преобразуются в сущности и связи концептуальных моделей. Это позволяет достичь большей гибкости при определении объектов и оптимизации логической модели. С помощью инструментов модель EDM (сущностная модель данных) формируются расширяемые классы данных, основанные на концептуальной модели.

6 РЕАЛИЗАЦИЯ ЗАКОНЧЕННОГО ПРИЛОЖЕНИЯ,  
РАБОТАЮЩЕГО С СОЗДАННОЙ БАЗОЙ ДАННЫХ

6.1 Разработка и построение интерфейса главной и рабочей форм

Главная форма приложения является объектом класса *MainWindow*, наследуемый от класса Window, определенного в .NET Framework. Создание всех компонентов формы, в частности главного меню, управляющих элементов, окон сообщений, диалогов настроек и выбора файлов, панели с закладками и др., происходит в методе по мере их выхова, соответствующими им конструкторами.

Все основные таблицы для представления данных были выполнены в виде DataGrid, что упрощает понимание и просмотр информации, т.к. она представляется в табличном виде.

Все основные формы и виды приложения выполнены в одном окне на основе страничного интерфейса. Дочерние (вспомогательные) формы выполнены в виде диалоговых окон. Основной упор при проектировании интерфейса приложения был сделан на привлекательность и понятность для конечного пользователя. Поэтому были заменены многие стандартные элементы оформления на доработанные.

При проектировании приложения были учтены все возможные случаи некорректной работы программы, поэтому большинство нештатных ситуаций сопровождается оповещениями с описанием проблемы.

Скриншоты главной и некоторых диалоговых окон представлены в приложении C.

6.2 Построение главного меню и кнопок панели инструментов

Главное меню программы представлено пятью пунктами: СПИСКИ, ДОКУМЕНТЫ, УЧЕТ, НАСТРОЙКИ, О ПРОГРАММЕ. Данные пункты выполнены в виде одной панели, которая реагирует на нажатия пользователя.

Основные операции по работе с тем или иным окном приложения вынесены в нижнюю панель, где располагаются пиктограммы, представляющие собой подобие кнопок. Все они являются объектами класса *CircleIcon*.

Элементы выбора представлены объектами *ComboBox* и *TreeView*.

6.3 Выполнение программного кода в среде Microsoft Visual C#

Опишем работу приложения с базой данных. Все необходимые интерфейсы для работы с базами данных находятся в классе *DatabaseAgent*. Подключение к базе данных начинается с формирования строки подключения и последующим созданием контекста на основе данной строки:

string conn = GetConnection(); [// получение](file:///\\получение) строки подключения путем построения объекта EntityConnectionStringBuilder и дальнейшем преобразовании его в строку

\_database = new ISDatabaseEntities(conn); // создание контекста модели, для последующей работы с базой данных и её сущностями

После подключения для осуществления запросов необходимо проверить достоверность базы данных, если проверка проходит успешно, то будут разблокированы возможности по работе с базой. Для этого используются методы: *DatabaseExist*() и событие *ConnectAction*.

Все запросы по работе с базой данных обращаются к контексту и строятся с помощью LINQ to Entity [2]. В следствии чего упрощается построение запросов, вызов процедур и фильтрация результатов. Пример текста запроса, возвращающего список студентов, со статусом «отчислен» выглядит следующим образом:

DatabaseAgent.Instance.Database.Students

.Where(s=>s.Status.Equals(“отчислен”);

Результат выполненного запроса представляет собой последовательность IQueryable<T>, которую можно назначить в качестве источника данных для таблиц, и прочих элементов управления.

Функционал по работе с отчетами реализован в классе *DocumentsManager*, в котором представлены методы для создания трех видов отчета и печати документов.

Управление информационными сообщениями и уведомлениями происходит в классе MessagesManager, который содержит все основные тексты уведомления и методы по выводу информационных сообщений и сообщений об ошибках.

Все перечисления и неизменяемые наборы данных расположены в классе *Enums*.

Хранение и обработку настроек приложения обеспечивает класс *AppSettings*, который позволяет загржать, сохранять и изменять настройки программы.

ЗАКЛЮЧЕНИЕ

В результате выполненной работы, была создана база данных для учета контингента студентов университета, а также приложение, а так же эффективно работающее с этой базой данных приложение. Полученная комбинация представляет собой информационную систему ВУЗа.

Разработанная база данных удовлетворяет всем требованиям, предъявленным в задании, и позволяет без проблем хранить и извлекать требуемую информацию.

Созданное приложение позволяет упростить работу с информацией для работников университета, позволяя систематизировать всю необходимую информацию. Также приложение упрощает операции по удалению, изменению, добавлению данных. В приложении реализованы запросы, позволяющие пользователю выбрать всю необходимую информацию по заданным критериям, осуществлять поиск данных и формировать отчеты.

Разработанная система реагирует на ошибочный ввод данных, а также способна определять возникающие ошибки и уведомлять об этом пользователя, чтобы в любой момент он знал из-за чего или почему произошла ошибка, и оперативно устранил её.

В процессе выполнения данной курсовой работы были закреплены навыки в программировании на языке C#, проектировании баз данных и реализации их в СУБД MS SQL Server 2008.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Шилдт Г. Полный справочник по SQL.: Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2004. – 752 c.: ил.
  2. Раттц Д. LINQ язык интегрированных запросов в C# 2008 для профессионалов. Пер. с англ. – М.: Вильямс, 2008. – 645с.: ил.
  3. Хернандес М., Вьескас Д. SQL-запросы. Практическое руководство.: Пер. с англ. – М.: Лори, 2003. – 473 c.: ил.
  4. Коннолли Т., Бегг К., Базы данных. Проектирование, реализация и сопровождение. Теория и практика.: Пер. с англ. – М.: Вильямс, 2003. – 1500 с.: ил.
  5. Jennings R., Professional ADO.NET 3.5 with LINQ and the Entity Framework.: – New York.: Wrox, 2009. – 560 c.: ил.

ПРИЛОЖЕНИЕ A

(обязательное)

КОНЦЕПТУАЛЬНАЯ СХЕМА БД

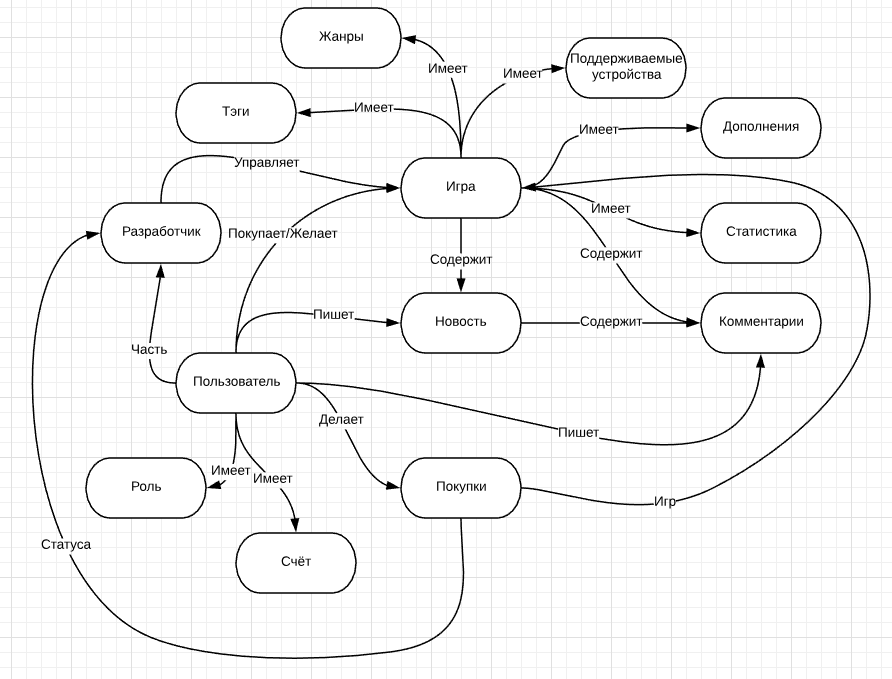


Рисунок A.1 – Инфологическая схема проектируемой базы данных

# 

ПРИЛОЖЕНИЕ B

(обязательное)

СХЕМА РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

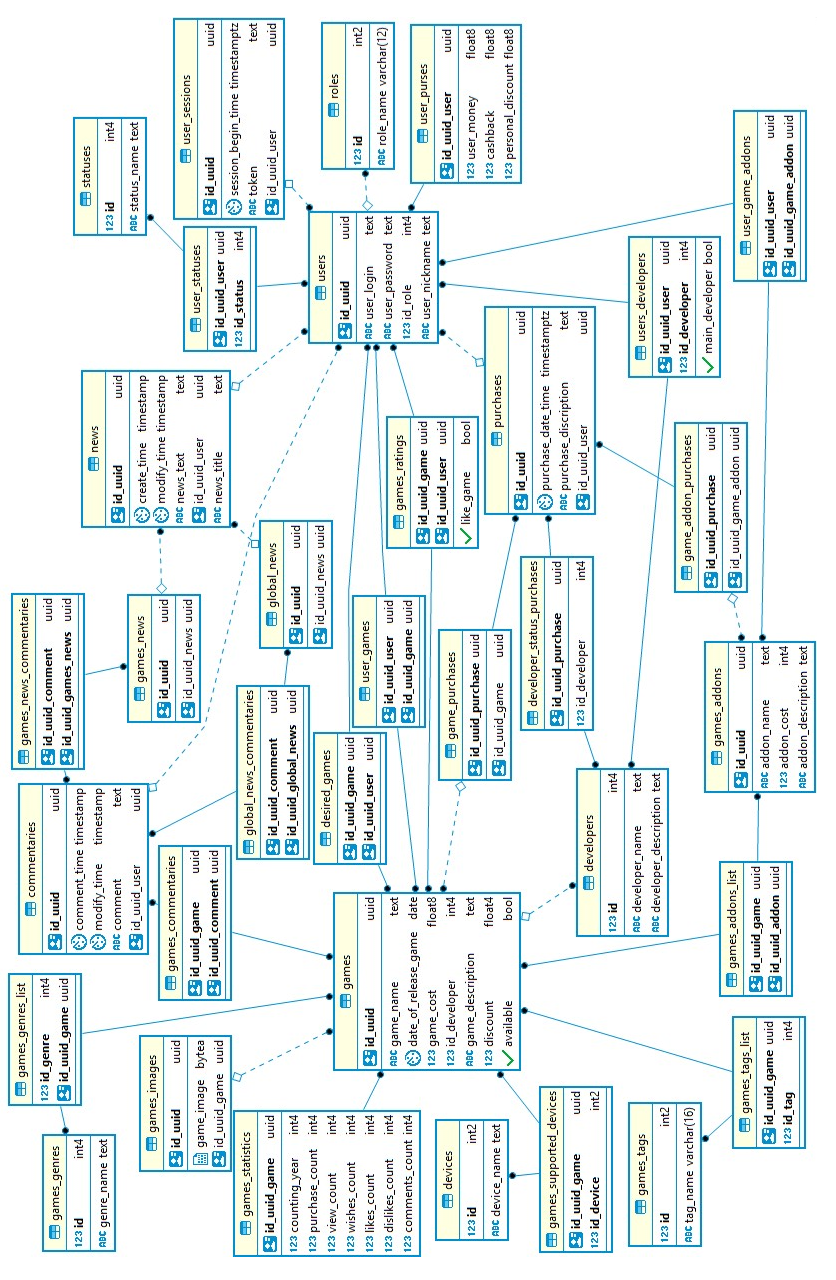


Рисунок B.1 – Схема реляционной базы данныхПРИЛОЖЕНИЕ C

(обязательное)

ГЛАВНАЯ И РАБОЧИЕ ФОРМЫ ПРИЛОЖЕНИЯ

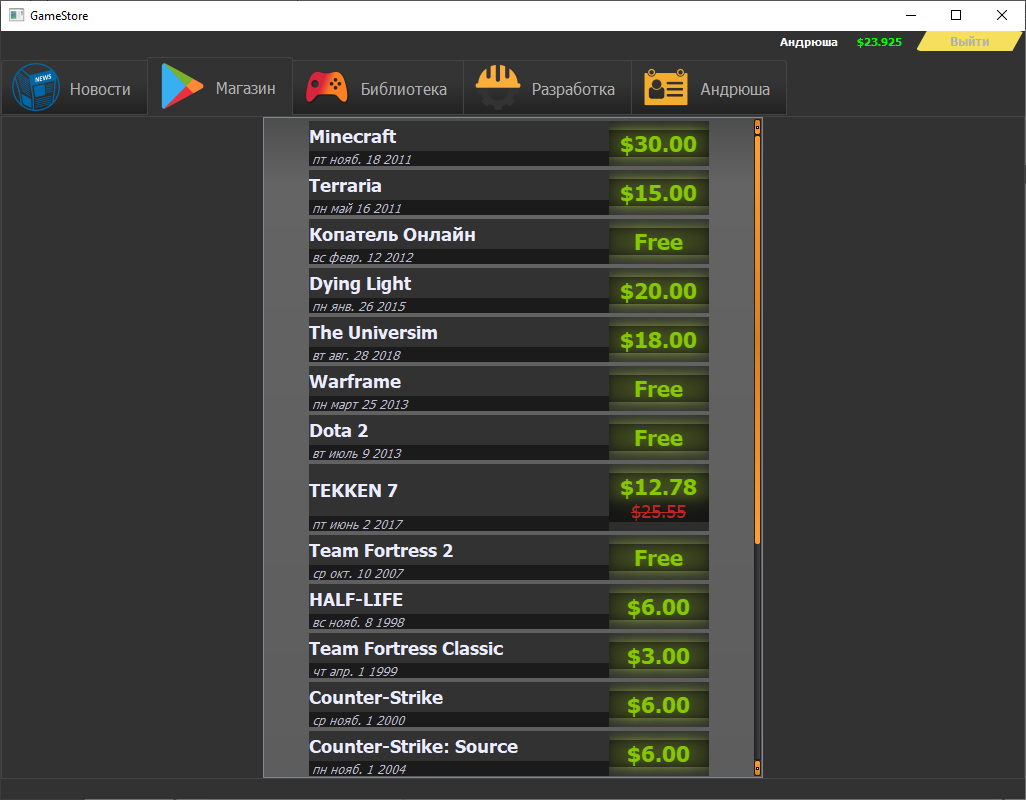


Рисунок C.1 – Главная форма приложения

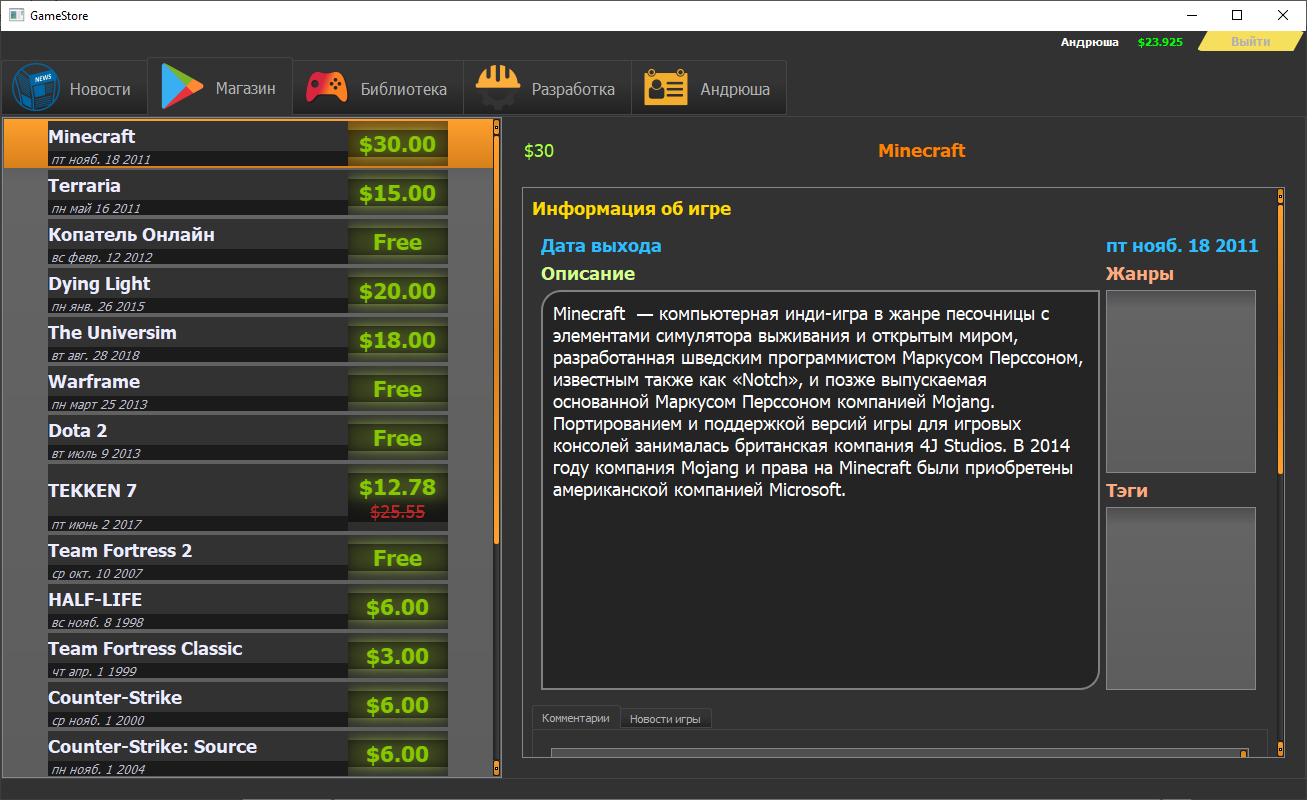


Рисунок C.2 – Просмотр информации об игре

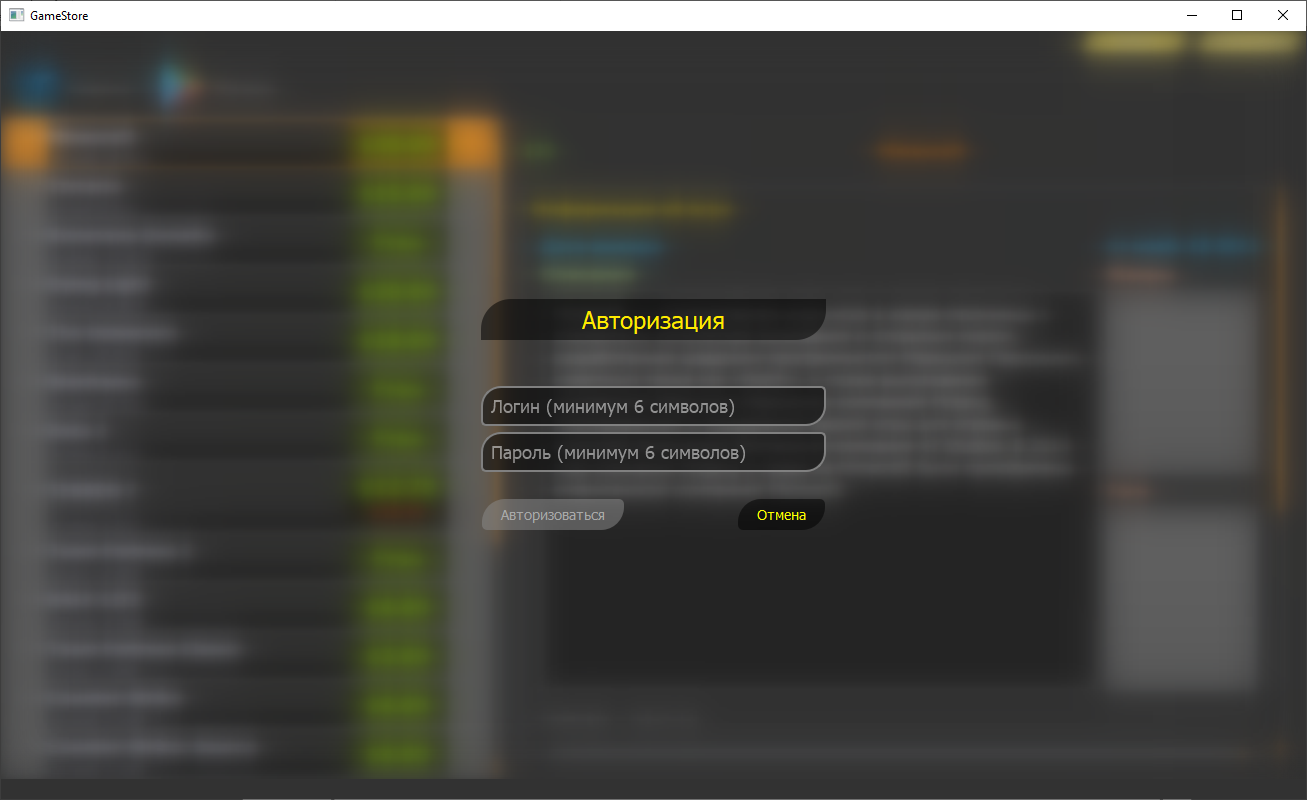


Рисунок C.3 – Форма авторизации пользователя

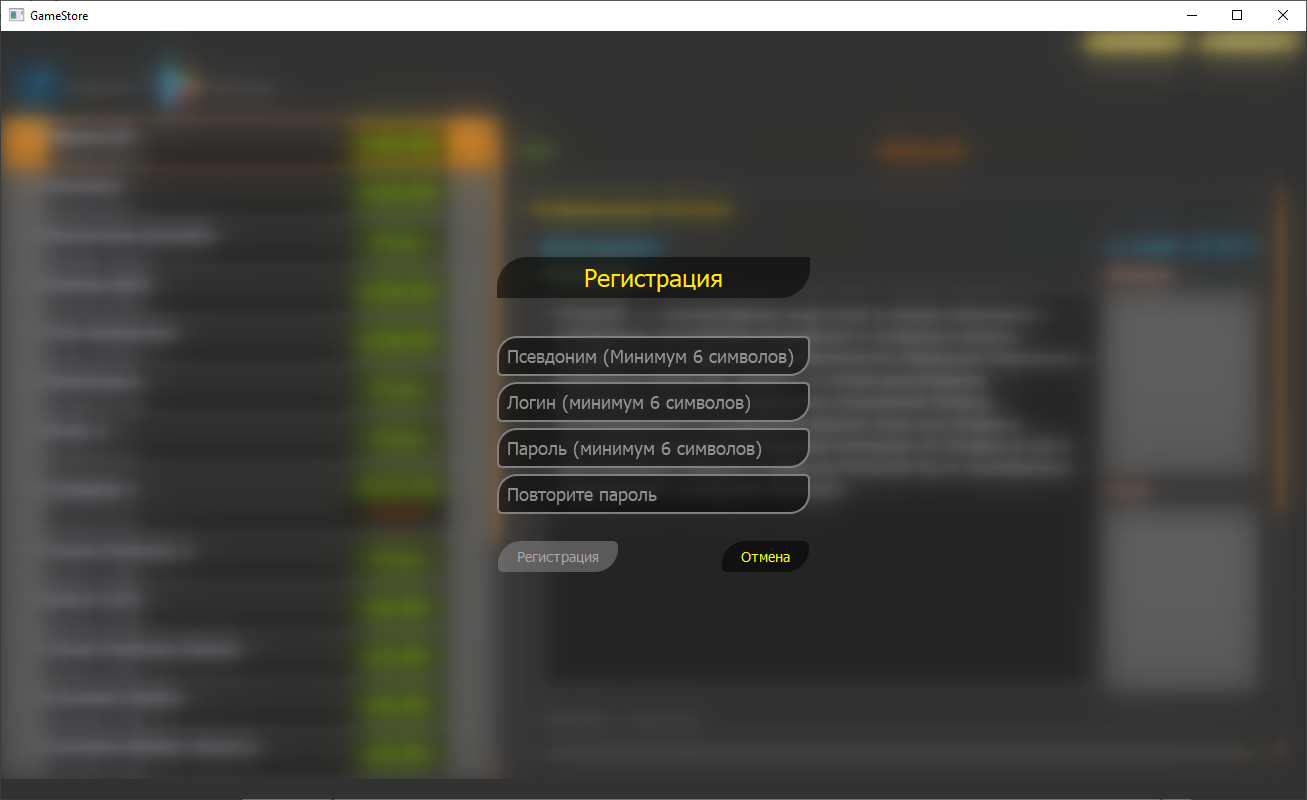


Рисунок C.4 – Форма регистрации пользователя

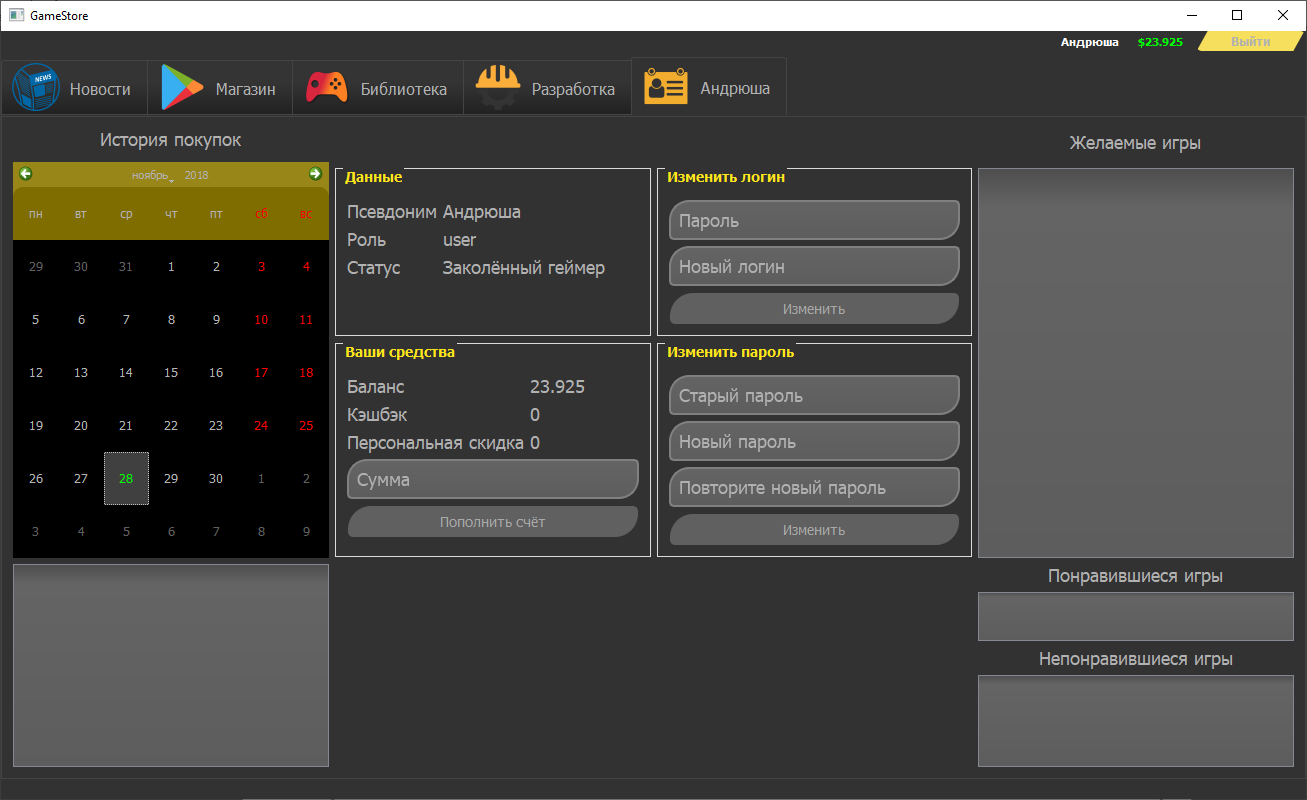


Рисунок C.5 – Управление аккаунтом пользователя

ПРИЛОЖЕНИЕ D

(обязательное)

ЛИСТИНГ ПРОГРАММЫ

**Исходный код класса DatabaseAgent**