МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра теории функций и стохастического анализа

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ СКЛАДСКОГО УЧЁТА, ОРИЕНТИРОВАННОГО НА МАЛЫЕ ПРЕДПРИЯТИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 451 группы
направления 38.03.05 — Бизнес-информатика

механико-математического факультета
Быковой Дарьи Михайловны

Научный руководитель
доцент, к. ф.-м. н. _____ О. А. Мыльцина

Заведующий кафедрой
доцент, д. ф.-м. н. _____ С. П. Сидоров

Саратов 2025

ВВЕДЕНИЕ

Актуальность темы исследования. За последние годы в России наблюдается рост доли малого и среднего бизнеса. Во многом этому поспособствовал Национальный проект «Малое и среднее предпринимательство и поддержка индивидуальной предпринимательской инициативы», который являлся одним из национальных проектов, запущенных в период с 2019 по 2024 годы. Этот проект был направлен на поддержку и развитие малого и среднего бизнеса, а также на стимулирование индивидуальной предпринимательской инициативы. По данным Российской газеты в начале 2025 года численность занятых в секторе малого и среднего предпринимательства в России достигла 29,5 миллиона человек. Это составляет около 39% от общего числа работающего населения страны, что свидетельствует о значительном вкладе МСП в экономику России. Кроме того, по итогам первого квартала 2025 года количество малых и средних предприятий в России превысило 6,7 миллионов, что является рекордным показателем с начала ведения специального реестра субъектов МСП в 2016 году — об этом сообщается на официальном сайте Минэкономразвития.

В условиях стремительного роста сектора малого и среднего бизнеса особенно остро встаёт вопрос цифровизации управленческих и операционных процессов. Одной из ключевых задач для малых предприятий, особенно в сфере торговли и производства, становится эффективное управление складскими запасами и логистикой. Однако готовые программные продукты зачастую оказываются либо избыточно сложными и дорогими, либо не соответствуют специфике конкретного бизнеса. В этой связи особую актуальность приобретает разработка адаптированных, простых в использовании веб-приложений для складского учёта, которые могут быть быстро внедрены и модифицированы в соответствии с потребностями предприятия.

Актуальность определила выбор темы данной работы: «Разработка веб-приложения для складского учёта, ориентированного на малые предприятия».

Целью работы является разработка веб-приложения для складского учёта, адаптированного под нужды малого бизнеса, с использованием современных и доступных средств разработки.

Для достижения поставленной цели необходимо решить следующие задачи:

— Изучить теоретический материал, связанный с разработкой веб-приложений;
— Разработать концептуальную схему базы данных, отражающую структуру хранения информации о поставщиках, покупателях, материалах, закупках, поставках и продажах;
— Подобрать программные средства для реализации приложения: систему управления базами данных, инструменты взаимодействия с базой данных и средства создания пользовательского интерфейса;
— Реализовать структуру базы данных в PostgreSQL и обеспечить взаимодействие с ней с помощью SQLAlchemy;
— Разработать веб-приложение на платформе Streamlit для удобного взаимодействия пользователя с системой;
— Протестировать разработанное приложение и проанализировать его функциональность, соответствие требованиям и возможности дальнейшего развития.

Практическая значимость работы заключается в разработке функционирующего веб-приложения, которое может быть использовано малыми предприятиями для автоматизации процессов складского учёта. Разработанная система позволяет учитывать поставщиков и покупателей, отслеживать закупки и поставки материалов на склад, а также фиксировать продажи. Приложение отличается доступностью, простотой в использовании и адаптивностью к потребностям конкретного предприятия. Благодаря использованию открытых и свободно распространяемых технологий — PostgreSQL, SQLAlchemy и Streamlit — система может быть развёрнута без значительных затрат и доработана под специфические задачи.

Основное содержание работы

Бакалаврская работа состоит из: введения, трёх теоретических и одного практического раздела, заключения, списка использованных источников, содержащего 20 наименований и 13 приложений. Общий объем работы составляет 42 страницы.

Первый раздел «Реляционные базы данных и СУБД PostgreSQL» посвящён базовым принципам реляционной модели и особенностям работы СУБД PostgreSQL.

Современные информационные системы используют базы данных для хранения и обработки структурированной информации. Наиболее распространённой является реляционная модель данных, предложенная Э. Ф. Коддом. Её основой являются таблицы (отношения), содержащие строки (записи) и столбцы (атрибуты). Каждая таблица имеет первичный ключ — уникальный идентификатор записи. Внешние ключи обеспечивают связи между таблицами и поддерживают ссылочную целостность данных.

Существуют три основных типа связей: один к одному, один ко многим и многие ко многим (через промежуточную таблицу). Для внешних ключей могут задаваться правила каскадного обновления и удаления, что повышает согласованность данных.

Нормализация — процесс упорядочивания структуры данных для устранения избыточности. На практике чаще всего применяются первые три нормальные формы: первая — все значения атомарны; вторая — все неключевые поля зависят от всего первичного ключа; третья — устранены транзитивные зависимости. Это помогает избежать аномалий при обновлении данных.

Целостность данных обеспечивается на трёх уровнях: уникальность записей (целостность сущности), корректные связи между таблицами (ссылочная целостность) и соответствие типов данных (целостность домена). Также важную роль играют транзакции, которые позволяют выполнять набор операций как единое целое, соответствующее принципам ACID.

СУБД PostgreSQL — современная система управления базами данных с открытым исходным кодом. Она поддерживает все основные принципы реляционной модели, отличается высокой надёжностью, масштабируемостью и активно используется как в бизнесе, так и в научных проектах. Благодаря поддержке расширяемости и возможности добавления пользовательских типов данных, функций и индексов, PostgreSQL является идеальной платформой для разработки приложений, требующих высокой гибкости в работе с данными. Её выбор в рамках дипломной работы обусловлен функциональностью, гибкостью и активной поддержкой сообщества.

Второй раздел «Объектно-реляционное отображение и библиотека SQLAlchemy» описывает библиотеку SQLAlchemy — инструмент ORM (Object-Relational Mapping) для Python, который упрощает взаимодействие с реляционной базой данных, позволяя работать с данными через объектно-ориентированные конструкции.

ORM позволяет описывать структуру таблиц и связи между ними в виде Python-классов, где атрибуты соответствуют столбцам, а механизмы вроде relationship() реализуют связи между сущностями. Такой подход делает код более наглядным, приближённым к предметной области, и способствует лучшей архитектуре приложения.

SQLAlchemy поддерживает как декларативный стиль (описание моделей через классы), так и прямую работу с SQL, что делает её удобной как для абстрактных, так и для ресурсоёмких задач. Одним из ключевых компонентов является сессия (Session), через которую выполняются запросы, отслеживаются изменения и управляются транзакции (commit, rollback). Это обеспечивает надёжность и согласованность данных.

В рамках дипломного проекта SQLAlchemy используется для описания моделей, настройки связей, управления транзакциями и реализации бизнес-логики. Такой подход позволяет разделить уровни логики, повысить читаемость кода и упростить поддержку системы. Возможность интеграции моделей с веб-фреймворками делает SQLAlchemy универсальным решением для современных web-приложений.

Третий раздел «Средства создания пользовательского интерфейса на базе Streamlit» описывает фреймворк Streamlit, предназначенному для быстрой разработки веб-интерфейсов к Python-приложениям. Рассматриваются принципы работы Streamlit, его компоненты и возможности для отображения данных, а также особенности создания интерфейса в рамках задачи.

Streamlit позволяет разрабатывать интерактивные веб-приложения без необходимости использовать HTML, CSS или JavaScript. Он использует декларативный подход: элементы управления добавляются в код как обычные команды Python, а обновление интерфейса происходит при каждом запуске скрипта.

Streamlit поддерживает текстовые блоки (st.text, st.markdown), кнопки (st.button), выпадающие списки (st.selectbox), чекбоксы (st.checkbox), слайдеры (st.slider) и создание таблиц и графиков с помощью Pandas, Matplotlib, Plotly. Это ускоряет создание интерфейсов для взаимодействия с данными и обеспечивает наглядную визуализацию информации в реальном времени.

В рамках работы Streamlit используется для визуализации данных и управления бизнес-логикой через интерфейс, например, для отображения и редактирования информации о поставках, материалах, остатках и статусах. Пользователь может запускать действия, которые взаимодействуют с базой данных через SQLAlchemy.

Особенностью Streamlit является отсутствие необходимости в отдельном веб-сервере. Приложения запускаются с помощью команды streamlit run, что упрощает развертывание и демонстрацию приложения. Для расширения функциональности используются библиотеки, такие как streamlit-option-menu и streamlit-aggrid, позволяющие добавлять панели навигации и интерактивные таблицы.

Streamlit идеально подходит для приложений, где важны скорость разработки и интерактивность. В данном проекте он обеспечил удобную визуализацию данных и взаимодействие между пользователем и системой, что делает его эффективным инструментом для автоматизации и учёта процессов.

Четвертый раздел «Разработка веб-приложения для складского учёта» описывает практическую часть работы — разработку веб-приложения для автоматизации процессов складского учёта на предприятии. Основной целью являлось создание информационной системы, обеспечивающей удобное и надёжное взаимодействие между пользователем и базой данных, охватывающее ключевые бизнес-процессы: закупка материалов, учёт поставок по закупке, продажа со склада, контроль их статусов оплаты и перемещений, контроль остатков на складе.

Разработка велась поэтапно. На первом этапе была определена архитектура проекта, включающая в себя модульную структуру приложения, разделённую на слои: работа с базой данных, модели бизнес-логики, обработчики операций и пользовательский интерфейс.

Далее была спроектирована и реализована структура базы данных на основе PostgreSQL, где определены ключевые сущности. На рисунке 1 представлена схема базы данных, отображающая таблицы и их взаимосвязи.

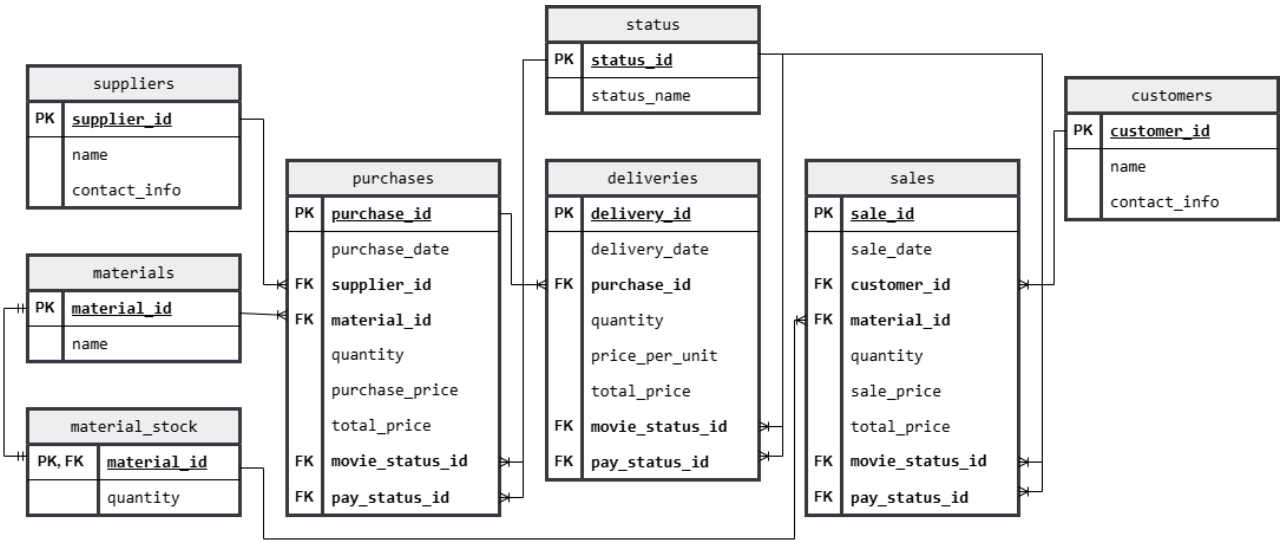

Рисунок 1 – Схема базы данных

После создания базы данных и необходимых таблиц, было выполнено подключение к базе данных из Python-приложения с использованием библиотеки SQLAlchemy. Это позволило реализовать взаимодействие между Python-приложением и СУБД PostgreSQL, включая выполнение запросов, транзакций и обработку данных.

Следующим этапом стало написание ORM-моделей с использованием SQLAlchemy. Эти модели отражают структуру таблиц, определяют связи между ними и включают дополнительные атрибуты и методы, обеспечивающие работу с данными. Например, далее приведена модель для таблицы 'deliveries' – поставки по закупке.

```
class Delivery(Base):
    __tablename__ = 'deliveries'

    delivery_id = Column(Integer, primary_key=True, index=True)
    delivery_date = Column(Date)
    purchase_id = Column(Integer, ForeignKey('purchases.purchase_id'))
    quantity = Column(Numeric, nullable=False)
    price_per_unit = Column(Numeric, nullable=False)
    total_price = Column(
```

```
        Numeric, Computed('quantity * price_per_unit', persisted=True)
    )
    movie_status_id = Column(Integer, ForeignKey('status.status_id'))
    pay_status_id = Column(Integer, ForeignKey('status.status_id'))

    # Связи с другими таблицами
    purchase = relationship("Purchase", back_populates="deliveries")
    movie_status = relationship(
        "Status",
        foreign_keys=[movie_status_id],
        back_populates="movie_status_deliveries"
    )
    pay_status = relationship(
        "Status",
        foreign_keys=[pay_status_id],
        back_populates="pay_status_deliveries"
    )
```

Затем поверх ORM-моделей была реализована бизнес-логика:

— Автоматическое изменение статусов движения и оплаты закупки на основании статусов соответствующих поставок;

— Обновление складских остатков при завершении поставки или продаже;

— Обеспечение надёжности транзакций, контроль целостности данных и повторное использование кода.

Пример одной из таких бизнес-логик — автоматическое обновление статуса движения по закупке. Если общее количество поставленного материала по данной закупке (с поставками в статусе движения 'Completed') достигает или превышает объём, указанный в самой закупке, то статус движения по закупке также автоматически изменяется на 'Completed'.

```
def handle_delivery_logic(session: Session, delivery: Delivery):
    try:
        delivery_status_completed = get_status_id('Completed', session)
        purchase = session.query(Purchase).get(delivery.purchase_id)

        # 1. Проверка завершенности поставки
        total_delivered = session.query(func.sum(Delivery.quantity)).filter_by(
            purchase_id=delivery.purchase_id,
```

```
            movie_status_id=delivery_status_completed
        ).scalar() or 0

        if purchase and total_delivered >= purchase.quantity:
            purchase.movie_status_id = delivery_status_completed

        # Сохраняем изменения по статусам
        session.commit()

    except Exception as e:
        print(f"[ERROR] Ошибка в обработке поставки: {e}")
        session.rollback()
        raise
```

Следующим уровнем реализации стали обработчики операций, представляющие собой функции, управляющие выполнением прикладных действий: созданием, изменением и удалением данных. Эти обработчики взаимодействуют с ORM-моделями и вызывают соответствующую бизнес-логику. Такой подход обеспечивает разделение ответственности и упрощает сопровождение кода. Ниже приведён пример функции добавления записи поставки. После создания записи вызывается логика пересчёта статуса закупки на основании новых данных:

```
# Создание новой поставки
def create_delivery(db: Session, delivery_data: dict):
    delivery_data.pop("total_price", None) # удаляем, так как поле вычисляется
    delivery = Delivery(**delivery_data)
    db.add(delivery)
    db.commit()
    db.refresh(delivery)

    # Вызываем логику после вставки
    handle_delivery_logic(db, delivery)

    return delivery
```

Завершающим этапом разработки стало создание пользовательского интерфейса с использованием библиотеки Streamlit. Интерфейс приложения

организован по принципу разделения на логические секции: «Поставщики», «Материалы», «Клиенты», «Закупки», «Поставки», «Склад» и «Продажи».

Каждая секция включает соответствующие формы для отображения, добавления, обновления и удаления данных. Все действия, выполняемые через интерфейс, обрабатываются с помощью заранее определённых функций, взаимодействующих с базой данных через ORM-модели и бизнес-логику.

Ниже приведён фрагмент кода, реализующий интерфейс добавления новой поставки в разделе Deliveries:

```
def app():
  st.title("Inventory Management")
  db = next(get_db())
  selection = st.radio(
    "Select Section", [
      "Suppliers", "Materials", "Customers",
      "Purchases", "Deliveries",
      "Material Stock", "Sales"])

  if selection == "Deliveries":
    st.subheader("Add New Delivery")

    with st.form(key='add_delivery_form'):
      delivery_date = st.date_input("Delivery Date")
      purchase_id = st.number_input(
        "Purchase ID", min_value=1, step=1)
      quantity = st.number_input(
        "Quantity", min_value=0.0)
      price_per_unit = st.number_input(
        "Price per Unit", min_value=0.0)
      movie_status_id = status_selectbox(
        db, "Select Delivery Status")
      pay_status_id = status_selectbox(
        db, "Select Payment Status")

      submit_button = st.form_submit_button("Add Delivery")

      if submit_button:
        delivery_data = {
          "delivery_date": delivery_date,
```

```
        "purchase_id": purchase_id,
        "quantity": quantity,
        "price_per_unit": price_per_unit,
        "movie_status_id": movie_status_id,
        "pay_status_id": pay_status_id
    }
    new_delivery = create_delivery(db, delivery_data)
    st.success(
        "Delivery for purchase '{}' "
        "added successfully!".format(new_delivery.purchase_id)
    )
```

Основные результаты

В ходе выполнения работы были изучены теоретические основы построения реляционных баз данных, принципы объектно-реляционного отображения и современные инструменты создания веб-интерфейсов. На основе проведённого анализа были выбраны СУБД PostgreSQL, библиотека SQLAlchemy и фреймворк Streamlit как основные средства разработки.

В результате была спроектирована концептуальная модель базы данных и реализована её структура в PostgreSQL с учётом связей между сущностями и обеспечения целостности данных. Разработаны ORM-модели и реализована бизнес-логика обработки операций складского учёта.

Создано веб-приложение, обеспечивающее удобную работу пользователя с системой: регистрацию закупок, поставок и продаж, контроль и обновление их статусов, контроль остатков материалов на складе, просмотр и редактирование данных. Интерфейс интуитивно понятен, включает формы и табличное отображение информации. Разработка завершилась успешным тестированием, подтвердившим работоспособность системы и её соответствие поставленным требованиям.

Программный код приводится в приложениях **A** — **M**, скриншоты в приложении **H** .