Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Дисциплина: Базы и банки данных

|  |
| --- |
|  |
|  |
|  |

Пояснительная записка

к курсовому проекту

на тему

**Автоматизированная система прогнозирования спроса**

БГУИР КП 1-53 01 02 01 10 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент группы 920603 |  | В. А. Кособуцкий |
| Руководитель |  | А. Ф. Трофимович |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск 2022

**РЕФЕРАТ**

Автоматизированная система прогнозирования спроса: курсовой проект / В. А. Кособуцкий. – Минск : БГУИР, 2022, – п.з. – 51 с.

Курсовой проект на тему «Автоматизированная система прогнозирования спроса» разработан с целью проектирования и реализации распределённой системы для ритейлинговой сети, решающей задачу прогнозирования потребительского спроса на определённые товары в определённой точке сети.

Пояснительная записка к курсовому проекту состоит из введения, четырёх разделов, заключения и списка использованных источников.

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

|  |  |
| --- | --- |
| Факультет ИТиУ––––––------ | Кафедра ИТАС–– ––– -- |
| Специальность 1-53 01 02— — | Специализация 01– –––––----––– |

УТВЕРЖДАЮ

Зав. кафедрой

*–––– –––––––––––––––*

(подпись)

––––––––– –––––2022г.

ЗАДАНИЕ

по курсовому проектированию студента

*—————— ——*Кособуцкого Владислава Александровича*— —— ——*

1 Тема проекта:  Автоматизированная система прогнозирования спроса *—*

2 Срок сдачи студентом законченной работы–*13 мая 2022 г.* —*——* —

3 Исходные данные к проекту: *Необходимо разработать распределённое приложение со следующими узлами: — —*

* *сервер представляет собой базу данных ритейлинговой сети, реализовать RESTful веб-API для всех необходимых CRUD операций;— —*
* *узел прогнозирования — оконное приложение, предназначенное для экспертов в области прогнозирования, включающее в себя редактор записей, функцию визуального контроля качества прогнозирования, RESTful API, при обращении к которому возвращаются результаты прогнозирования;— —*
* *клиент — оконное приложение, предназначенное для руководящего персонала магазинов, включающее в себя функцию создания отчёта (плана закупок конкретных наименований товаров в конкретный магазин сети) в формате XML, генерацию yml-файла (Yandex Market Language) для загрузки товаров в систему «мультимаркет». — —*

*Схема базы данных должна быть спроектирована с учётом особенностей набора данных, взятого из открытого доступа (лицензия CC0) на платформе Kaggle, и знаний предметной области. — —*

*Модель прогнозирования должна представлять собой адаптивную композицию алгоритмов с несимметричной функцией потерь. Данная функция потерь должна отображать действительные финансовые потери сети. Должна быть возможность задания параметров функции потерь экспертами в области. В условии отсутствия экспертов параметры инициализировать случайными значениями. — —*

4 Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):

Введение ———— ————— — —— —

1 Обзор исследуемой области ———— ————— — —— —

2 Проектирование системы— с ————— — —

3 Программная реализация системы с ——— ——

4 Руководство пользователя с ———

Заключение———————————————————————— —

5 Дата выдачи задания *30 января 2022 г. –––––––––––––* –*–––*

6 Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*разделы 1-2 к 27.02 – 30 %; ––––––– –– –––––– –––––*

*раздел 3 к 26.03 – 60 %; – –––––– –– –––––– –––––*

*раздел 4 к 23.04 – 90 %; ––––––– ––––––– – –––––*

*оформление пояснительной записки к 04.05 – 100 %; –––*

*Защита курсового проекта с* ***04.05*** *по* ***18.05.2022*** *г.   – ––––––––––– –––*

Руководитель*– ––––– А. Ф. Трофимович*

(подпись)

Задание принял к исполнению *–––––––\_\_\_\_\_\_\_\_ \_ – В. А. Кособуцкий*

(дата и подпись студента)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc103339837)

[1 ОБЗОР ИССЛЕДУЕМОЙ ОБЛАСТИ 7](#_Toc103339838)

[1.1 Описание исследуемого объекта 7](#_Toc103339839)

[1.2 Обзор аналогов 7](#_Toc103339840)

[1.3 Техническое задание 9](#_Toc103339841)

[1.4 Технические требования 9](#_Toc103339842)

[2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ 10](#_Toc103339843)

[2.1 Проектирование бизнес-логики 10](#_Toc103339844)

[2.2 Проектирование базы данных 11](#_Toc103339845)

[2.3 Проектирование модели прогнозирования 14](#_Toc103339846)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ 21](#_Toc103339847)

[3.1 Выбор средств реализации 21](#_Toc103339848)

[3.2 Реализация базы данных 21](#_Toc103339849)

[3.3 Реализация потока нормализации 22](#_Toc103339850)

[3.4 Реализация сервера базы данных 23](#_Toc103339851)

[3.5 Реализация узла прогнозирования 30](#_Toc103339852)

[3.6 Реализация клиента 38](#_Toc103339853)

[4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 40](#_Toc103339854)

[4.1 Руководство по серверу базы данных 40](#_Toc103339855)

[4.2 Руководство по узлу прогнозирования 41](#_Toc103339856)

[4.3 Руководство клиента 45](#_Toc103339857)

[ЗАКЛЮЧЕНИЕ 49](#_Toc103339858)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 50](#_Toc103339859)

[ВЕДОМОСТЬ КУРСОВОГО ПРОЕКТА 51](#_Toc103339860)

# ВВЕДЕНИЕ

В данной работе рассматривается задача прогнозирования потребительского спроса. Она заключается в том, чтобы предсказать объем продаж заданного товара в заданном магазине на заданный интервал времени вперёд. Конечной целью прогнозирования является планирование закупок.

Решения о сроках и объектах закупок принимаются на основании не только прогнозов потребительского спроса, но и календаря поставок, возможностей поставщика, остатка товара, выделяемого бюджета на заказ товара, максимальной загрузке склада данным товаром. Однако в данной работе рассматривается только прогнозирование спроса, как наиболее наукоёмкая задача, решаемая в рамках планирования закупок.

Задачи прогнозирования потребительского спроса обладают специфическими особенностями. При прогнозировании необходимо учитывать ряд следующих факторов: сезонность, тренд, праздники, рекламные акции, временное отсутствие товара на складе, неслучайное отсутствие спроса на товар, взаимное влияние продаж различных товаров, планы продаж, параметры выкладки товаров [1].

Темой курсового проекта является автоматизированная система прогнозирования спроса.

Целью данного курсового проекта является построение распределённой системы прогнозирования потребительского спроса, используя адаптивную композицию методов прогнозирования временных рядов.

Задачами курсового проекта являются:

* проектирование базы данных;
* проектирование модели прогнозирования;
* проектирование распределённого приложения;
* реализация базы данных;
* реализация модели прогнозирования;
* реализация распределённого приложения;
* создание документации для пользователей приложения.

# ОБЗОР ИССЛЕДУЕМОЙ ОБЛАСТИ

## 1.1 Описание исследуемого объекта

Данная система разработана с целью оптимизации закупок ритейлинговых сетей. Недопрогноз спроса на товар приводит к упущенной прибыли и, в более серьёзных случаях, потере лояльности клиентов. Перепрогноз спроса увеличивает издержки на хранение, замораживает средства в нереализованном товаре и, по истечению срока годности, приводит к потерям в размере всей закупочной стоимости товара.

Ручное прогнозирование невозможно в необходимых масштабах: сеть магазинов может состоять из десятков точек, в каждой из которых продаются сотни товаров. Это довольно сдержанная оценка, но уже достаточная для аргументации позиции.

## 1.2 Обзор аналогов

**1.2.1 Модели прогнозирования**

Существующие модели прогнозирования временных рядов включают в себя авторегрессию (скользящее среднее, как частный случай), семейство *ARIMA*, семейство *GARCH*, нейросетевые методы и методы множественной линейной регрессии.

Большинство из них реализовано для решения задачи при симметричной (квадратичной) функции потерь, что не позволяет интерпретировать эти потери в денежном эквиваленте.

Нейросетевые методы вычислительно сложны, а методы семейств *GARCH* и *ARIMA* требуют ручной настройки, что не позволяет их использовать в поставленной задаче.

Возможно использование ансамбля моделей при помощи методов адаптивной селекции или адаптивной композиции. Реализация данных методов является задачей конкретного разработчика за неимением *open-source* реализаций, совместимых с часто используемым набором библиотек языка *Python*, широко используемого для задач машинного обучения.

**1.2.2 Программные продукты**

***GMDH Streamline***

*GMDH Streamline* — это платформа планирования цепочки поставок с прогнозированием продаж и планированием поставок. Имеет бесплатную версию с ограниченным функционалом. Двунаправленно интегрируется с *Excel*, 1C, *SAP Business One*, *Oracle Net Suite*, *Shopify* [2]. Внешний вид представлен на рисунке 1.1.

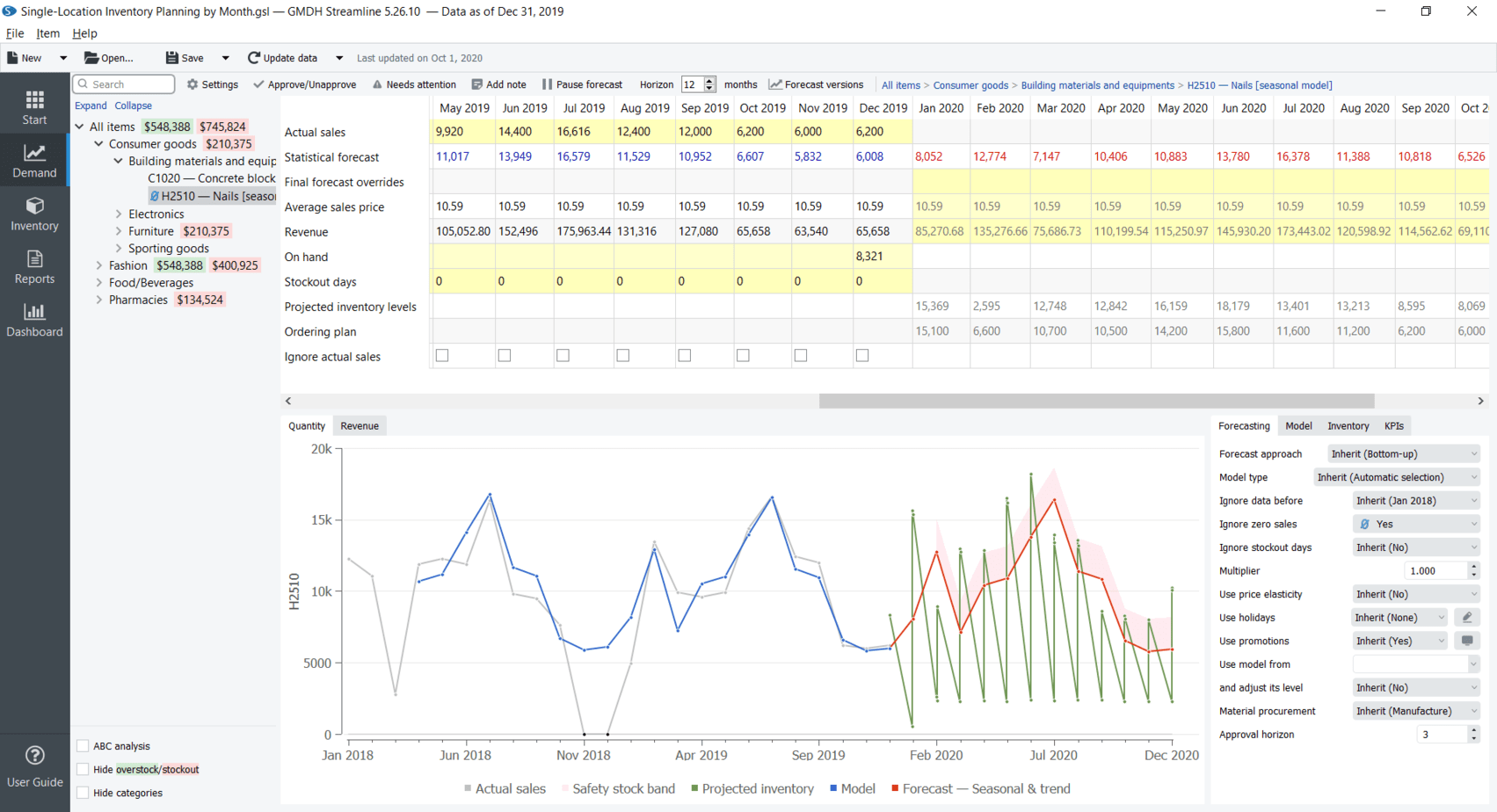


Рисунок 1.1 — *GMDH Streamline*

**Интегрированное бизнес-планирование *SAP***

*SAP* Программное обеспечение для интегрированного бизнес-планирования предлагает решения для планирования и консолидации, которые предоставляют клиентам инструменты финансового прогнозирования, бюджетирования, планирования сценариев, консолидации и совместной работы [3]. Внешний вид представлен на рисунке 1.2.

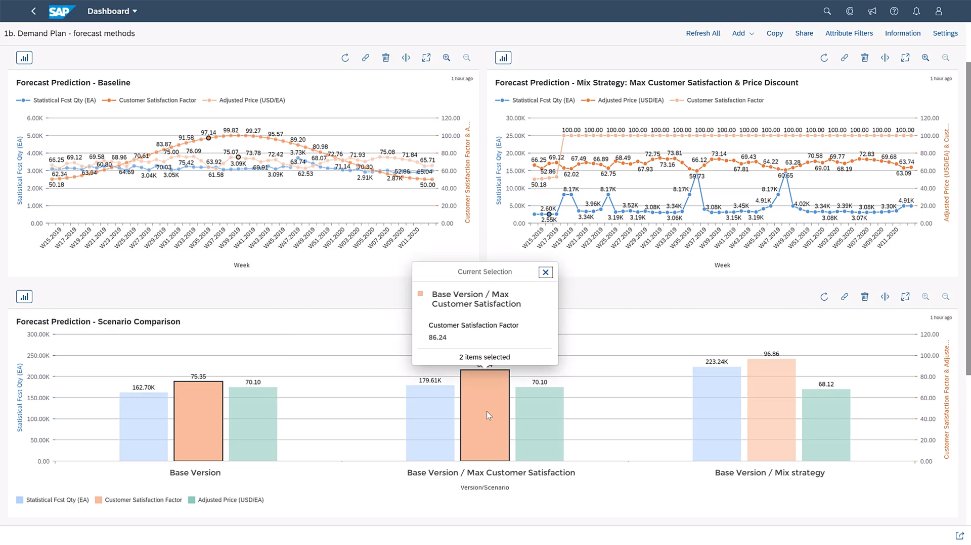


Рисунок 1.2 — Интегрированное бизнес-планирование *SAP*

## 1.3 Техническое задание

Необходимо разработать распределённое приложение со следующими узлами:

* сервер представляет собой базу данных ритейлинговой сети, реализовать *RESTful* веб-*API* для всех необходимых *CRUD* операций;
* узел прогнозирования — оконное приложение, предназначенное для экспертов в области прогнозирования, включающее в себя редактор записей, функцию визуального контроля качества прогнозирования, *RESTful API*, при обращении к которому возвращаются результаты прогнозирования;
* клиент — оконное приложение, предназначенное для руководящего персонала магазинов, включающее в себя функцию создания отчёта (плана закупок конкретных наименований товаров в конкретный магазин сети) в формате *XML*, генерацию *yml*-файла (*Yandex Market Language* [4]) для загрузки товаров в систему «мультимаркет» [5].

Схема базы данных должна быть спроектирована с учётом особенностей набора данных, взятого из открытого доступа (лицензия *CC*0) на платформе *Kaggle* [6], и знаний предметной области.

Модель прогнозирования должна представлять собой адаптивную композицию алгоритмов с несимметричной функцией потерь. Данная функция потерь должна отображать действительные финансовые потери сети. Должна быть возможность задания параметров функции потерь экспертами в области. В условии отсутствия экспертов параметры инициализировать случайными значениями.

## 1.4 Технические требования

Минимальные технические требования для системы:

* поддержка *Python* (версия 3.7 и выше);
* *Microsoft SQL Server* (версия 12.0 и выше);
* операционная система семейства *Windows* (7, 8, 10) или *Linux* (*Ubuntu* 18.04 и выше);
* 4 Гб оперативной памяти;
* 4 Гб свободного места на диске;
* процессор *Intel Core i3* (8-ого поколения и выше);
* менеджер пакетов *Python* (*PIP* или *Anaconda*).

# ПРОЕКТИРОВАНИЕ СИСТЕМЫ

## 2.1 Проектирование бизнес-логики

Для проектирования вариантов использований и действующих лиц можно использовать язык проектирования *UML*.

Исходя из технического задания можно определить следующие действующие лица:

* база данных — содержит информацию о ритейлинговой сети, включая историю продаж и сведения о товарах и отделениях сети;
* сервер БД — предоставляет *RESTful* веб-*API* для запросов к БД со всеми необходимыми *CRUD* (*Create*, *Read*, *Update*, *Delete*) операциями;
* администратор сервера — осуществляет настройку и запуск сервера, на котором располагается база данных;
* система прогнозирования — выполняет запросы к базе данных и на основании полученной истории продаж формирует план закупок;
* эксперт по продажам — осуществляет контроль системы прогнозирования на адекватность, осуществляет контроль за качеством данных, задаёт параметры кусочно-линейной функции потерь
* магазин — отправляет запросы на подготовку плана продаж, формирует *YML*-каталог.

Диаграмма вариантов использования представлена на рисунке 2.1.

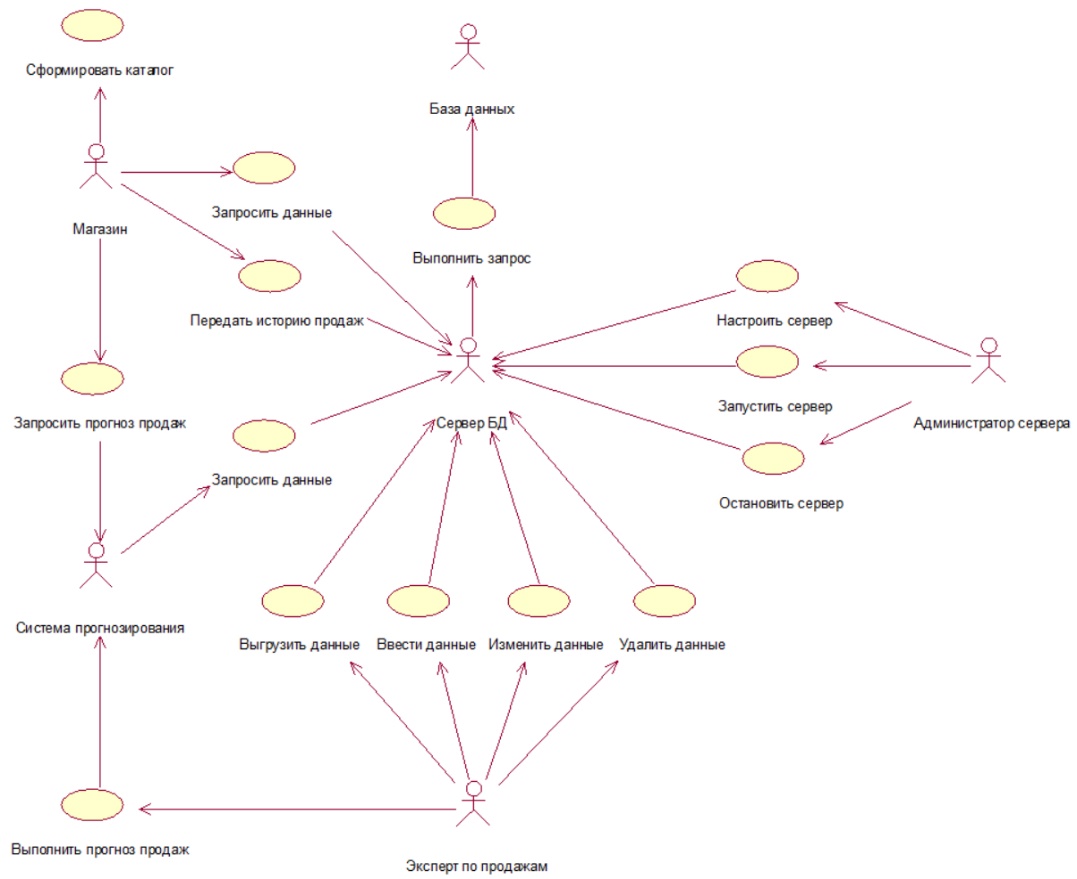


Рисунок 2.1 — Диаграмма вариантов использования

## 2.2 Проектирование базы данных

Проектируемая база данных строится на реляционной модели, в которой данные представлены в виде отношений, связанных между собой внешними ключами.

Основанием проектированной базы данных является набор данных, взятый из открытого доступа, который необходимо нормализовать.

**2.2.1 Проектирование таблиц**

Таблица *StoreTypes* используется для хранения данных о виде точки торговой сети (супермаркет, гипермаркет, дискаунтер). Описание полей таблицы:

* *storetype\_id* — поле со строковым типом данных, хранит информацию из соответствующего поля *storetype\_id* исходного набора данных, имеет фиксированную длину в четыре символа, является первичным ключом;
* *storetype\_description* — поле со строковым типом данных, описывает вид точки торговой сети, в исходном наборе данных соответствующая информация отсутствует.

Таблица *Cities* используется для хранения данных о городах, в которых размещены точки торговой сети (размер, название, страна). Описание полей таблицы:

* *city\_id* — поле со строковым типом данных, хранит информацию из соответствующего поля *city\_id* исходного набора данных, имеет фиксированную длину в четыре символа, является первичным ключом;
* *city\_name* — поле со строковым типом данных, хранит название города, в исходном наборе данных соответствующая информация отсутствует.
* *city\_size* — поле со строковым типом данных, хранит размер города, в исходном наборе данных соответствующая информация отсутствует;
* *country* — поле со строковым типом данных, хранит название государства, в котором находится город, в исходном наборе данных соответствующая информация отсутствует.

Таблица *Stores* используется для хранения данных о точках торговой сети (вид точки, название, город, размер магазина). Описание полей таблицы:

* *store\_id* — поле со строковым типом данных, хранит информацию из соответствующего поля *store\_id* исходного набора данных, имеет фиксированную длину в пять символов, является первичным ключом;
* *storetype\_id* — поле со строковым типом данных, имеет фиксированную длину в четыре символа, является внешним ключом, ссылающимся на поле *storetype\_id* таблицы *StoreTypes*;
* *city\_id* — поле со строковым типом данных, имеет фиксированную длину в четыре символа, является внешним ключом, ссылающимся на поле *city\_id* таблицы *Cities*;
* *store\_size* — поле с целочисленным типом данных, хранит размер точки сети из соответствующего поля *store\_size* исходного набора данных.

Таблица *Products* используется для хранения данных о товарах торговой сети (имя, код, цена и размер). Описание полей таблицы:

* *product\_id* — поле со строковым типом данных, хранит информацию из соответствующего поля *product\_id* исходного набора данных, имеет фиксированную длину в пять символов, является первичным ключом;
* *product\_name* — поле со строковым типом данных, хранит название товара, в исходном наборе данных соответствующая информация отсутствует;
* *hierarchy\_code* — поле со строковым типом данных информацию из соответствующего поля *hierarchy5\_id* исходного набора данных, имеет фиксированную длину в 11 символов;
* *price* — поле с вещественным типом данных, хранит цену товара из соответствующего поля price исходного набора данных;
* *product\_length* — поле с вещественным типом данных, хранит длину товара из соответствующего поля *product\_length* исходного набора данных;
* *product\_depth* — поле с вещественным типом данных, хранит высоту товара из соответствующего поля *product\_depth* исходного набора данных;
* *product\_width* — поле с вещественным типом данных, хранит ширину товара из соответствующего поля *product\_width* исходного набора данных.

Таблица *Stock* используется для хранения данных о наличии товаров в точках сети. Описание полей таблицы:

* *stock\_id* — поле с целочисленным типом данных, является первичным ключом;
* *store\_id* — поле со строковым типом данных, имеет фиксированную длину в пять символов, является внешним ключом, ссылающимся на поле *store\_id* таблицы *Stores*;
* *product\_id* — поле со строковым типом данных, имеет фиксированную длину в пять символов, является внешним ключом, ссылающимся на поле *product\_id* таблицы *Products*;
* *stock* — поле с целочисленным типом данных, хранит количество оставшегося товара из соответствующего поля *stock* исходного набора данных.

Таблица *Purchases* используется для хранения данных о наличии товаров в точках сети. Описание полей таблицы:

* *purchase\_id* — поле с целочисленным типом данных, является первичным ключом;
* *store\_id* — поле со строковым типом данных, имеет фиксированную длину в пять символов, является внешним ключом, ссылающимся на поле *store\_id* таблицы *Stores*;
* *product\_id* — поле со строковым типом данных, имеет фиксированную длину в пять символов, является внешним ключом, ссылающимся на поле *product\_id* таблицы *Products*;
* *purchase\_date* — поле с типом данных «дата» формата «гггг-мм-дд», хранит дату покупки из соответствующего поля *date* исходного набора данных;
* *price* — поле с вещественным типом данных, хранит стоимость товара на момент продажи из соответствующего поля *price* исходного набора данных;
* *sales* — поле с целочисленным типом данных, хранит количество купленного товара из соответствующего поля *sales* исходного набора данных;
* *discount* — поле с вещественным типом данных, хранит размер скидки на товар при его покупке из соответствующего поля *discount* исходного набора данных;
* *revenue* — поле с вещественным типом данных, хранит прибыль от продажи товара из соответствующего поля *revenue* исходного набора данных.

Таблица *LossFunctionParameters* используется для хранения параметров функций потерь, оптимизируемых при прогнозировании продаж конкретных товаров в конкретных точках сети. Описание полей таблицы:

* *papameters\_id* — поле с целочисленным типом данных, является первичным ключом;
* *store\_id* — поле со строковым типом данных, имеет фиксированную длину в пять символов, является внешним ключом, ссылающимся на поле *store\_id* таблицы *Stores*;
* *product\_id* — поле со строковым типом данных, имеет фиксированную длину в пять символов, является внешним ключом, ссылающимся на поле *product\_id* таблицы *Products*;
* *loyalty\_charge\_x* — поле с вещественным типом данных, отражает значение недопрогноза, ведущее к потере лояльности клиентов;
* *loyalty\_charge\_coef* — поле с вещественным типом данных, отражает убытки от потери лояльности клиентов из-за отсутствия товара;
* *storage\_cost\_coef* — поле с вещественным типом данных, отражает потери на хранение товара;
* *bank\_rate\_x* — поле с вещественным типом данных, отражает значение перепрогноза, ведущее к потерям из-за замороженности активов в товаре;
* *bank\_rate\_coef* — поле с вещественным типом данных, отражает убытки от замороженности активов в товаре;
* *product\_cost\_x* — поле с вещественным типом данных, отражает значение перепрогноза, ведущее к потерям из-за нереализованности товара;
* *product\_cost\_coef* — поле с вещественным типом данных, отражает убытки от нереализованности товара.

На рисунке 2.2 изображена логическая схема базы данных, описанной выше.

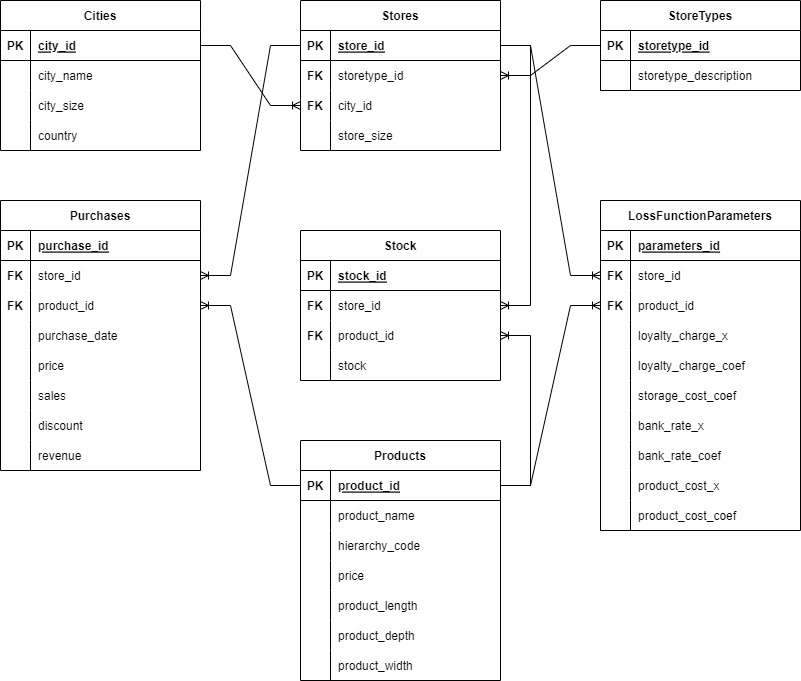


Рисунок 2.2 — Логическая схема базы данных

## 2.3 Проектирование модели прогнозирования

**2.3.1 Постановка задачи**

Имеется временной ряд — значения наблюдений через одинаковые интервалы времени. Обозначив вектором (*y*1, *y*2, …, *yt*) известные значения временного ряда на момент, можно определить задачу прогнозирования, как определение значения временного ряда в момент времени *t*+1. Точнее, нахождение такой функции

*ŷt*+1 = *f*(*y*1, *y*2, …, *yt*; *w*), (2.1)

где *w* — параметры функции *ft*,

которая минимизирует значение функции потерь *L*(*y*1, *y*2, …, *yt*, *yt*+1; *w*) — величину ошибки модели прогнозирования на объекте *yt*+1.

**2.3.2 Описание функции потерь**

Для задач регрессии, которой прогнозирование спроса является, характерно использование метода наименьших квадратов, определяющего функцию потерь, как квадрат разности действительного значения и прогноза. Но это значение не может быть интерпретируемо, квадрат денежных единиц не имеет значения, как единица измерения. Поэтому имеет смысл использовать кусочно-линейную функцию потерь. Простейший пример — модуль ошибки:

*L*(*y*1, *y*2, …, *yt*, *yt*+1; *w*) = |*yt*+1 –  *ŷt*+1|. (2.2)

График функции потерь (2.2) изображён на рисунке 2.3.



Рисунок 2.3 — График модуля ошибки

Для краткости выражение *yt*+1 –  *f*(*y*1, *y*2, …, *yt*; *w*) в дальнейшем будет обозначаться *ε*.

Обобщением модуля ошибки является квантильная функция потерь, если значения недопрогноза и перепрогноза имеют различную стоимость:

*L*(*y*1, *y*2, …, *yt*, *yt*+1; *w*) = (2.3)

где *a, b* — параметры функции потерь, определяющие стоимость недопрогноза и перепрогноза соответственно.

График функции потерь (2.3) изображён на рисунке 2.4.

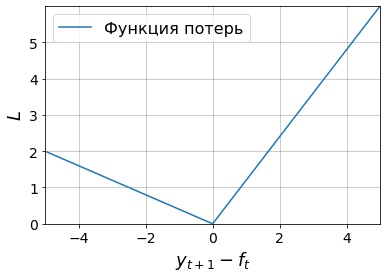


Рисунок 2.4 — График квантильной функции потерь

Более точно функцию потерь в задаче прогнозирования спроса можно задать, учитывая особенности предметной области:

*L*(*y*1, *y*2, …, *yt*, *yt*+1; *w*) (2.4)

где *a* — значение недопрогноза, ведущее к потере лояльности клиентов (*loyalty\_charge\_x* из таблицы *LossFunctionParameters*),

*j* — убытки от потери лояльности клиентов из-за отсутствия товара (*loyalty\_charge\_coef* из таблицы *LossFunctionParameters*),

*k* — потери на хранение товара (storage\_cost\_coef из таблицы *LossFunctionParameters*),

*b* — значение перепрогноза, ведущее к потерям из-за замороженности активов (*bank\_rate\_x* из таблицы *LossFunctionParameters*),

*l* — потери из-за замороженности активов в товаре, банковская ставка (*bank\_rate\_coef* из таблицы *LossFunctionParameters*),

*c* — значение перепрогноза, ведущее к потерям из-за нереализованности товара (*product\_cost\_x* из таблицы *LossFunctionParameters*),

*m* — убытки от нереализованности товара, его стоимость (*product\_cost\_coef* из таблицы *LossFunctionParameters*).

На рисунке 2.5 представлен график функции (2.4).

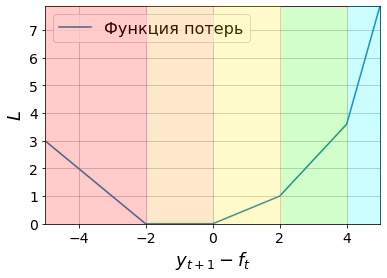


Рисунок 2.5 — График кусочно-линейной функции потерь

Функционал качества — среднее функции потерь по всем объектам:

**2.3.3 Описание модели прогнозирования**

**2.3.3.1 Экспоненциальное скользящее среднее**

Выявление и анализ тенденции ряда часто производится с помощью его выравнивания или сглаживания. Один из простейших и наиболее распространённых методов — экспоненциальное сглаживание. Оно осуществляется по рекуррентной формуле

*ŷt*+1 = *f*(*y*1, *y*2, …, *yt*; *α*) = *αyt* + (1 – *α*)*ŷt*, (2.6)

где *α* — параметр сглаживания, 0 < *α* < 1.

Оптимизация функционала качества (2.5) по параметру *α* выполняется при помощи полного перебора.

**2.3.3.2 Модель Хольта**

Для учитывания линейных трендов применяется модель Хольта [7]:

*ŷt*+1 = *at* + *dbt*, (2.7)

где *at* — прогноз, очищенный от тренда,

*bt* — параметр линейного тренда,

*d* — горизонт прогнозирования.

Параметры *at* и *bt* определяются по следующим формулам:

|  |  |
| --- | --- |
|  | (2.8) |
|  |

где *α*1 и *α*2 — параметры сглаживания, 0 < *α*1 < 1, 0 < *α*2 < 1.

Оптимизация функционала качества (2.5) по параметрам *α*1 и *α*2 выполняется при помощи полного перебора.

**2.3.3.3 Следящий контрольный сигнал. Модель Тригга-Лича**

Используя модель прогнозирования временного ряда, её адекватность можно проверить при помощи следящего контрольного сигнала:

где — экспоненциальное скользящее среднее ошибки прогноза, ,

— экспоненциальное скользящее среднее модуля ошибки прогноза, .

Параметр γ определяется в пределах 0.05…0.1. В модели Тригга-Лича следящий контрольный сигнал используется для определения параметра сглаживания *αt =* |*Kt*|.

**2.3.3.4 Авторегрессия**

Модель авторегрессии описывается следующим выражением:

где *w* — вектор весов, подлежащий оптимизации.

Оптимизацию вектора весов *w* можно производить при помощи градиентного спуска. Основная идея метода заключается в том, чтобы изменять вектор весов в направлении антиградиента функции потерь.

Подставив модель (2.10) в выражение (2.4), получим функцию потерь в явном виде:

*L*(*w*) (2.11)

Продифференцировав функцию потерь (2.11), получим

(2.12)

Тогда изменение вектора весов можно представить в виде:

*wj*+1 = *wj* – *λ*∇*L*(*wj*), (2.13)

где *λ* — шаг градиентного спуска, задающий скорость,

*j* — номер итерации градиентного спуска.

Прекратить градиентный спуск можно по достижению заданного количества итераций или заданной точности изменения функционала качества.

**2.3.3.5 Адаптивная композиция моделей**

Идея адаптивной композиции моделей состоит в том, чтобы вычислять прогноз, как взвешенную сумму прогнозов нескольких моделей. Пусть имеется *n* моделей, тогда прогноз примет вид

где для определения вектора *w* можно использовать адаптивный подбор весов:

где — экспоненциальное скользящее среднее модуля ошибки прогноза, .

# ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ

## 3.1 Выбор средств реализации

Для реализации базы данных была выбрана система управления реляционными базами данных *Microsoft SQL Server.*

Для нормализации исходного набора данных и последующей обработки новых данных был выбран программный продукт *Tableau Prep Builder* в связи с простотой разработки и лёгкостью интеграции с *Microsoft SQL Server.*

Для программной реализации системы прогнозирования спроса был выбран язык программирования *Python*.

Для интеграции *Python* с *MS SQL* используется пакет *pyodbc*.

Для визуального интерфейса используется обёртка *PyImGui* над библиотекой языка *C*++ *Dear ImGui* с интеграцией c *OpenGL*.

Для создания *RESTful* веб-*API* был выбран фреймворк *FastAPI*.

## 3.2 Реализация базы данных

Логическая схема базы данных была спроектирована в подразделе 2.2. Физическая схема базы данных представлена на рисунке 3.1

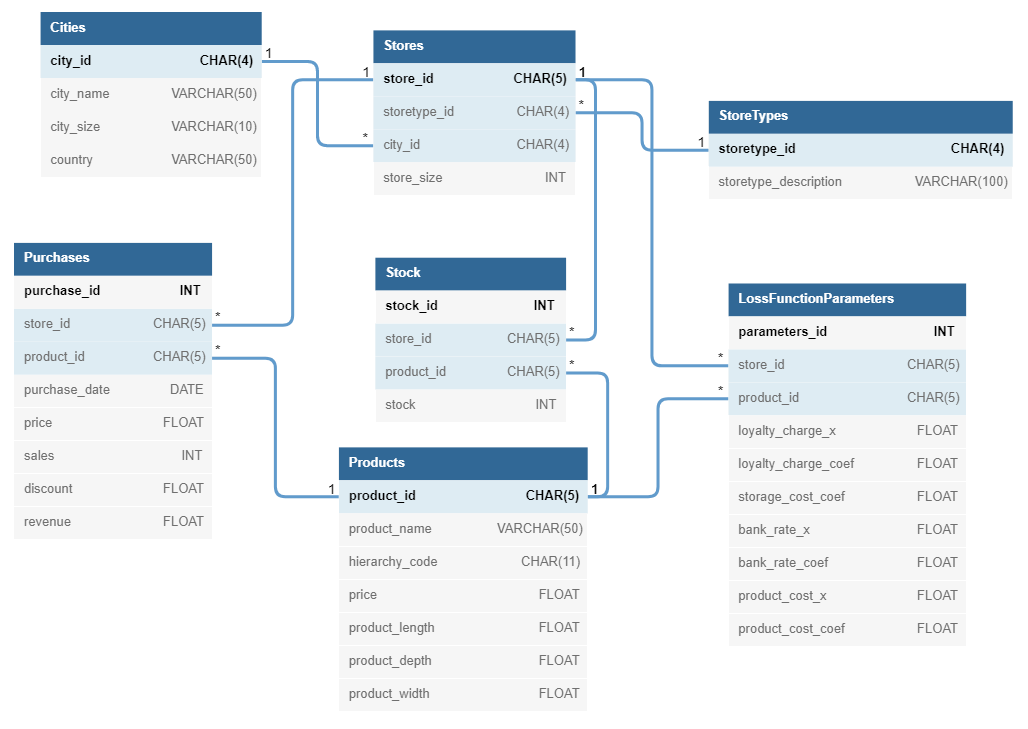


Рисунок 3.1 — Физическая схема базы данных

## 3.3 Реализация потока нормализации

*Tableau Prep Builder* позволяет настраивать потоки данных (*Flow*) с множественными входами и выходами. Входами и выходами могут служить .*csv* файлы, .*xlsx* файлы, облачные и локальные базы данных.

В любой момент времени данные в потоке представляют собой отношение, поэтому над ними могут выполняться операции реляционной алгебры и произвольные пользовательские скрипты на языке *Python*.

На рисунке 3.2 представлен внешний вид среды *Tableau Prep Builder*.

Изображение выглядит как текст, снимок экрана, компьютер, ноутбук

Автоматически созданное описание

Рисунок 3.2 — Внешний вид среды *Tableau Prep Builder*

Слева имеется меню соединений с хранилищами данных для ввода и вывода. Верх правой части занимает область визуализации потока, где он выполняется слева направо и состоит из блоков различного назначения. Низ правой части содержит область работы с конкретным блоком, где можно увидеть статистику из данных текущего состояния потока в этой ветви и настроить блок.

В данном случае выбран блок с группировкой по идентификатору продукта и агрегацией по максимум дату. Это необходимо для того, чтобы определить цену продукта в последний известный промежуток времени, объединив в дальнейшем полученное отношение с исходной информацией о продажах.

Для корректной повторной работы потока необходимо было добавить ряд *insert-of* триггеров, не позволяющих вставлять записи с уже имеющимися значениями первичных ключей.

Полный поток обработки представлен на рисунке 3.3.

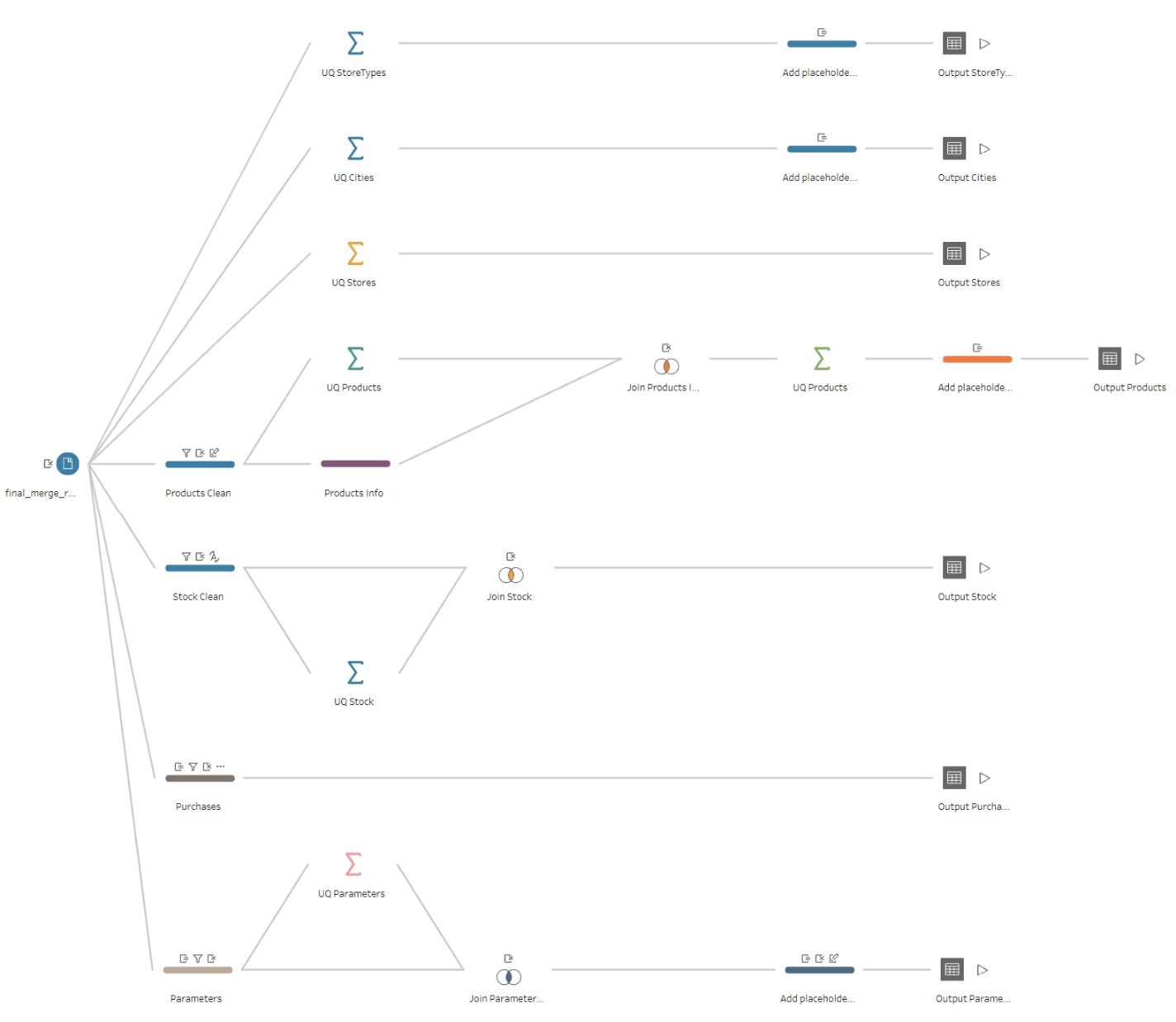


Рисунок 3.3 — Поток нормализации и обработки данных

Данный поток может использоваться как для первичного заполнения базы данных, так и для добавления новых записей о продажах.

## 3.4 Реализация сервера базы данных

Приложение представляет собой минимальный функционал для запуска и остановки сервера.

**3.4.1 Реализация графического интерфейса**

Для реализации *GUI* требуется сперва инициализировать *OpenGL* и получить объект, представляющий системное окно. Это делается с помощью следующей функции модуля *render\_utils*:

*def impl\_glfw\_init(width, height, window\_name):*

*if not glfw.init():*

*print("Could not initialize OpenGL context")*

*sys.exit(1)*

*glfw.window\_hint(glfw.CONTEXT\_VERSION\_MAJOR, 3)*

*glfw.window\_hint(glfw.CONTEXT\_VERSION\_MINOR, 3)*

*glfw.window\_hint(glfw.OPENGL\_PROFILE, glfw.OPENGL\_CORE\_PROFILE)*

*glfw.window\_hint(glfw.OPENGL\_FORWARD\_COMPAT, gl.GL\_TRUE)*

*window = glfw.create\_window(width, height, window\_name, None, None)*

*glfw.make\_context\_current(window)*

*if not window:*

*glfw.terminate()*

*print("Could not initialize Window")*

*sys.exit(1)*

*return window*

Библиотека *PyImGui* поддерживает как непрерывную покадровую отрисовку графического интерфейса, так и обновление только при поступлении действий. Первый вариант требует значимо больше вычислительных ресурсов и может применяться, например, в игровой индустрии. Но, в данном случае, избыточен. Это поведение определяется в следующей функции модуля *render\_utils*:

*def render\_frame(impl, window, render\_function):*

*impl.process\_inputs()*

*imgui.new\_frame()*

*gl.glClearColor(0.1, 0.1, 0.1, 1)*

*gl.glClear(gl.GL\_COLOR\_BUFFER\_BIT)*

*render\_function()*

*imgui.render()*

*impl.render(imgui.get\_draw\_data())*

*glfw.swap\_buffers(window)*

*glfw.wait\_events()*

После инициализации окна запускается цикл его отрисовки в следующей функции модуля *render\_utils*:

*def start\_window(width, height, window\_name, render\_function):*

*imgui.create\_context()*

*window = impl\_glfw\_init(width, height, window\_name)*

*impl = GlfwRenderer(window)*

*while not glfw.window\_should\_close(window):*

*render\_frame(impl, window, render\_function)*

*impl.shutdown()*

*glfw.terminate()*

Три функции, приведённые выше, будут использованы в дальнейшем в таком же виде для отрисовки окон системы прогнозирования и клиента (точки торговой сети). Передаваемыми параметрами являются размеры окна, его имя и ссылка на объект, представляющий собой функцию отрисовки конкретного окна, что позволяет гибко использовать данный код.

На рисунке 3.4 представлен внешний вид окна сервера.

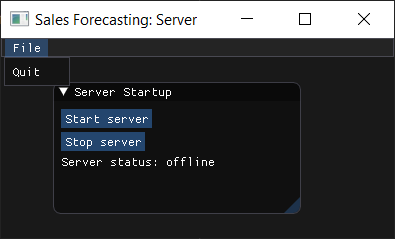


Рисунок 3.4 — Окно сервера

Библиотека *PyImGui* позволяет писать код графического интерфейса, используя не объектно-ориентированный, а процедурный подход. Примером может служить следующий фрагмент кода, обрабатывающий нажатие кнопки остановки сервера и выводящий статус сервера:

*if imgui.button("Stop server"):*

*stop\_server()*

*server\_status = "offline"*

*imgui.text(f"Server status: {server\_status}")*

**3.4.2 Реализация *RESTful веб-API***

Для развёртывания сервера используется пакет *uvicorn*, совместимый с фреймворком *FastAPI*, используемым для написания *RESTful* веб-*API*.

Для начала работы необходимо получить объект класса *fastapi.FastAPI*.

Для подключения конечных точек *API* из нескольких *Python*-пакетов, необходимо определить в них объекты класса *fastapi.APIRouter* и подключить их к *fastapi.FastAPI* с помощью метода *fastapi.FastAPI.include\_router()*.

Содержимое серверной части представлено на рисунке 3.5

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.5 — Серверная часть распределённого приложения

Каждый из .*py*-пакетов в каталоге *endpoints* содержит соответствующие конечные точки *CRUD*-операций для одноимённых таблиц. Далее приведено по одному примеру кода для обработки каждой из *CRUD* операций.

Пример обработки запроса на добавление данных представлен следующей функцией модуля *endpoints.storetypes*:

*@router.post("/add-storetype")*

*def add\_storetype(storetype: StoreType):*

*conn = pyodbc.connect(*

*"Driver={SQL Server Native Client 11.0};"*

*"Server=DESKTOP-P8N0IJI;"*

*"Database=retail;"*

*"Trusted\_Connection=yes;")*

*cursor = conn.cursor()*

*cursor.execute(f"SELECT storetype\_id FROM StoreTypes \*

*WHERE storetype\_id = '{storetype.storetype\_id[:4]}'")*

*data = []*

*for row in cursor:*

*data.append(list(row))*

*if data:*

*conn.close()*

*raise HTTPException(status\_code=422,*

*detail=f"{storetype.storetype\_id[:4]} already exists.")*

*cursor.execute(f"INSERT INTO StoreTypes \*

*VALUES('{storetype.storetype\_id[:4]}', \*

*'{storetype.storetype\_description[:100]}')")*

*conn.commit()*

*conn.close()*

*return {"storetype\_id": storetype.storetype\_id[:4],*

*"storetype\_description":*

*storetype.storetype\_description[:100]}*

Данная функция обёрнута в декоратор, определяющий URL и тип запроса (*POST* в данном случае).

Сперва устанавливается связь с базой данных при помощи *pyodbc.connect().* Затем получается курсор для операции выборки по переданному идентификатору. Если операция выборки содержит непустое множество, то запись с данным идентификатором уже присутствует в базе данных и возвращается *http*-ответ с кодом 422. Иначе запись добавляется в базу данных и возвращается запрашивающему.

Информация для добавления передаётся в виде объекта класса *StoreType*, наследующего от класса *pydantic.BaseModel*. Данный класс поддерживает преобразование во встроенный класс *dict* и в строку-представление *JSON*, что удобно при работе с веб-*API*. Так же модуль *pydantic* предлагает возможности для валидации параметров.

Пример кода обработки запроса на чтение данных представлен в следующем фрагменте кода из модуля *endpoints.stores*:

*@router.get("/get-store/{store\_id}")*

*def get\_store(store\_id: str):*

*conn = pyodbc.connect(*

*"Driver={SQL Server Native Client 11.0};"*

*"Server=DESKTOP-P8N0IJI;"*

*"Database=retail;"*

*"Trusted\_Connection=yes;")*

*cursor = conn.cursor()*

*cursor.execute(f"SELECT \* FROM Stores \*

*WHERE store\_id = '{store\_id}'")*

*data = []*

*for row in cursor:*

*data.append(list(row))*

*conn.close()*

*if data:*

*return {"Data": data}*

*raise HTTPException(status\_code=404,*

*detail=f"{store\_id} not found.")*

Данный код реализует ответ на *GET*-запрос, где идентификатор точки сети передаётся через *URL*, и, если запись с данным идентификатором в базе данных отсутствует, возвращается ответ со кодом 404.

Пример обработки запроса на обновление данных представлен следующим фрагментом кода из модуля *endpoints.storetypes*:

*@router.put("/update-storetype")*

*def update\_storetype(storetype: StoreType):*

*conn = pyodbc.connect(*

*"Driver={SQL Server Native Client 11.0};"*

*"Server=DESKTOP-P8N0IJI;"*

*"Database=retail;"*

*"Trusted\_Connection=yes;")*

*cursor = conn.cursor()*

*cursor.execute(f"SELECT storetype\_id FROM StoreTypes \*

*WHERE storetype\_id = '{storetype.storetype\_id}'")*

*data = []*

*for row in cursor:*

*data.append(list(row))*

*if not data:*

*conn.close()*

*raise HTTPException(status\_code=404,*

*detail=f"{storetype.storetype\_id} not found.")*

*cursor = conn.cursor()*

*cursor.execute(f"UPDATE StoreTypes SET \*

*storetype\_description = \*

*'{storetype.storetype\_description[:100]}' \*

*WHERE storetype\_id = '{storetype.storetype\_id}'")*

*conn.commit()*

*cursor.execute(f"SELECT \* FROM StoreTypes \*

*WHERE storetype\_id = '{storetype.storetype\_id}'")*

*data = []*

*for row in cursor:*

*data.append(list(row))*

*conn.close()*

*return {"Data": data}*

При обработке данного *PUT*-запроса сначала происходит проверка на наличие данного идентификатора в базе данных. Если он присутствует, то формируется *UPDATE SQL*-запрос с имеющимися заполненными полями.

Пример обработки запроса на удаление данных о товаре представлен следующим фрагментом кода из модуля *endpoints.products*:

*@router.delete("/delete-product/{product\_id}")*

*def delete\_product(product\_id: str):*

*conn = pyodbc.connect(*

*"Driver={SQL Server Native Client 11.0};"*

*"Server=DESKTOP-P8N0IJI;"*

*"Database=retail;"*

*"Trusted\_Connection=yes;")*

*cursor = conn.cursor()*

*cursor.execute(f"SELECT product\_id FROM Product \*

*WHERE product\_id = '{product\_id}'")*

*data = []*

*for row in cursor:*

*data.append(list(row))*

*if not data:*

*conn.close()*

*raise HTTPException(status\_code=404,*

*detail=f"{product\_id} not found.")*

*try:*

*cursor.execute(f"DELETE FROM Products \*

*WHERE product\_id = '{product\_id}'")*

*conn.commit()*

*except pyodbc.IntegrityError:*

*raise HTTPException(status\_code=409,*

*detail=f"{product\_id} is being referenced by a foreign key.")*

*finally:*

*conn.close()*

*return {"product\_id": product\_id, "is\_deleted": True}*

При обработке данного *DELETE*-запроса сначала происходит проверка на наличие данного идентификатора в базе данных. Если он отсутствует, возвращается ответ с кодом 404. Далее происходит проверка на то, ссылаются ли записи из других отношений на данную запись. Если проверка не пройдена, то возвращается ответ с кодом 409. Иначе продуктов удаляется и возвращается ответ с кодом 200.

## 3.5 Реализация узла прогнозирования

Данный узел представляет собой оконное приложение для эксперта по продажам и прогнозированию, предоставляющее функционал визуализации прогноза и редактирования записей, если были обнаружены аномалии. Так же этот узел предоставляет конечную точку веб-*API* для получения прогноза обо всех товарах в интересуемом магазине.

**3.5.1 Реализация графического интерфейса**

Графический интерфейс состоит из ряда свободно перемещаемых и масштабируемых областей. Первой из них являются настройки сети: порт сервера базы данных и запуск и остановка сервера прогнозирования. Также здесь находится включение отображения главного меню. За это отвечает следующий фрагмент кода из модуля *forecast\_window*:

*imgui.begin("Network settings")*

*imgui.text("Server connection settings")*

*\_, host = imgui.input\_text("Server host", host, 256)*

*\_, port = imgui.input\_text("Server port", port, 256)*

*imgui.separator()*

*if imgui.button("Start forecast"):*

*try:*

*start\_forecast()*

*forecast\_status = "running"*

*except Exception:*

*forecast\_status = "could not start a forecast"*

*if imgui.button("Stop forecast"):*

*stop\_forecast()*

*forecast\_status = "offline"*

*imgui.text(f"Forecast status: {forecast\_status}")*

*imgui.separator()*

*if imgui.button("Open main menu"):*

*show\_main\_menu = True*

*if imgui.button("Close main menu"):*

*show\_main\_menu = False*

*imgui.end()*

*if show\_main\_menu:*

*main\_menu\_frame(host, port)*

Главное меню представляет собой набор флагов, включающих редакторы записей соответствующих таблиц, и флага отображения области контроля качества прогнозирования. За это отвечает следующий фрагмент кода в модуле *frames.main\_menu\_frame*:

*imgui.begin("Main menu")*

*for editor in active\_editors:*

*\_, active\_editors[editor] = imgui.checkbox(*

*f"Show {editor} editor",*

*active\_editors[editor]*

*)*

*imgui.separator()*

*\_, show\_forecast\_window = imgui.checkbox(*

*f"Show forecast window",*

*show\_forecast\_window*

*)*

*imgui.end()*

*for editor in active\_editors:*

*if active\_editors[editor]:*

*eval(f"{editor}\_frame(host, port)")*

*if show\_forecast\_window:*

*forecast\_window(host, port)*

Для отрисовки графика прогноза используется модуль *matplotlib.pyplot*. Код для генерации графика представлен следующей функцией в модуле *frames.load\_image*:

*def create\_image(purchases\_df, prediction\_results, aggr\_window):*

*plt.plot(prediction\_results, label="predicted")*

*plt.plot(purchases\_df["sales"].values, alpha=0.6, label="true")*

*plt.xticks(*

*np.arange(len(purchases\_df.index), step=TICKS\_STEP[aggr\_window]),*

*purchases\_df.index[::TICKS\_STEP[aggr\_window]].strftime('%Y-%m-%d'),*

*fontsize=10, rotation=90*

*)*

*plt.yticks(fontsize=12)*

*plt.legend(fontsize=14)*

*plt.savefig('prediction.jpg', bbox\_inches="tight")*

*plt.close()*

Для отображения изображений в *PyImGui* необходимо работать с бэкендом — интегрированным *OpenGL*. Привязка изображения к идентификатору, его получение и настройка параметров отображения реализованы в следующей функции в модуле *frames.load\_image*:

*def get\_textureID():*

*image = Image.open("prediction.jpg")*

*image = image.convert('RGB')*

*img\_data = np.array(list(image.getdata()), np.uint8)*

*textureID = gl.glGenTextures(1)*

*gl.glPixelStorei(gl.GL\_UNPACK\_ALIGNMENT, 1)*

*gl.glBindTexture(gl.GL\_TEXTURE\_2D, textureID)*

*gl.glTexParameterf(gl.GL\_TEXTURE\_2D, gl.GL\_TEXTURE\_MIN\_FILTER, gl.GL\_NEAREST)*

*gl.glTexParameterf(gl.GL\_TEXTURE\_2D, gl.GL\_TEXTURE\_MAG\_FILTER, gl.GL\_NEAREST)*

*gl.glTexParameterf(gl.GL\_TEXTURE\_2D, gl.GL\_TEXTURE\_WRAP\_S, gl.GL\_CLAMP\_TO\_EDGE)*

*gl.glTexParameterf(gl.GL\_TEXTURE\_2D, gl.GL\_TEXTURE\_WRAP\_T, gl.GL\_CLAMP\_TO\_EDGE)*

*gl.glTexParameteri(gl.GL\_TEXTURE\_2D, gl.GL\_TEXTURE\_BASE\_LEVEL, 0)*

*gl.glTexParameteri(gl.GL\_TEXTURE\_2D, gl.GL\_TEXTURE\_MAX\_LEVEL, 0)*

*gl.glTexImage2D(*

*gl.GL\_TEXTURE\_2D, 0, gl.GL\_RGB, image.size[0],*

*image.size[1], 0, gl.GL\_RGB, gl.GL\_UNSIGNED\_BYTE, img\_data*

*)*

*image.close()*

*return textureID, image.size*

**3.5.2 Реализация *RESTful веб-API***

В данном случае необходимо реализовать только конечную точку для получения результатов прогнозов. Она также написана с помощью фреймворка *FastAPI*, и сервер развёрнут при помощи *uvicorn*. Здесь также происходят проверки на наличие *store\_id* и *product\_id* в базе данных. Кроме этого, необходимо наличие параметров функции потерь в таблице *LossFunctionParameters* и, разумеется, транзитивная доступность сервера базы данных, для чего тоже реализованы проверки и возвраты ответов с кодом 404.

**3.5.3 Реализация модели прогнозирования**

**3.5.3.1 Реализация функции потерь**

Функция для расчёта значения функции потерь (2.4) представлена следующей функцией из модуля *forecasting.loss\_function:*

*def get\_loss(y\_true, y\_pred, params):*

*offset = y\_pred - y\_true*

*if offset < params[0]:*

*return (offset-params[0])\*params[1]*

*if offset < 0:*

*return 0*

*if offset < params[3]:*

*return offset\*params[2]*

*if offset < params[5]:*

*return (params[2]\*params[3]*

*+ (offset - params[3])\*(params[2] + params[4]))*

*return (params[2]\*params[3]*

*+ (params[5] - params[3])\*(params[2] + params[4])*

*+ (offset - params[5])\*(params[2] + params[4] + params[6]))*

Расчёт производной функции (2.4), приведённой в формуле (2.12) за исключением умножения на значение *y*, представлен следующей функцией из модуля *forecasting.loss\_function:*

*def get\_loss\_derivative(y\_true, y\_pred, parameters):*

*offset = y\_pred - y\_true*

*if offset < parameters[0]:*

*return parameters[1]*

*if offset < 0:*

*return 0.01*

*if offset < parameters[3]:*

*return parameters[2]*

*if offset < parameters[5]:*

*return parameters[2] + parameters[4]*

*return parameters[2] + parameters[4] + parameters[6]*

**3.5.3.2 Реализация адаптивной композиции**

Для соответствия всех моделей общей логике создан следующий абстрактный класс в модуле *forecasting.models.base*:

*class Model(ABC):*

*@abstractmethod*

*def predict(self):*

*pass*

*@abstractmethod*

*def fit(self):*

*pass*

Экспоненциальное скользящее среднее (подпункт 2.3.3.1) представлено следующим классом в модуле *forecasting.models.ema*:

*class EMA(Model):*

*alpha = 0.01*

*y\_prev = 0*

*def \_\_init\_\_(self, alpha=None, y\_prev=None):*

*if alpha is not None:*

*self.alpha = alpha*

*if y\_prev is not None:*

*self.y\_prev = y\_prev*

*def predict(self, y\_true):*

*self.y\_prev = self.alpha\*y\_true + (1-self.alpha)\*self.y\_prev*

*return self.y\_prev*

*def fit(self, y, loss\_params):*

*best\_loss = np.inf*

*best\_alpha = 0.01*

*for alpha in np.arange(0.05, 1, 0.05):*

*y\_pred = []*

*self.alpha = alpha*

*self.y\_prev = y[0]*

*for y\_true in y[:-1]:*

*y\_pr = self.predict(y\_true)*

*y\_pred.append(y\_pr)*

*loss = sum(map(*

*get\_loss, y[1:], y\_pred, repeat(loss\_params))*

*) / len(y - 1)*

*if best\_loss > loss:*

*best\_loss = loss*

*best\_alpha = alpha*

*self.alpha = best\_alpha*

В нём определён конструктор *\_\_init\_\_()*, устанавливающий параметр сглаживания и начальное приближение. Переопределены абстрактные методы *fit* и *predict*.Первый реализует вычисление прогноза по формуле (2.6), второй — подбор параметра сглаживания оптимизацией функционала (2.5) полным перебором.

Модель Хольта (подпункт 2.3.3.2) представлена классом *Holts* в модуле *forecasting.models.holts.* Переопределение метода *fit* аналогично экспоненциальному скользящему среднему, но содержит вложенный цикл для перебора второго параметра сглаживания. Метод *predict*, реализующий расчёт прогноза по формуле (2.8) представлен следующим фрагментом кода:

*def predict(self, y\_true, dist=1):*

*a\_prev = self.a*

*self.a = self.alpha1\*y\_true + (1-self.alpha1)\*(self.a-self.b)*

*self.b = self.alpha2\*(self.a-a\_prev) + (1-self.alpha2)\*self.b*

*return self.a + self.b\*dist*

Следящий контрольный сигнал (подпункт 2.3.3.3) представлен классом *TrackingSignal* в модуле *forecasting.models.tracking\_signal* использует два объекта класса *EMA*, инициализируемых в конструкторе. Метод *predict*  вычисляет значение следящего контрольного сигнала с проверкой на деление на ноль.

*def \_\_init\_\_(self):*

*self.error\_ema = EMA(0.05, 0)*

*self.error\_abs\_ema = EMA(0.05, 0)*

*def predict(self, y\_true):*

*error = y\_true - self.y\_prev*

*error\_pred = self.error\_abs\_ema.predict(abs(error))*

*if error\_pred == 0:*

*error\_pred = 0.01*

*self.alpha = abs(self.error\_ema.predict(error) / error\_pred)*

*self.y\_prev = self.alpha\*y\_true + (1-self.alpha)\*self.y\_prev*

*return self.y\_prev*

Модель авторегрессии (подпункт 2.3.3.4) представлена классом *Autoregression* в модуле *forecasting.models.autoregression*. Метод *\_\_init\_\_()* инициализирует значение весов нулями. Метод *predict*  вычисляет значение прогноза, как произведение матрицы объектов на транспонированную матрицу весов. Метод *fit()* реализует градиентный спуск по формуле (2.13).

*class Autoregression(Model):*

*weights = None*

*def \_\_init\_\_(self, feature\_count):*

*self.weights = np.zeros((feature\_count,))*

*def predict(self, X):*

*return X.dot(self.weights.T)*

*def fit(self, X, y, loss\_params, step\_size=1e-5, n\_iter=100):*

*for \_ in range(n\_iter):*

*y\_pred = self.predict(X)*

*loss\_der = np.array(list(map(*

*get\_loss\_derivative,*

*y[1:],*

*y\_pred,*

*repeat(loss\_params)*

*)))*

*gradient = X[1:].T.dot(loss\_der) / (y.shape[0]-1)*

*self.weights -= gradient\*step\_size*

Модель адаптивной композиции (подпункт 2.3.3.5) представлена классом *AdaptiveComposition* в модуле *forecasting.models.adaptive\_composition*. Конструктор принимает список моделей, для каждой из которых создаётся экспоненциальное скользящее среднее модуля ошибки. Метод *fit()* вызывает соответствующий метод для каждой модели композиции. Метод *predict()* вызывает соответствующий метод для каждой модели композиции, рассчитывает веса, используя экспоненциальные скользящие средние модулей ошибок, и возвращает взвешенную сумму.

Содержимое узла прогнозирования проекта представлено на рисунке 3.6.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.6 — Содержимое узла прогнозирования распределённого приложения

## 3.6 Реализация клиента

Клиент представляет собой оконное приложение с настройками сети для установки портов серверов базы данных и прогнозирования. Функционал включает в себя генерацию *XML*-файла закупок, *YML*-файла товаров для загрузки на платформу «мультимаркет», передачу отчёта по продажам для обновления информации в базе данных.

Для генерации XML-файла используется модуль *xml.etree.ElementTree*. Фрагмент кода, генерирующий структуру файла, представлен ниже:

*products = ET.Element(*

*"products",*

*{"date": datetime.now().strftime("%Y-%m-%d %H:%M"),*

*"store\_id": store\_id}*

*)*

*for product\_id in products\_list:*

*response\_get\_prediction = requests.get(*

*f"http://{forecast\_host}:{forecast\_port}/predict/",*

*params={"store\_id": store\_id,*

*"product\_id": product\_id,*

*"host": server\_host,*

*"port": server\_port}*

*)*

*if response\_get\_prediction.status\_code == 200:*

*product\_prediction = response\_get\_prediction.json()*

*product = ET.SubElement(products, "product", id=product\_id)*

*product.text = str(product\_prediction[0])*

Сначала генерируется корневой тег products, содержащий атрибуты *date* и *store\_id*. Затем внутри этого тега генерируются теги *product* с атрибутом *id*, содержащим *product\_id*. Внутрь тега *product* помещается подлежащее к закупке количество товара.

Модуль *xml.etree.ElementTree* записывает весь XML-файл в одну строку, поэтому для удобства чтения необходимо реализовать функцию, добавляющие знаки табуляции (или пробелы) и переносы на новую строку. Ниже представлена реализация подобной функции из модуля *xml\_generation.procurement\_plan*:

*def indent(elem, level=0):*

*i = "\n" + level\*" "*

*if len(elem):*

*if not elem.text or not elem.text.strip():*

*elem.text = i + " "*

*if not elem.tail or not elem.tail.strip():*

*elem.tail = i*

*for elem in elem:*

*indent(elem, level+1)*

*if not elem.tail or not elem.tail.strip():*

*elem.tail = i*

*else:*

*if level and (not elem.tail or not elem.tail.strip()):*

*elem.tail = i*

Применение приведённой выше функции и запись в файл представлены фрагментом кода ниже:

*indent(products)*

*tree = ET.ElementTree(products)*

*tree.write(f"procurement\_plan.xml", encoding="utf-8")*

Содержимое клиентской части распределённого приложения представлено на рисунке 3.7.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.7 — Клиентская часть распределённого приложения

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Общее руководство пользователя по графическому интерфейсу:

1. двойное нажатие по названию области сворачивает область;
2. зажатие и перемещение правого нижнего угла области растягивает область;
3. двойное нажатие по правому нижнему углу области подстраивает размер области под её содержимое;
4. зажатие в любом пустом месте области и перемещение передвигает область по окну;
5. *Tab*/*Shift*+*Tab* перемещает курсор в следующее поле ввода;
6. колесо мыши для прокрутки;
7. при редактировании текста:
   1. зажмите *Shift* или используйте мышку для выделения;
   2. *Ctrl*+*Left*/*Right* для перемещения курсора к предыдущему/следующему слову;
   3. *Ctrl*-*A* или двойное нажатие выделяет весь текст;
   4. *Ctrl-X*, *Ctrl-C*, *Ctrl-V* для использования буфера обмена;
   5. *Ctrl-Z*, *Ctrl-Y* для отмены действия и его повтора;
   6. *Escape* для отмены всех изменений и сброса выделения редактируемого поля;
   7. можно использовать +, \*, / для арифметических операций, +– для вычитания.

## 4.1 Руководство по серверу базы данных

Внешний вид оконного приложения сервера базы данных представлен на рисунке 4.1

Изображение выглядит как текст, снимок экрана, экран

Автоматически созданное описание

Рисунок 4.1 — Внешний вид приложения сервера БД

Кнопка «*Start server*» запускает сервер базы данных на локальном порту 5000. Нажатие на кнопку «*Stop server*» останавливает сервер. При закрытии оконного приложения остановка сервера и освобождение порта происходят автоматически.

## 4.2 Руководство по узлу прогнозирования

При запуске оконного приложения узла программирования сперва появится область настроек сети, изображённая на рисунке 4.2.

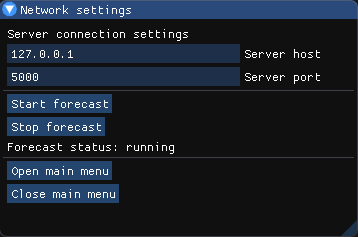


Рисунок 4.2 — Настройки сети узла прогнозирования

В первых двух текстовых полях необходимо прописать *IP*-адрес и номер порта сервера базы данных. При использовании *URL*-адреса необходимо ввести его в поле «*Server host*» и оставить после «*Server port*» пустым.

Кнопка «*Start forecast*» запускает сервер прогнозирования на локальном порту 5001. Нажатие на кнопку «*Stop forecast*» останавливает сервер прогнозирования. При закрытии оконного приложения остановка сервера и освобождение порта происходят автоматически.

При нажатии на кнопку «*Open main menu*» появляется область, изображённая на рисунке 4.3.

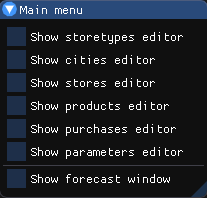


Рисунок 4.3 — Главное меню

Первые шесть флагов отвечают за редакторы записей в базе данных. Рассмотрим редактор записей о городах для примера. При установке флага «*Show cities editor*» появляется область, изображённая на рисунке 4.4.

Изображение выглядит как текст, снимок экрана, экран

Автоматически созданное описание

Рисунок 4.4 — Редактор записей таблицы *Stores*

При нажатии на кнопку «*Load cities list*» выполняется *GET*-запрос по адресу *http://127.0.0.1:5000/cities/get-cities*, получающий от сервера базы данных список всех идентификаторов городов. После этого необходимо нажать на кнопку «*Show cities list*», при этом появится меню выбора городов, изображённое на рисунке 4.5.

Изображение выглядит как текст, монитор, снимок экрана, экран

Автоматически созданное описание

Рисунок 4.5 — Меню выбора городов

В этом меню при выборе одного или нескольких городов появляется меню просмотра и редактирования записей, изображённое на рисунке 4.6.

Изображение выглядит как текст, монитор, снимок экрана, экран

Автоматически созданное описание

Рисунок 4.6 — Меню просмотра и редактирования

Необходимо обратить внимание на то, что подпись заполняемого поля находится справа от него.

При нажатии на кнопку «*Update cities*» все изменённые записи отправятся серверу базы данных.

При нажатии на кнопку «*Delete cities*» все выбранные записи будут удалены.

При заполнении поле в нижней части области редактирования и нажатии на кнопку «*Add a city*» новая запись будет добавлена в базу данных.

При отсутствии ответа от сервера поверх всех областей будет выведена ошибка, изображённая на рисунке 4.7.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.7 — Ошибка «Сервер недоступен»

При нажатии на кнопку «*Show cities list*» перед «*Load cities list*» будет выведено сообщение об ошибке, представленное на рисунке 4.8.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.8 — Ошибка отсутствия списка городов

При попытке добавить запись с уже имеющимся идентификатором будет выведено сообщение об ошибке, представленное на рисунке 4.9.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.9 — Ошибка наличия записи

При попытке удалить запись, на которую ссылаются записи других отношений, будет выведено сообщение об ошибке, представленное на рисунке 4.10.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.10 — Ошибка удаления записи

При установке в главном меню флага «*Show forecast window*» будет выведена область графического интерфейса, представленная на рисунке 4.11.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.11 — Область прогнозирования

В поля «*Get by store\_id*» и «*Get by product\_id*» необходимо ввести идентификаторы магазина и продукта соответственно. После ввода идентификаторов и нажатия на кнопку «*Predict sales*» будет выведен график прогноза за имеющуюся историю продаж. Пример графика представлен на рисунке 4.12.

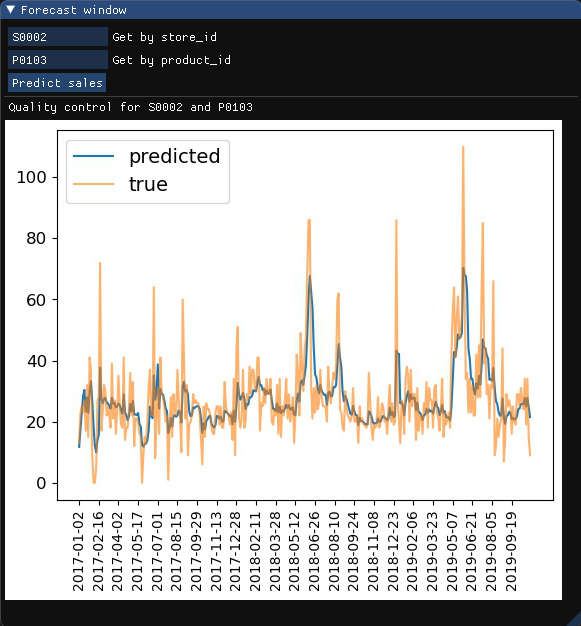


Рисунок 4.12 — Пример прогноза продаж

Пример внешнего вида всего оконного приложения представлен на рисунке 4.13.

Изображение выглядит как текст, монитор, снимок экрана, внутренний

Автоматически созданное описание

Рисунок 4.13 — Внешний вид окна узла прогнозирования

## 4.3 Руководство клиента

При запуске оконного приложения клиента появится область настроек сети, изображённая на рисунке 4.14.

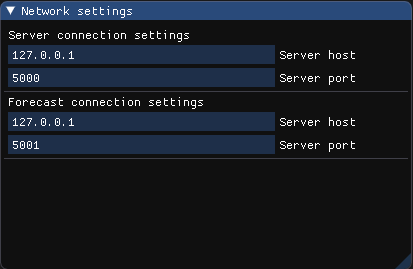


Рисунок 4.14 — Настройки сети узла прогнозирования

В первых двух текстовых полях необходимо прописать *IP*-адрес и номер порта сервера базы данных. При использовании *URL*-адреса необходимо ввести его в поле «*Server host*» и оставить после «*Server port*» пустым.

Во второй паре текстовых полей необходимо прописать *IP*-адрес и номер порта сервера узла прогнозирования. При использовании *URL*-адреса необходимо ввести его в поле «*Forecast host*» и оставить после «*Forecast port*» пустым.

Также появится область для генерации отчётности, представленная на рисунке 4.15.

Изображение выглядит как текст, снимок экрана, знак, закрыть

Автоматически созданное описание

Рисунок 4.15 — Область генерации отчётности

В первое текстовое поле необходимо ввести путь, по которому будет создан файл отчёта. Если введён абсолютный путь, то он используется в нетронутом виде, если введён относительный путь, то он присоединяется к текущей рабочей директории.

Во второе текстовое поле необходимое ввести идентификатор магазина, для которого генерируется лист закупок или *YML* каталог.

При нажатии на кнопку «*Generate procurement plan*» будет осуществлён запрос к узлу прогнозирования и создан файл *procurement\_plan.xml*, содержимое которого представлен на рисунке 4.16.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 4.16 — Содержимое файла *procurement\_plan.xml*

При нажатии на кнопку «*Generate YML catalogue*» будет осуществлён запрос к узлу прогнозирования и созданы файлы *catalogue\_номерфайла.xml*, содержимое которого представлен на рисунках 4.16 и 4.17.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 4.16 — Категории товаров *catalogue\_1.xml*

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.17 — Список товаров *catalogue\_1.xml*

Так как платформа «мультимаркет» накладывает ограничения на размер файла и количество товаров в одном файле, их может быть сгенерировано несколько штук.

# ЗАКЛЮЧЕНИЕ

В соответствии с поставленной задачей создано распределённое приложение, состоящее из трёх узлов: сервера базы данных, узла прогнозирования (выполняющего функции как клиента, так и сервера) и клиента в виде точки торговой сети.

Спроектирована и реализована база данных, заполненная набором данных из открытого доступа, нормализованного при помощи ПО *Tableau Prep Builder*. Создан сервер с *RESTful* веб-*API*, предоставляющий возможность выполнения *CRUD* операций.

Спроектирована и реализована модель прогнозирования, представляющая собой адаптивную композицию моделей с несимметричной функцией потерь. Для контроля качества, выявления аномалий в данных, предоставления *API* для прогнозирования создан узел прогнозирования распределённой системы.

Спроектирован и реализован клиент, предоставляющий функции генерации отчётности.

Полный исходный код размещён на платформе *GitHub* [8].

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Коваль А. С. Построение адаптивных композиций алгоритмов прогнозирования при несимметричной функции потерь – М.: МФТИ, 2009. – 59 с.
2. *https://gmdhsoftware.com/ru/*
3. *https://www.sap.com/products/integrated-business-planning.html*
4. *https://yandex.ru/support/partnermarket-catalog/export/yml.html*
5. *https://www.kaggle.com/datasets/berkayalan/retail-sales-data*
6. *https://multimarket.pro/*
7. Лукашин, Ю. П. Адаптивные методы краткосрочного прогнозирования временных рядов / Ю. П. Лукашин: Учеб. пособие. – М.: Финансы и статистика, 2003. – 416 с.
8. *https://github.com/kosobutskyVlad/DBCP*

# ВЕДОМОСТЬ КУРСОВОГО ПРОЕКТА

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | *Дополнительные сведения* | | | | |
|  | | | | | *Текстовые документы* |  | | | | |
|  | | | | |  |  | | | | |
| *БГУИР КП 1-53 01 02 01 10 ПЗ* | | | | | *Пояснительная записка* | *51с.* | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  |  |  |  |  | *БГУИР КП 1-53 01 02 01 10 Д1* | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм.* | *Л* | *№ докум.* | *Подп* | *Дата* | *Автоматизированная система прогнозирования спроса*  Ведомость курсового проекта | *Лит* | | | *Лист* | *Листов* |
| *Разраб.* | | *Кособуцкий* |  |  |  | *Т* |  | *50* | *50* |
| *Провер.* | | *Трофимович* |  |  | *Кафедра ИТАС*  *гр. 920603* | | | | |
| *Т.контр.* | |  |  |  |
| *Н.контр.* | |  |  |  |
| *Утв.* | |  |  |  |