

МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»
(РУТ(МИИТ))

Институт управления и цифровых технологий

Кафедра «Цифровые технологии управления транспортными процессами»

КУРСОВОЙ ПРОЕКТ
По дисциплине «Проектирование баз данных»
На тему: «Разработка пользовательского интерфейса в
рамках предметной области Склад»

Группа: УИС-311
Студент: Чибаяев А. Т.,
Преподаватели: Басов А. Р.,
Шейнов Н. Г.

Москва
2024 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
1 ЦЕЛЬ И ЗАДАЧИ.....	4
2 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	6
3 ER-МОДЕЛЬ	8
4 ЗАПРОСЫ НА ЯЗЫКЕ SQL	14
5 СТРУКТУРА ПРОГРАММЫ.....	20
6 ОПИСАНИЕ ИНТЕРФЕЙСА СО СКРИНШОТАМИ.....	25
ЗАКЛЮЧЕНИЕ.....	30
СПИСОК ЛИТЕРАТУРЫ.....	31
ПРИЛОЖЕНИЕ А	32
А.1Создание реляционной модели базы данных и запросов к базе данных 32	
А.2Файл Sklad.py, содержащий класс базы данных и контроллеры	48

ВВЕДЕНИЕ

Современные бизнес-процессы требуют эффективного управления ресурсами и оптимизации логистических операций, особенно в таких сферах, как торговля и складское хранение. Управление складом представляет собой ключевой аспект, влияющий на общую производительность и прибыльность компании. В условиях динамично изменяющегося рынка, когда магазины ежемесячно формируют заявки на товары, важно обеспечить актуальность и доступность информации о наличии продукции, объемах заказов и предстоящих отгрузках.

Данная курсовая работа посвящена разработке реляционной базы данных для предметной области «Склад». Основной целью работы является создание системы, способной обрабатывать и хранить информацию о товарах, их остатках, а также о заявках и отгрузках, что позволит оптимизировать процессы поставок и улучшить сервис для клиентов.

В ходе исследования будут проанализированы сущности и их атрибуты, а также разработаны связи между ними, что обеспечит целостность данных и возможность их быстрого извлечения. Также будут определены основные запросы, направленные на получение актуальной информации о состоянии склада и товарных запасов.

Работа будет выполнена в соответствии с принципами нормализации данных, что позволит избежать избыточности и обеспечить целостность базы данных. Внедрение системы с графическим интерфейсом обеспечит удобный доступ к необходимой информации как для сотрудников склада, так и для административного аппарата, что, в свою очередь, повысит оперативность принятия решений.

Таким образом, данное исследование представляет собой важный шаг к созданию эффективной системы управления складом, что, безусловно, будет способствовать повышению конкурентоспособности и эффективности бизнеса в целом.

1 ЦЕЛЬ И ЗАДАЧИ

Цель работы: выполнить курсовой проект по разработке пользовательского интерфейса в рамках предметной области «Склад».

Задачи:

Отобразить заданную предметную область, описанную вербально, в реляционную БД, отношения которой должны находиться, по крайней мере, в БКНФ. Согласно варианту заданий, организовать операции ввода, редактирования, удаления и просмотра информации из БД посредством реализации пользовательского интерфейса в выбранной по желанию среде разработки (Python, C++, Java и др.). Обеспечить ответ на заданные запросы.

Предметная область: Склад

Исходные данные:

1. Товары
2. Магазины
3. Склад
4. Заказы
5. Накладные

Допущения:

1. В конце каждого месяца магазины присылают на склад заявки на товары на следующий месяц.
2. Магазины могут забирать товары согласно заявке в любое время. При отгрузке товара оформляется накладная.
3. Информация должна быть доступна непосредственно в складе и в административных помещениях оптовой базы. Это необходимо для оперативного доступа к заказам всех магазинов и актуальной информации по текущим отгрузкам, заказам и планированию завоза товаров на склад.

Запросы:

1. Объемы заказов по каждому из товаров на текущий месяц.
2. Остатки товаров на складе.
3. Количество и виды товаров, которые необходимо завезти на склад, чтобы выполнить заказы на следующий месяц.
4. Список товаров, не пользующихся спросом в текущем месяце (не заказаны ни одним из магазинов).
5. Перечень товаров и их количество, входящие в конкретный заказ.

2 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Данная предметная область описывает процесс управления складскими операциями, начиная от получения заявок от магазинов до учета остатков и планирования поставок. Включает аспекты оперативного доступа к информации о товарах и заказах, что позволяет улучшить планирование и контроль товарооборота.

Предметная область "Склад" охватывает процессы управления запасами и логистикой на складе, включая следующие ключевые аспекты:

1. Управление заявками от магазинов:

В конце каждого месяца склады получают от магазинов заявки на товары, которые планируется поставлять в следующем месяце. Эти заявки помогают планировать запасы и предотвращать дефицит или избыток товаров.

Заявки формируются на основе потребностей магазинов, что позволяет эффективно распределять товарные ресурсы и управлять их перемещением на складе.

2. Отгрузка товаров:

Магазины могут забирать заказанные товары со склада в любое удобное время в течение месяца. При этом, для каждого заказа оформляется накладная, которая содержит информацию о наименованиях товаров, количестве, а также данные о получателе.

Склад должен обеспечивать наличие заказанных товаров и оперативно организовывать отгрузку.

3. Информационная доступность и контроль:

Для эффективного выполнения складских операций необходим доступ к актуальной информации о заявках, запасах и отгрузках. Это особенно важно как для складских работников, так и для административного персонала.

Информация должна быть легко доступной, чтобы сотрудники могли оперативно реагировать на запросы магазинов и планировать будущие поставки и отгрузки.

4. Планирование поставок и пополнения запасов:

На основе заявок от магазинов и текущих остатков товаров на складе производится планирование пополнения запасов. Это позволяет избежать ситуации,

когда товаров недостаточно для выполнения заказов на следующий месяц.

Анализ спроса позволяет определить, какие товары необходимо заказать у поставщиков заранее, а какие можно не включать в план поставок, так как они не востребованы.

5. Анализ спроса и остатков:

Для эффективного управления запасами проводится анализ спроса на товары. Если некоторые позиции не заказываются магазинами в текущем месяце, это указывает на их низкий спрос. Такие данные помогают оптимизировать закупки и сократить расходы на хранение.

Также важно учитывать остатки товаров на складе, чтобы своевременно выявлять дефицитные позиции и корректировать планы закупок.

3 ER-МОДЕЛЬ

ER-модель базы данных в нотации IDEFIX была разработана в программе Erwin.

Работа с Erwin Data Modeler включает несколько ключевых шагов, необходимых для создания и редактирования концептуальной модели данных []. Вот более подробное описание процесса работы с этим инструментом:

1. Запуск программы:

Переход в меню "Пуск" -> "Программы" -> "CA" -> "AllFusion" -> "ERwin Data Modeler r7" -> "ERwin Data Modeler r7".

При появлении диалогового окна с подсказками "AllFusion ERwin Data Modeler Tips" нужно нажать на кнопку "Close", чтобы закрыть его.

2. Создание новой модели:

Необходимо открыть меню "File" -> "New" или используя соответствующую иконку на панели инструментов.

3. Добавление сущностей:

Выбор инструмента для создания сущностей. В Erwin Data Modeler это вторая иконка на панели инструментов (обычно изображена в виде прямоугольника или другой фигуры, обозначающей сущность).

Нужно щелкнуть в рабочей области модели, чтобы добавить сущность. В ней можно перемещать и изменять размер сущности по мере необходимости (рисунок 3.1).



Рисунок 3.1 – Создание сущности

4. Настройка сущностей:

Нужно дважды щелкнуть по созданной сущности, чтобы открыть окно свойств.

Следует вводить названия сущности на русском языке, чтобы оно было понятно специалистам предметной области.

На этом этапе атрибуты сущностей на концептуальном уровне не добавляются,

но можно указать общие свойства, такие как описание и бизнес-правила.

5. Создание связей между сущностями:

Для добавления связи нужно выбрать один из инструментов для создания связей на панели инструментов (обычно это одна из трех последних иконок, например, линия или стрелка).

Нужно щелкнуть на сущность, с которой хотите начать связь, и протянуть линию до другой сущности, чтобы создать связь.

Нужно дважды щёлкнуть по созданной связи, чтобы открыть её свойства. Здесь можно указать тип связи (например, один к одному, один ко многим), а также дополнительные характеристики, такие как обязательность и кардинальность.

6. Добавление атрибутов:

Хотя атрибуты обычно не рассматриваются на концептуальном уровне, их можно добавить на этапе детального проектирования.

Дважды щёлкнув на сущность, чтобы открыть её свойства, и перейти на вкладку "Attributes".

Можно добавить новый атрибут, указав его название и тип данных. Например, можно выбрать тип данных из предустановленного списка или задать пользовательский тип данных, как показано на рисунке 3.2.

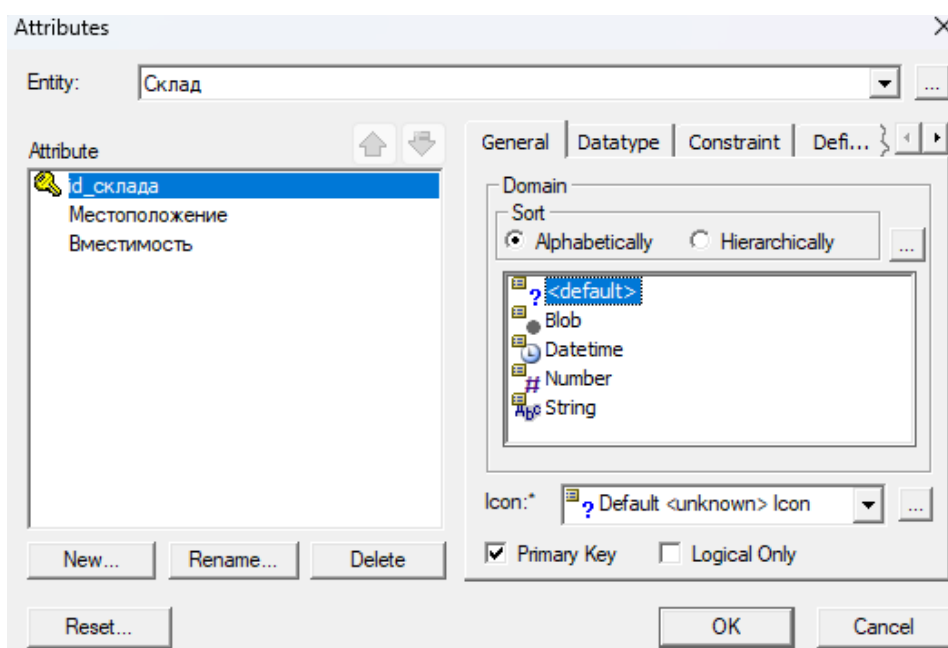


Рисунок 3.2 – Добавление атрибутов

7. Сохранение и экспорт модели:

Сохранить модель нужно через меню "File" -> "Save" или "Save As".

Erwin Data Modeler позволяет экспортировать модель в различные форматы для интеграции с другими инструментами или для создания документации.

На рисунке 3.3 показана разработанная ER-модель базы данных в нотации IDEFIX в предметной области «Склад», созданная при помощи Erwin.

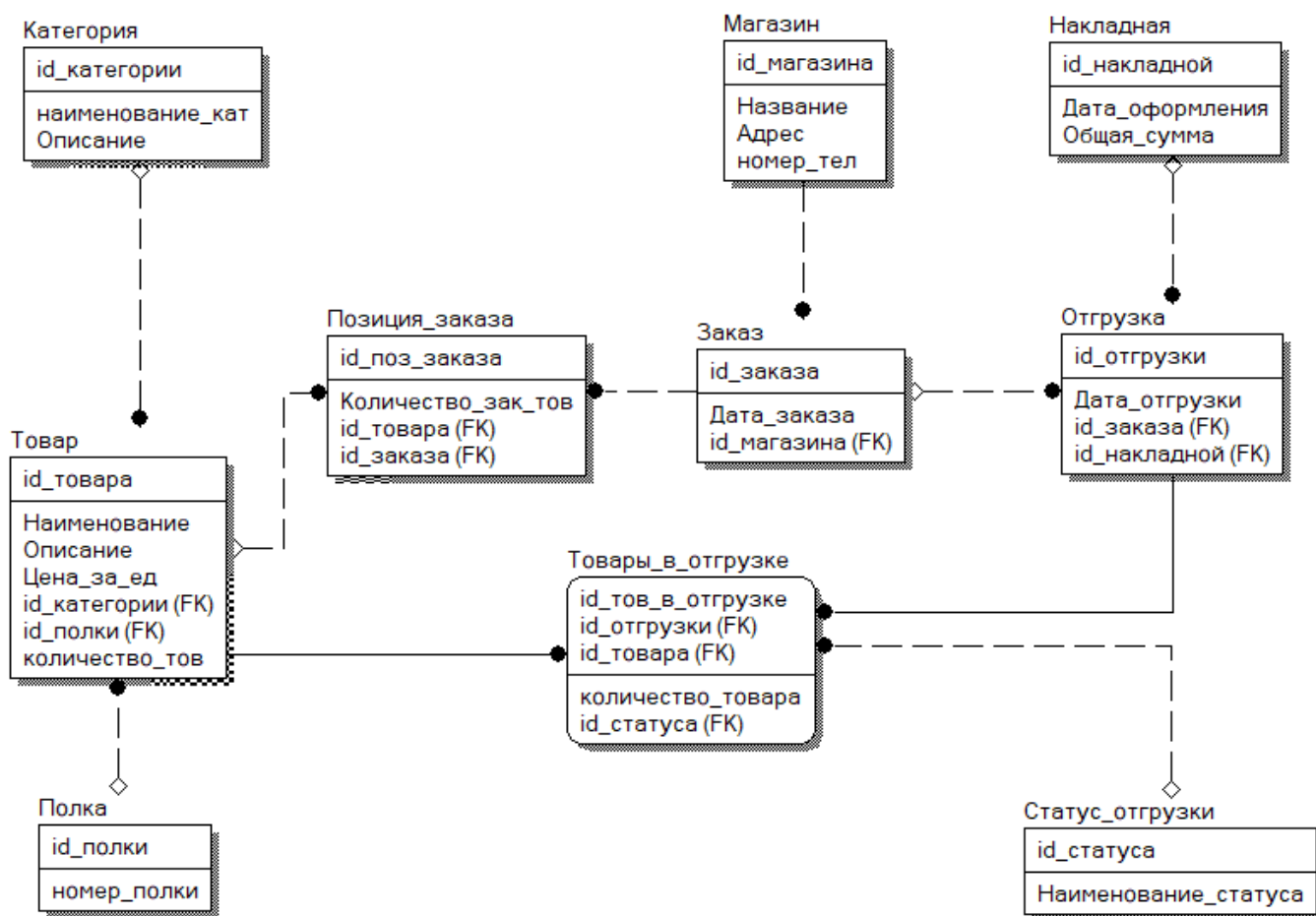


Рисунок 3.3 - ER-модель базы данных в нотации IDEFIX, созданная при помощи Erwin

Описание сущностей и атрибутов в модели:

1. Сущность «Товар» (Атрибуты: id_товара, Наименование, Описание, цена_за_ед, id_категории, id_полки, количество_тов)

Первичный ключ: id_товара

Внешние ключи: id_категории, id_полки

2. Сущность «Магазин» (Атрибуты: id_магазина, Название, Адрес, номер_тел)
Первичный ключ: id_магазина
3. Сущность «Заказ» (Атрибуты: id_заказа, id_магазина, Дата_заказа)
Первичный ключ: id_заказа
Внешний ключ: id_магазина
4. Сущность «Позиция_заказа» (Атрибуты: id_позиции, id_заказа, id_товара, Количество_зак_тов)
Первичный ключ: id_позиции
Внешние ключи: id_заказа, id_товара
5. Сущность «Накладная» (Атрибуты: id_накладной, id_заказа, дата_оформления, Общая_сумма)
Первичный ключ: id_накладной
Внешний ключ: id_заказа
6. Сущность «Категория» (Атрибуты: id_категории, наименование_кат, Описание)
Первичный ключ: id_категории
7. Сущность «Полка» (Атрибуты: id_полки, номер_полки)
Первичный ключ: id_полки
8. Сущность «Отгрузка» (Атрибуты: id_отгрузки, id_заказа, id_накладной, Количество_зак_тов)
Первичный ключ: id_отгрузки
Внешние ключи: id_заказа, id_накладной

9. Сущность «Товары_в_отгрузке» (Атрибуты: id_тов_в_отгрузке, id_отгрузки, id_товара, количество_товара, id_статуса)
Первичный ключ: id_тов_в_отгрузке
Внешние ключи: id_отгрузки, id_товара, id_статуса
10. Сущность «Статус_отгрузки» (Атрибуты: id_статуса, Наименование_статуса)
Первичный ключ: id_статуса

Связи в модели:

1. Магазин – Заказ (1 к М – связь неидентифицирующая)
2. Заказ – Позиция_заказа (1 к М – связь неидентифицирующая)
3. Заказ – Отгрузка (1 к М – связь неидентифицирующая)
4. Накладная – Отгрузка (1 к М – связь неидентифицирующая)
5. Товар – Позиция_заказа (1 к М – связь неидентифицирующая)
6. Категория – Товар (1 к М – связь неидентифицирующая)
7. Полка – Товар (1 к М – связь неидентифицирующая)
8. Товар – Товары_в_отгрузке (1 к М – связь идентифицирующая)
9. Отгрузка – Товары_в_отгрузке (1 к М – связь идентифицирующая)
10. Статус_отгрузки – Товары_в_отгрузке (1 к М – связь неидентифицирующая)

На рисунке 3.4 показана ER-диаграмма, сгенерированная через pgAdmin.

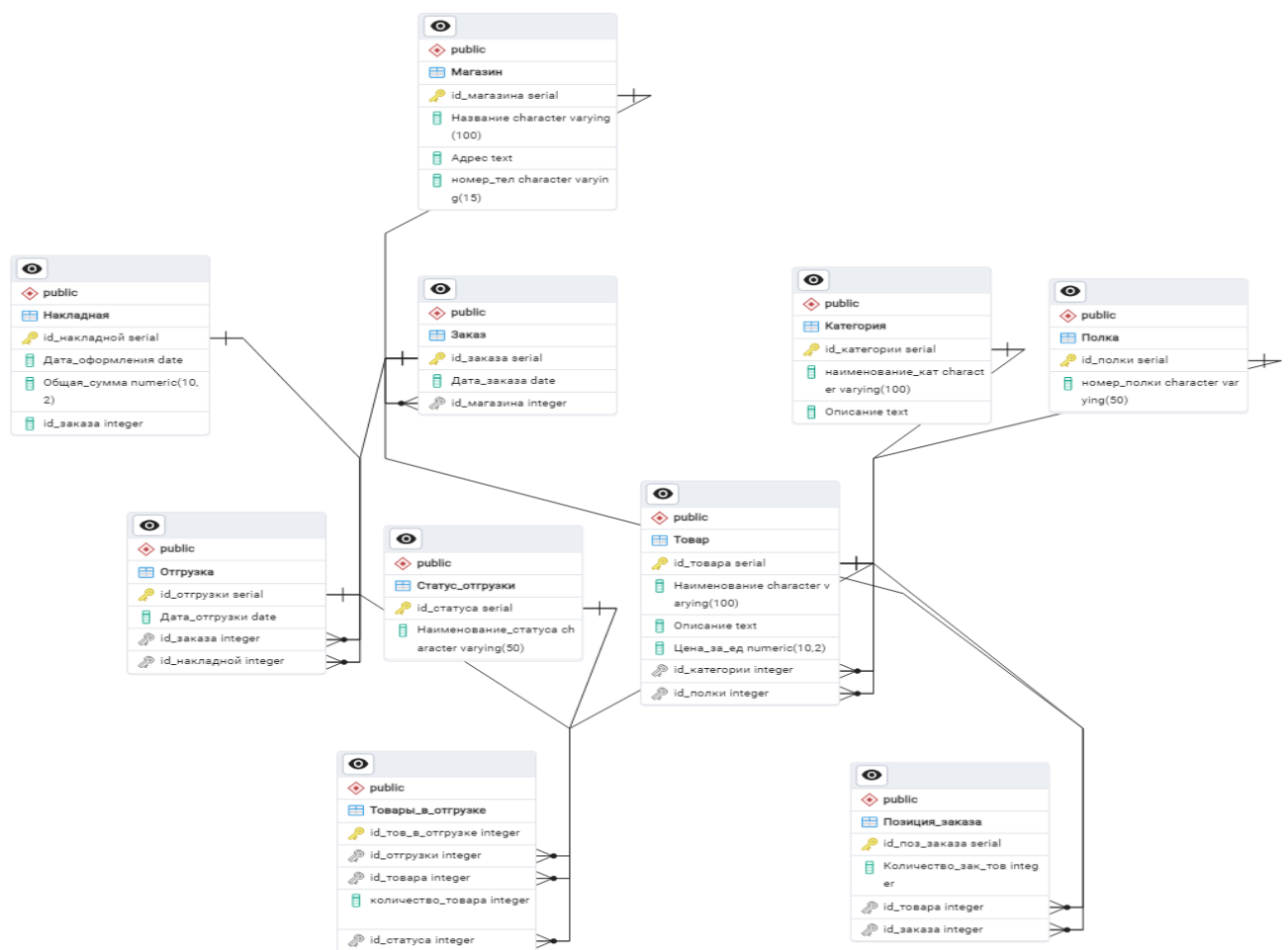


Рисунок 3.4 – ER-модель базы данных в pgAdmin

4 ЗАПРОСЫ НА ЯЗЫКЕ SQL

1. Запрос – Объемы заказов по каждому из товаров на текущий месяц.

SQL запрос:

```
SELECT
    t.id_товара,
    t.Наименование,
    SUM(pz.Количество_зак_тов) AS Общий_объем_заказов
FROM
    Товар t
LEFT JOIN
    Позиция_заказа pz ON t.id_товара = pz.id_товара
LEFT JOIN
    Заказ z ON pz.id_заказа = z.id_заказа
WHERE
    z.Дата_заказа IS NOT NULL
    AND z.Дата_заказа BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY
    t.id_товара, t.Наименование
ORDER BY
    Общий_объем_заказов DESC;
```

На рисунке 4.1 показан ответ на запрос.

	id_товара [PK] integer	Наименование character varying (100)	Общий_объем_заказов bigint
1	9	Ручка	10
2	39	Палатка	9
3	20	Кресло	9
4	8	Сыр	8
5	28	Ластик	8
6	37	Кукла	7
7	19	Настольная лампа	7
8	27	Ежедневник	6
9	16	Колбаса	6
10	7	Джинсы	6
11	34	Шампунь	6
12	31	Шкаф	5
13	23	Куртка	5
14	3	Телевизор	5
15	15	Шоколад	5
16	33	Крем для лица	4
17	13	Кроссовки	4
18	6	Футболка	4
19	4	Холодильник	3
20	11	Миксер	3
21	18	Степлер	3
22	26	Чай	3
23	29	Гантели	3
24	36	Настольная игра	3
25	1	Смартфон	2
26	35	Рюкзак	2
27	25	Кофе	2
28	17	Маркеры	2
29	21	Пылесос	2
30	24	Кепка	1
31	10	Тетрадь	1
32	32	Стул	1
33	14	Часы	1

Рисунок 4.1 – Ответ на первый запрос

2. Запрос – Остатки товаров на складе.

SQL запрос:

```
SELECT
    t.id_товара,
    t.Наименование,
    SUM(pz.Количество_зак_тов) AS Общий_объем_заказов
FROM
    Товар t
JOIN
    Позиция_заказа pz ON t.id_товара = pz.id_товара
JOIN
    Заказ z ON pz.id_заказа = z.id_заказа
WHERE
    TO_CHAR(z.Дата_заказа, 'MM') = '11'
    AND TO_CHAR(z.Дата_заказа, 'YYYY') = TO_CHAR(CURRENT_DATE, 'YYYY')
GROUP BY
    t.id_товара, t.Наименование
ORDER BY
    Общий_объем_заказов DESC;
```

На рисунке 4.2 показан ответ на запрос.

Data Output Messages Notifications			
	id_товара [PK] integer	Наименование character varying (100)	Остаток_на_складе bigint
1	1	Смартфон	18
2	2	Ноутбук	14
3	3	Телевизор	20
4	4	Холодильник	9
5	5	Стиральная машина	10
6	6	Футболка	26
7	7	Джинсы	19
8	8	Сыр	0
9	9	Ручка	40
10	10	Тетрадь	39
11	11	Миксер	2
12	12	Блендер	8
13	13	Кроссовки	16
14	14	Часы	14
15	15	Шоколад	30
16	16	Колбаса	4
17	17	Маркеры	18
18	18	Степлер	12
19	19	Настольная лампа	3
20	20	Кресло	4
21	21	Пылесос	10
22	22	Утюг	8

Рисунок 4.2 – Ответ на второй запрос

3. Запрос – Сколько и каких товаров необходимо завезти на склад, чтобы не поставить под угрозу выполнение заказов на следующий месяц.

SQL запрос:

```
SELECT
    t.id_товара,
    t.Наименование,
    t.количество_тов - COALESCE(tv.количество_товара, 0) AS Остаток_на_складе
FROM
    Товар t
LEFT JOIN
    Товары_в_отгрузке tv ON t.id_товара = tv.id_товара
WHERE
    (t.количество_тов - COALESCE(tv.количество_товара, 0)) < 10
ORDER BY
    t.id_товара;
```

На рисунке 4.3 показан ответ на запрос.

Data Output Messages Notifications			
	id_товара [PK] integer	Наименование character varying (100)	Остаток_на_складе bigint
1	4	Холодильник	9
2	8	Сыр	0
3	11	Миксер	2
4	12	Блендер	8
5	16	Колбаса	4
6	19	Настольная лампа	3
7	20	Кресло	4
8	22	Утюг	8
9	23	Куртка	5
10	26	Чай	7
11	30	Велотренажер	9
12	31	Шкаф	5
13	36	Настольная игра	9
14	37	Кукла	3
15	38	Машинка	8
16	39	Палатка	5

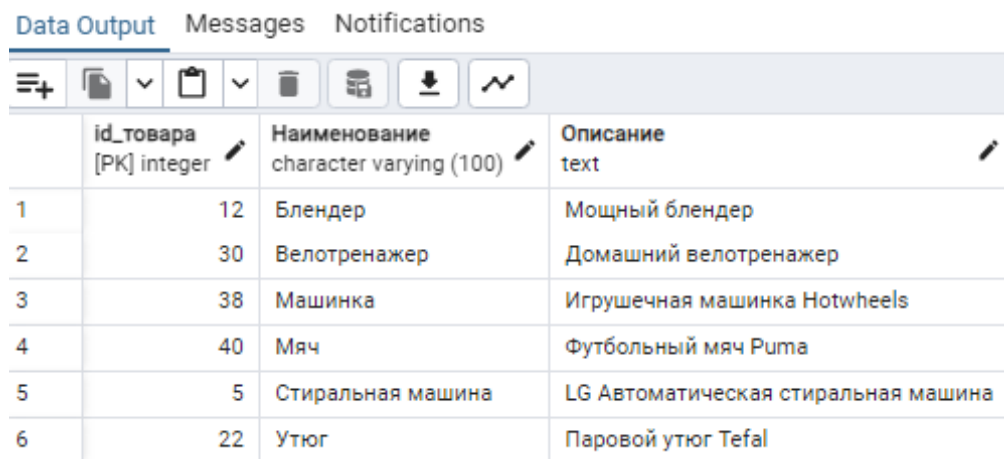
Рисунок 4.3 – Ответ на второй запрос

4. Запрос – Какие товары не пользуются спросом в текущем месяце (Не заказаны ни одним из магазинов).

SQL запрос:

```
SELECT
    t.id_товара,
    t.Наименование,
    t.Описание
FROM
    Товар t
LEFT JOIN
    Позиция_заказа pz ON t.id_товара = pz.id_товара
LEFT JOIN
    Заказ z ON pz.id_заказа = z.id_заказа
WHERE
    z.id_заказа IS NULL
ORDER BY
    t.Наименование;
```

На рисунке 4.4 показан ответ на запрос.



	id_товара [PK] integer	Наименование character varying (100)	Описание text
1	12	Блендер	Мощный блендер
2	30	Велотренажер	Домашний велотренажер
3	38	Машинка	Игрушечная машинка Hotwheels
4	40	Мяч	Футбольный мяч Puma
5	5	Стиральная машина	LG Автоматическая стиральная машина
6	22	Утюг	Паровой утюг Tefal

Рисунок 4.4 – Ответ на четвертый запрос

5. Запрос – Список товаров и количество их, которые входят в определенный заказ.

SQL запрос:

```
SELECT
    t.id_товара,
    t.Наименование,
```

```

        t.Описание,
        COALESCE(SUM(pz.Количество_зак_тов), 0) AS Количество_в_заказах
FROM
    Товар t
LEFT JOIN
    Позиция_заказа pz ON t.id_товара = pz.id_товара
WHERE
    pz.id_заказа = 1
GROUP BY
    t.id_товара, t.Наименование, t.Описание
ORDER BY
    t.id_товара;

```

На рисунке 4.5 показан ответ на запрос по конкретному заказу.

Data Output Messages Notifications				
	id_товара [PK] integer	Наименование character varying (100)	Описание text	Количество_в_заказах bigint
1	1	Смартфон	Iphone 15 Pro Black 256 Gb	2
2	2	Ноутбук	Ноутбук Apple MacBook Air 13 M2 8 core/8 core/8/256/Space Gray (MLXW...	1

Рисунок 4.5 – Ответ на пятый запрос

5 СТРУКТУРА ПРОГРАММЫ

Программа представляет собой графическое приложение для взаимодействия с базой данных PostgreSQL, разработанное с использованием библиотеки tkinter для создания интерфейса и библиотеки psycorg2 для подключения к базе данных. Выбор этих технологий обусловлен их функциональностью, удобством и совместимостью с поставленными задачами.

Tkinter является стандартной библиотекой Python для создания графических интерфейсов и предоставляет все необходимые инструменты для быстрого создания оконных приложений. Библиотека легко интегрируется с Python, что устраняет необходимость установки сторонних пакетов, а также обеспечивает совместимость с различными операционными системами, такими как Windows, macOS и Linux. Tkinter позволяет реализовать функциональные интерфейсы с такими элементами, как окна, кнопки, поля ввода и таблицы, упрощая задачу создания графического интерфейса для приложения. В данном проекте Tkinter используется для построения основного окна приложения, организации формы ввода данных для подключения к базе данных и отображения информации в виде таблиц.

Библиотека psycorg2 является мощным инструментом для работы с базой данных PostgreSQL. Она позволяет эффективно выполнять запросы к базе данных, обеспечивая поддержку транзакций, работы с курсорами и обработки ошибок. Это делает psycorg2 одной из самых популярных библиотек для работы с PostgreSQL в Python, обладающей хорошей документацией и устойчивой производительностью. Выбор этой библиотеки был обусловлен её стабильностью и возможностью работы с большими объемами данных, а также поддержкой всех основных типов запросов, таких как SELECT, INSERT, UPDATE и DELETE, которые являются основой взаимодействия с базой данных.

Основной класс программы, DatabaseApp, отвечает за организацию взаимодействия между пользователем и базой данных. Он содержит методы для подключения к базе данных, выполнения SQL-запросов, а также для отображения результатов в виде таблиц, редактирования строк и добавления новых записей. Каждый элемент интерфейса имеет четкую структуру, и логика работы с каждым

компонентом легко расширяется и модифицируется. Интерактивные элементы, такие как кнопки для выполнения действий с базой данных и поля ввода для параметров подключения, организованы с использованием Tkinter, что позволяет обеспечить простоту взаимодействия с программой.

Реализованные в программе окна для работы с различными сущностями базы данных динамически генерируются на основе метаданных, что делает приложение гибким и масштабируемым. Вкладка для выполнения произвольных SQL-запросов также интегрирована в интерфейс, что позволяет пользователю напрямую взаимодействовать с базой данных через веб-консоль, обеспечивая высокий уровень контроля над данными. Функция добавления и редактирования данных реализована с учетом особенностей работы с внешними ключами и проверки уникальности первичных ключей, что минимизирует вероятность ошибок при вводе данных.

Программа использует PostgreSQL как систему управления базами данных, поскольку она является одной из наиболее мощных и стабильных СУБД с открытым исходным кодом. PostgreSQL поддерживает множество передовых технологий, включая транзакции, полнотекстовый поиск и работу со сложными типами данных. Это обеспечивает высокий уровень надежности и масштабируемости системы, что позволяет применять её в широком спектре задач, от небольших проектов до крупных корпоративных решений.

Таким образом, структура программы основана на простоте и гибкости использования при сохранении высокой производительности и стабильности работы с базой данных. Использование библиотеки Tkinter для создания интерфейса и psycopg2 для работы с PostgreSQL обеспечивает удобство разработки и масштабируемость, что делает данную программу эффективным инструментом для работы с данными в реальных системах.

На рисунке 5.1 представлена диаграмма классов, которая основана на коде приложения для работы с базой данных PostgreSQL.

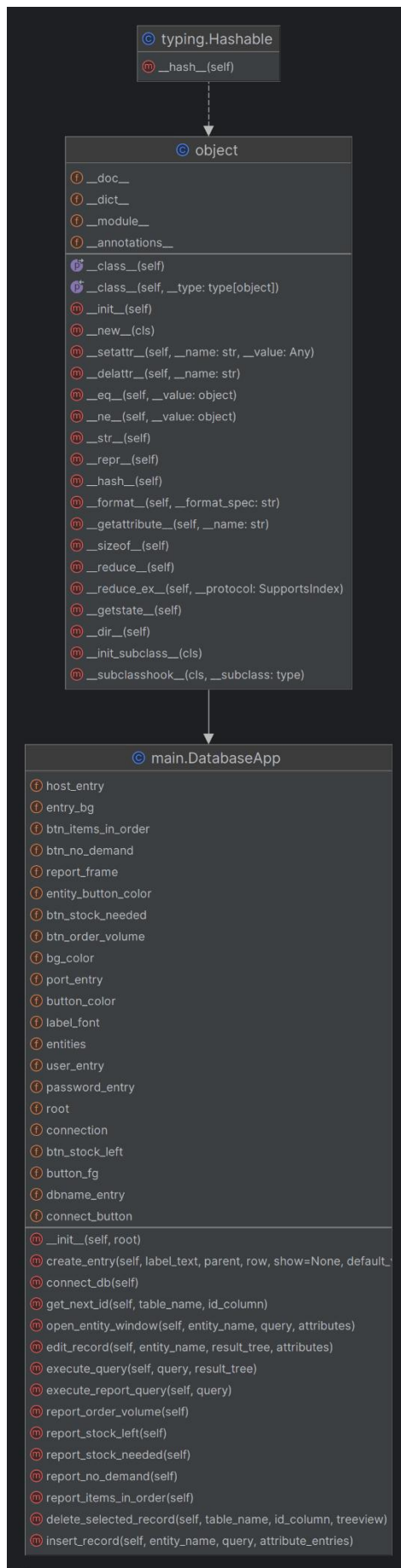


Рисунок 5.1 – Диаграмма классов

Класс DatabaseApp служит основным компонентом приложения для работы с базой данных PostgreSQL через графический интерфейс, построенный с использованием библиотеки Tkinter. Его главная задача — организация удобного и интуитивного взаимодействия с базой данных, что включает отображение, добавление, редактирование и удаление записей, а также формирование различных отчетов с помощью SQL-запросов.

В основе класса лежит атрибут `root`, представляющий главное окно приложения. Все элементы интерфейса располагаются внутри этого окна, которое можно настраивать через такие параметры, как `bg_color`, определяющий фон приложения, `button_color` и `entity_button_color`, отвечающие за окрашивание кнопок, а также `entry_bg`, задающий фон полей ввода. Цвет текста кнопок задается параметром `button_fg`, а шрифт для меток и других элементов интерфейса определяется через `label_font`. Эти настройки позволяют создавать гармоничный и эстетически приятный пользовательский интерфейс, адаптируемый под разные задачи.

Подключение к базе данных осуществляется с помощью метода `connect_db`, который использует библиотеку `psycopg2`. В приложении предусмотрены поля для ввода параметров подключения: `host_entry`, `port_entry`, `dbname_entry`, `user_entry` и `password_entry`, что дает возможность гибко настраивать соединение с базой. Для хранения данных о таблицах и связанных с ними операциях используется атрибут `entities`, представляющий собой словарь, в котором каждая сущность описывает соответствующую таблицу базы данных, её атрибуты и SQL-запросы. Это позволяет централизованно управлять структурами данных и их обработкой, упрощая дальнейшую модификацию или расширение приложения.

Для работы с конкретной таблицей реализован метод `open_entity_window`, который открывает отдельное окно. В этом окне пользователь может просматривать данные таблицы, добавлять новые записи, редактировать существующие и удалять ненужные записи. Метод `insert_record` отвечает за добавление новых данных в таблицу, а удаление записей осуществляется с помощью метода `delete_selected_record`. Для формирования SQL-запросов используется метод

`execute_query`, который отправляет команды в базу данных и отображает результат их выполнения. Визуализация результатов, как и всех других операций, организована в удобной форме, чтобы пользователь мог быстро анализировать данные.

Класс также предоставляет мощные инструменты для создания отчетов. Методы, такие как `report_order_volume`, `report_stock_left`, `report_stock_needed` и `report_no_demand`, выполняют специфические запросы и формируют отчеты, например, по объемам заказов, остаткам товаров на складе, необходимости пополнения запасов и отсутствию спроса на определенные товары. Интерфейс дополнен кнопками, такими как `btn_stock_left`, `btn_stock_needed`, `btn_no_demand` и `btn_order_volume`, что упрощает доступ к функционалу отчетов. Эти элементы взаимодействуют с `report_frame`, который отвечает за отображение отчетов в виде таблиц или других визуальных форм.

Для упрощения работы с интерфейсом реализован метод `create_entry`, который автоматически создает поле ввода с меткой, что особенно полезно при сборе данных от пользователя. Метод `get_next_id` помогает определить следующий уникальный идентификатор для новой записи, основываясь на данных таблицы, что гарантирует целостность и уникальность записей в базе данных.

Архитектура класса продумана так, чтобы обеспечить модульность и легкость расширения. Все элементы интерфейса и методы взаимодействия с базой данных разработаны таким образом, чтобы их можно было модифицировать или дополнять без значительных изменений в структуре приложения. Использование словаря `entities` делает приложение легко адаптируемым к новым требованиям, включая изменение структуры базы данных или добавление новых таблиц.

Таким образом, `DatabaseApp` объединяет функции управления базой данных, гибкий графический интерфейс и простоту использования. Он предоставляет пользователю надежный инструмент для работы с данными, подходящий как для базового администрирования, так и для более сложных аналитических задач.

6 ОПИСАНИЕ ИНТЕРФЕЙСА СО СКРИНШОТАМИ

На рисунке 6.1 изображен экран авторизации программы. На нем представлены поля для ввода параметров подключения к базе данных, таких как хост, порт, имя пользователя, пароль и имя базы данных. Под полями ввода находится кнопка "Вход", которая инициирует попытку подключения к базе данных при введении правильных учетных данных. Ошибки авторизации, как правило, отображаются в виде всплывающего сообщения в случае неверного ввода данных. Также здесь представлены кнопки с отчётами, нажав на которые выведется определённая информация по БД.

Графический интерфейс PostgreSQL

Хост:

Порт:

Имя базы данных:

Пользователь:

Пароль:

Таблицы базы данных 'Склад'

<input type="button" value="Категория"/>	<input type="button" value="Товар"/>
<input type="button" value="Полка"/>	<input type="button" value="Магазин"/>
<input type="button" value="Заказ"/>	<input type="button" value="Позиция_заказа"/>
<input type="button" value="Накладная"/>	<input type="button" value="Отгрузка"/>
<input type="button" value="Товары_в_отгрузке"/>	<input type="button" value="Статус_отгрузки"/>

ОТЧЁТЫ

<input type="button" value="Объемы заказов по каждому товару на текущий месяц"/>
<input type="button" value="Остатки товаров на складе"/>
<input type="button" value="Товары, которые необходимо завезти"/>
<input type="button" value="Товары, не пользующиеся спросом"/>
<input type="button" value="Список товаров в заказах"/>

Рисунок 6.1 – Вход в систему

После ввода данных появляется окно об успешном подключении к базе данных, как показано на рисунке 6.2.

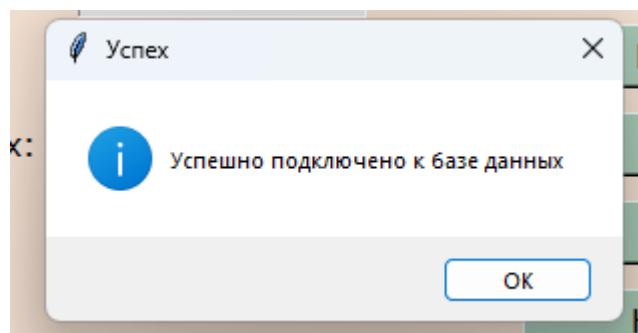


Рисунок 6.2 – Уведомление об успешном подключении к базе данных

На рисунке 6.3 показана демонстрация поведения системы при неверном вводе учетных данных для авторизации. В случае, если хотя бы одно из полей (хост, порт, имя пользователя или пароль) введено некорректно, появляется всплывающее окно с ошибкой, в котором подробно указано сообщение о причине отказа в подключении. Пользователь должен исправить ошибку и повторно попытаться войти.

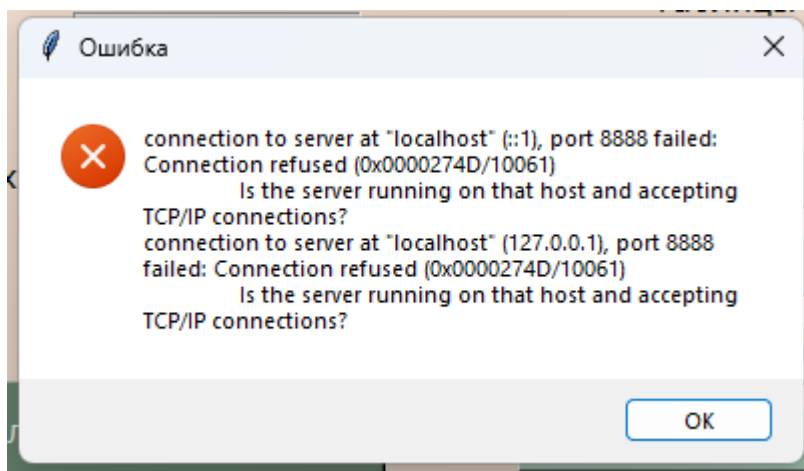


Рисунок 6.3 – Ошибка к подключению к БД

Рисунок 6.4 иллюстрирует окно, в котором отображаются данные выбранной таблицы. После успешной авторизации пользователь может выбрать таблицу для работы, и на экране отобразится список строк данной таблицы. В этом окне видно представление информации по строкам и столбцам, с возможностью сортировки и фильтрации данных. Важно отметить, что интерфейс поддерживает возможность редактирования и удаления записей.

id_магазина	Название	Адрес	номер_тел
1	DNS	ул. Центральная, 1	849547390253
2	M-Video	ул. Ленина, 10	84954068788
3	ZARA	ул. Победы, 25	84950345129
4	Пятерочка	ул. Молодежная, 15	84958567391
5	Комус	ул. Северная, 5	84958937783
6	Детский Мир	ул. Южная, 8	84956767890
7	Спортмастер	ул. Восточная, 12	849512824523
8	Hoff	ул. Западная, 3	849578349454
9	Подружка	ул. Звездная, 9	84957683449
10	Книжный Лабиринт	ул. Парковая, 20	84956723339

id_магазина:

Название:

Адрес:

номер_тел:

Рисунок 6.4 – Окно таблицы

На рисунке 6.5 показан список всех строк выбранной таблицы. Каждая строка представлена в виде набора атрибутов (полей), которые являются частью структуры данной таблицы. Визуально пользователю предоставляется доступ к просмотру всех данных, при этом реализована возможность их выбора для дальнейшего добавления.

id_магазина:

Название:

Адрес:

номер_тел:

Рисунок 6.5 – Список всех строк в таблице

На рисунке 6.6 показано, как система реагирует, если все было заполнено правильно.

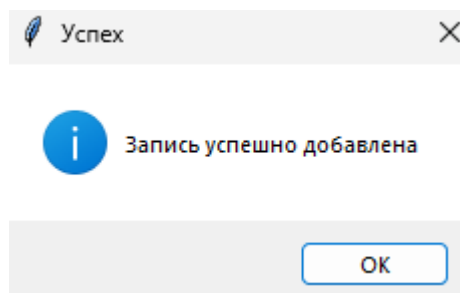


Рисунок 6.6 – Успешное добавление записи в таблицу

На рисунке 6.7 показано, как система реагирует на ошибку при попытке добавления новой строки. Если в процессе заполнения формы были допущены ошибки (например, нарушение уникальности первичного ключа или введение неверных значений), появляется соответствующее сообщение об ошибке. Этот рисунок помогает понять, как программа сообщает пользователю о причинах неудачи и что нужно исправить.

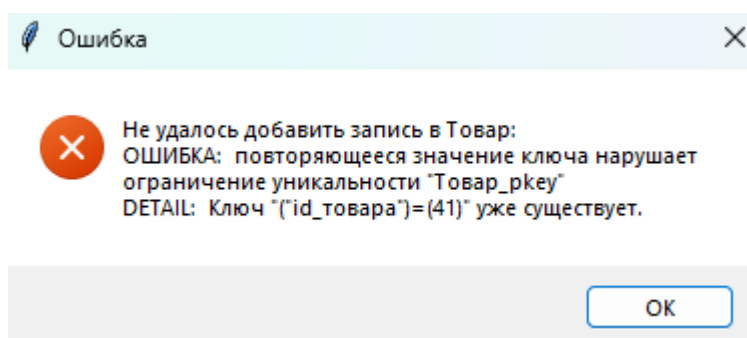


Рисунок 6.7 – Ошибка при добавлении записи в таблицу

На рисунке 6.8 представлено редактирование и удаление записей в базе данных, которое происходит следующим образом: пользователь выбирает запись в интерфейсе, для редактирования отображаются текущие данные, которые можно изменить. После ввода новых значений формируется SQL-запрос UPDATE, который обновляет запись в базе данных, и изменения отображаются в интерфейсе. Для удаления записи пользователь выбирает её, подтверждает удаление, и система выполняет SQL-запрос DELETE, удаляя запись из базы данных и обновляя интерфейс.

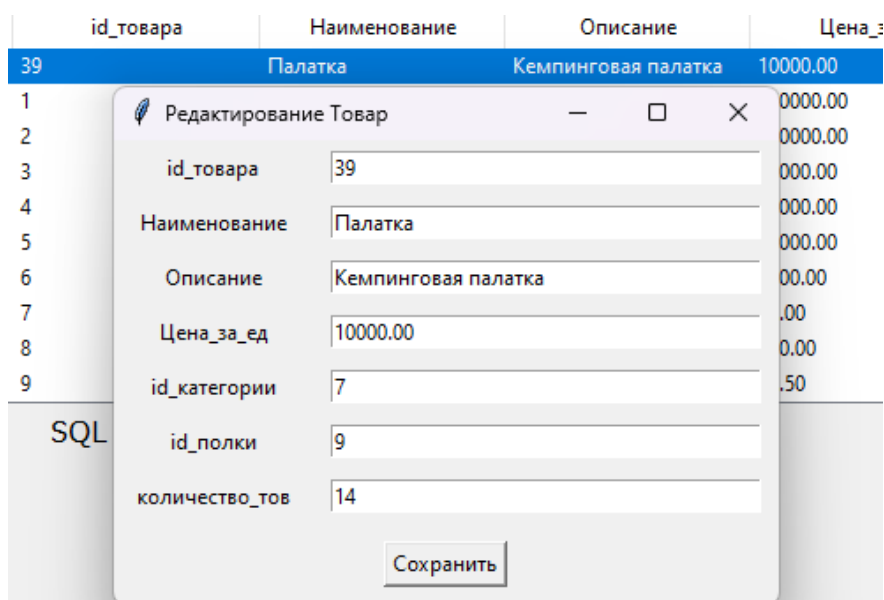


Рисунок 6.8 – Редактирование данных в записи

При нажатии на кнопку отчёта система выполняет соответствующий SQL-запрос, извлекающий необходимые данные из базы, как изображено на рисунке 6.9. После этого результаты запроса отображаются в интерфейсе в виде таблицы или другого подходящего формата. Это позволяет пользователю быстро просматривать актуальную информацию по заданному отчету.

Отчет		
id_товара	Наименование	Описание
12	Блендер	Мощный блендер
30	Велотренажер	Домашний велотренажер
38	Машинка	Игрушечная машинка Hotwheels
40	Мяч	Футбольный мяч Puma
5	Стиральная машина	LG Автоматическая стиральная машина
22	Утюг	Паровой утюг Tefal

Рисунок 6.9 – Пример вывода данных при нажатии на одну из кнопок отчёта

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана система управления складом на основе реляционной базы данных и пользовательского интерфейса. В рамках работы удалось создать модель данных, отражающую ключевые процессы на складе, такие как учет товаров, управление остатками, а также обработка заявок и отгрузок. Разработанный интерфейс, интегрированный с базой данных, обеспечивает удобный доступ к необходимой информации для пользователей и способствует улучшению процесса принятия решений.

Особое внимание было уделено нормализации данных, что позволило избежать избыточности информации и обеспечило целостность базы. Использование графического интерфейса позволило сделать систему интуитивно понятной и доступной для пользователей с различным уровнем подготовки, а также повысить скорость выполнения операций на складе.

Процесс разработки также включал создание SQL-запросов для получения актуальной информации, что позволяет оперативно отслеживать состояние склада и взаимодействовать с данными. В результате, созданная система демонстрирует высокий уровень интеграции и автоматизации работы склада, а также делает взаимодействие с пользователем эффективным и удобным.

Предложенная система управления складом является мощным инструментом для оптимизации логистических операций, повышения оперативности учета и улучшения качества обслуживания клиентов. Внедрение подобной системы может существенно повысить эффективность работы торговых и складских компаний, обеспечивая им конкурентные преимущества на рынке.

СПИСОК ЛИТЕРАТУРЫ

1. Кузнецов, С. Д. «Основы баз данных». М.: Бином. Лаборатория знаний, Интернет-университет информационных технологий, 2017.
2. Голицына, О. Л. «Базы данных» / О. Л. Голицына, Н. В. Максимов, И. И. Попов. М.: Форум, Инфра-М, 2020.
3. Кумскова, И. А. «Базы данных». М.: КноРус, 2021.
4. Латыпова, Р. Р. «Базы данных. Курс лекций». Москва: Гостехиздат, 2018.
5. Стружкин, Н. П. «Базы данных. Проектирование. Учебник» / Н.П. Стружкин, В.В. Годин. М.: Юрайт, 2022.
6. Агальцов, В. П. «Базы данных» (+ CD-ROM). М.: Мир, 2022.
7. Арсеньев, Б. П. «Интеграция распределённых баз данных» / Б. П. Арсеньев, С. А. Яковлев. Москва: Гостехиздат, 2019.
8. Гринченко, Н. Н. «Проектирование баз данных. СУБД Microsoft Access: Учебное пособие для вузов» / Н. Н. Гринченко и др. М.: РиС, 2013.
9. Лукин, В. Н. «Введение в проектирование баз данных». М.: Вузовская книга, 2015.
10. Малыхина, М. П. «Базы данных: основы, проектирование, использование». СПб.: BHV, 2007.
11. Мартишин, С. А. «Проектирование и реализация баз данных в СУБД MySQL с использованием MySQL Workbench: Методы и средства проектирования информационных систем и технологий» / С.А. Мартишин, В. Л. Симонов, М. В. Храпченко. М.: Форум, 2017.
12. Мюллер, Р. Д. «Проектирование баз данных и UML» / Р. Д. Мюллер; Пер. с англ. Е. Н. Молодцова. М.: Лори, 2013.

ПРИЛОЖЕНИЕ А

А.1 Создание реляционной модели базы данных и запросов к базе данных

CREATE запрос:

```
CREATE TABLE Магазин (  
id_магазина SERIAL PRIMARY KEY,  
Название VARCHAR(100) NOT NULL,  
Адрес TEXT NOT NULL,  
номер_тел VARCHAR(15)  
);
```

INSERT запрос:

```
INSERT INTO Магазин (id_магазина, Название, Адрес, номер_тел)  
VALUES  
(1, 'DNS', 'ул. Центральная, 1', '849547390253'),  
(2, 'М-Video', 'ул. Ленина, 10', '84954068788'),  
(3, 'ZARA', 'ул. Победы, 25', '84950345129'),  
(4, 'Пятерочка', 'ул. Молодежная, 15', '84958567391'),  
(5, 'Комус', 'ул. Северная, 5', '84958937783'),  
(6, 'Детский Мир', 'ул. Южная, 8', '84956767890'),  
(7, 'Спортмастер', 'ул. Восточная, 12', '849512824523'),  
(8, 'Hoff', 'ул. Западная, 3', '849578349454'),  
(9, 'Подружка', 'ул. Звездная, 9', '84957683449'),  
(10, 'Книжный Лабиринт', 'ул. Парковая, 20', '84956723339');
```

Наполнение таблицы «Магазин» изображено на рисунке 4.1.

Query

Query History

1

SELECT * FROM public."Магазин"

2

ORDER BY "id_магазина" ASC

Data Output

Messages

Notifications

≡

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

	id_магазина [PK] integer	Название character varying (100)	Адрес text	номер_тел character varying (15)
1	1	DNS	ул. Центральная, 1	849547390253
2	2	M-Video	ул. Ленина, 10	84954068788
3	3	ZARA	ул. Победы, 25	84950345129
4	4	Пятерочка	ул. Молодежная, 15	84958567391
5	5	Комус	ул. Северная, 5	84958937783
6	6	Детский Мир	ул. Южная, 8	84956767890
7	7	Спортмастер	ул. Восточная, 12	849512824523
8	8	Hoff	ул. Западная, 3	849578349454
9	9	Подружка	ул. Звездная, 9	84957683449
10	10	Книжный Лабиринт	ул. Парковая, 20	84956723339

Рисунок 4.1 – Таблица «Магазин»

CREATE запрос:

```
CREATE TABLE Заказ (
    id_заказа SERIAL PRIMARY KEY,
    Дата_заказа DATE NOT NULL,
    id_магазина INT REFERENCES Магазин(id_магазина)
);
```

INSERT запрос:

```
INSERT INTO Заказ (Дата_заказа, id_магазина)
VALUES
(1, '2024-11-01', 1),
(2, '2024-11-02', 2),
(3, '2024-11-03', 3),
(4, '2024-11-04', 4),
(5, '2024-11-05', 5),
(6, '2024-11-06', 6),
(7, '2024-11-07', 7),
(8, '2024-11-08', 8),
(9, '2024-11-09', 9),
(10, '2024-11-10', 10),
(11, '2024-11-11', 1),
```

```
(12, '2024-11-12', 2),
(13, '2024-11-13', 3),
(14, '2024-11-14', 4),
(15, '2024-11-15', 5),
(16, '2024-11-16', 6),
(17, '2024-11-17', 7),
(18, '2024-11-18', 8),
(19, '2024-11-19', 9),
(20, '2024-11-20', 10),
(21, '2024-11-21', 1),
(22, '2024-11-22', 2),
(23, '2024-11-23', 3);
```

Наполнение таблицы «Заказ» изображено на рисунке 4.2.

Query Query History

```
1 SELECT * FROM public."Заказ"
2 ORDER BY "id_заказа" ASC
```

Data Output Messages Notifications

	id_заказа [PK] integer	Дата_заказа date	id_магазина integer
1	1	2024-11-01	1
2	2	2024-11-02	2
3	3	2024-11-03	3
4	4	2024-11-04	4
5	5	2024-11-05	5
6	6	2024-11-06	6
7	7	2024-11-07	7
8	8	2024-11-08	8
9	9	2024-11-09	9
10	10	2024-11-10	10
11	11	2024-11-11	1
12	12	2024-11-12	2
13	13	2024-11-13	3
14	14	2024-11-14	4
15	15	2024-11-15	5
16	16	2024-11-16	6
17	17	2024-11-17	7
18	18	2024-11-18	8
19	19	2024-11-19	9
20	20	2024-11-20	10
21	21	2024-11-21	1
22	22	2024-11-22	2
23	23	2024-11-23	3

Рисунок 4.2 – Таблица «Заказ»

CREATE запрос:

```
CREATE TABLE Отгрузка (  
    id_отгрузки SERIAL PRIMARY KEY,  
    Дата_отгрузки DATE NOT NULL,  
    id_заказа INT REFERENCES Заказ(id_заказа),  
    id_накладной INT REFERENCES Накладная(id_накладной)  
);
```

INSERT запрос:

```
INSERT INTO Отгрузка (id_отгрузки, id_заказа, id_накладной, Дата_отгрузки)  
VALUES  
(1, 1, 1, '2024-11-02'),  
(2, 2, 2, '2024-11-03'),  
(3, 3, 3, '2024-11-04'),  
(4, 4, 4, '2024-11-05'),  
(5, 5, 5, '2024-11-06'),  
(6, 6, 6, '2024-11-07'),  
(7, 7, 7, '2024-11-08'),  
(8, 8, 8, '2024-11-09'),  
(9, 9, 9, '2024-11-10'),  
(10, 10, 10, '2024-11-11'),  
(11, 11, 11, '2024-11-12'),  
(12, 12, 12, '2024-11-13'),  
(13, 13, 13, '2024-11-14'),  
(14, 14, 14, '2024-11-15'),  
(15, 15, 15, '2024-11-16'),  
(16, 16, 16, '2024-11-17'),  
(17, 17, 17, '2024-11-18'),  
(18, 18, 18, '2024-11-19'),  
(19, 19, 19, '2024-11-20'),  
(20, 20, 20, '2024-11-21'),  
(21, 21, 21, '2024-11-22'),  
(22, 22, 22, '2024-11-23'),  
(23, 23, 23, '2024-11-24');
```

Наполнение таблицы «Отгрузка» изображено на рисунке 4.3.

Query Query History

```
1 SELECT * FROM public."Отгрузка"  
2 ORDER BY "id_отгрузки" ASC
```

Data Output Messages Notifications

	id_отгрузки [PK] integer	Дата_отгрузки date	id_заказа integer	id_накладной integer
1	1	2024-11-02	1	1
2	2	2024-11-03	2	2
3	3	2024-11-04	3	3
4	4	2024-11-05	4	4
5	5	2024-11-06	5	5
6	6	2024-11-07	6	6
7	7	2024-11-08	7	7
8	8	2024-11-09	8	8
9	9	2024-11-10	9	9
10	10	2024-11-11	10	10
11	11	2024-11-12	11	11
12	12	2024-11-13	12	12
13	13	2024-11-14	13	13
14	14	2024-11-15	14	14
15	15	2024-11-16	15	15
16	16	2024-11-17	16	16
17	17	2024-11-18	17	17
18	18	2024-11-19	18	18
19	19	2024-11-20	19	19
20	20	2024-11-21	20	20
21	21	2024-11-22	21	21
22	22	2024-11-23	22	22
23	23	2024-11-24	23	23

Рисунок 4.3 – Таблица «Отгрузка»

CREATE запрос:

```
CREATE TABLE Накладная (  
    id_накладной SERIAL PRIMARY KEY,  
    Дата_оформления DATE NOT NULL,  
    Общая_сумма NUMERIC(10, 2),  
    id_заказа INT REFERENCES Заказ(id_заказа)  
);
```

INSERT запрос:

```
INSERT INTO Накладная (id_накладной, id_заказа, Дата_оформления, Общая_сумма)  
VALUES  
(1, 1, '2024-11-01', 110000 * 2 + 120000 * 1), -- Смартфон (2 шт) + Ноутбук (1 шт)  
(2, 2, '2024-11-02', 80000 * 5), -- Телевизор (5 шт)  
(3, 3, '2024-11-03', 70000 * 3), -- Холодильник (3 шт)  
(4, 4, '2024-11-04', 30000 * 3), -- Стиральная машина (3 шт)  
(5, 5, '2024-11-05', 2000 * 4), -- Футболка (4 шт)  
(6, 6, '2024-11-06', 50 * 6), -- Джинсы (6 шт)  
(7, 7, '2024-11-07', 350 * 8), -- Сыр (8 шт)  
(8, 8, '2024-11-08', 11.50 * 10), -- Ручка (10 шт)  
(9, 9, '2024-11-09', 20 * 1), -- Тетрадь (1 шт)  
(10, 10, '2024-11-10', 4500 * 3), -- Миксер (3 шт)  
(11, 11, '2024-11-11', 12000 * 4 + 150000 * 1), -- Кроссовки (4 шт) + Часы (1 шт)  
(12, 12, '2024-11-12', 90.50 * 5 + 230 * 6), -- Шоколад (5 шт) + Колбаса (6 шт)  
(13, 13, '2024-11-13', 150 * 2 + 120 * 3), -- Маркеры (2 шт) + Степлер (3 шт)  
(14, 14, '2024-11-14', 3000 * 7 + 18000 * 9), -- Настольная лампа (7 шт) + Кресло (9  
шт)  
(15, 15, '2024-11-15', 17050 * 2), -- Пылесос (2 шт)  
(16, 16, '2024-11-16', 3500 * 5 + 12000 * 1), -- Утюг (5 шт) + Куртка (1 шт)  
(17, 17, '2024-11-17', 250 * 2 + 55.50 * 3), -- Кофе (2 шт) + Чай (3 шт)  
(18, 18, '2024-11-18', 2500 * 6 + 10.50 * 8), -- Ежедневник (6 шт) + Ластик (8 шт)  
(19, 19, '2024-11-19', 400 * 3), -- Гантели (3 шт)  
(20, 20, '2024-11-20', 35000 * 5 + 30000 * 1), -- Велотренажер (5 шт) + Шкаф (1 шт)  
(21, 21, '2024-11-21', 500 * 4 + 200 * 6), -- Стул (4 шт) + Крем для лица (6 шт)  
(22, 22, '2024-11-22', 890 * 2 + 3500 * 3), -- Шампунь (2 шт) + Рюкзак (3 шт)  
(23, 23, '2024-11-23', 250 * 7 + 150 * 9); -- Настольная игра (7 шт) + Кукла (9 шт)
```

Наполнение таблицы «Накладная» изображено на рисунке 4.4.

Query

Query History

1

▼

SELECT * FROM public."Накладная"

2

ORDER BY "id_накладной" ASC

Data Output

Messages

Notifications

☰+

📄

▼

📋

▼

🗑

🗄

⬇

📈

	id_накладной [PK] integer	Дата_оформления date	Общая_сумма numeric (10,2)	id_заказа integer
1	1	2024-11-01	340000.00	1
2	2	2024-11-02	400000.00	2
3	3	2024-11-03	210000.00	3
4	4	2024-11-04	90000.00	4
5	5	2024-11-05	8000.00	5
6	6	2024-11-06	300.00	6
7	7	2024-11-07	2800.00	7
8	8	2024-11-08	115.00	8
9	9	2024-11-09	20.00	9
10	10	2024-11-10	13500.00	10
11	11	2024-11-11	198000.00	11
12	12	2024-11-12	1832.50	12
13	13	2024-11-13	660.00	13
14	14	2024-11-14	183000.00	14
15	15	2024-11-15	34100.00	15
16	16	2024-11-16	29500.00	16
17	17	2024-11-17	666.50	17
18	18	2024-11-18	15084.00	18
19	19	2024-11-19	1200.00	19
20	20	2024-11-20	205000.00	20
21	21	2024-11-21	3200.00	21
22	22	2024-11-22	12280.00	22
23	23	2024-11-23	3100.00	23

Рисунок 4.4 – Таблица «Накладная»

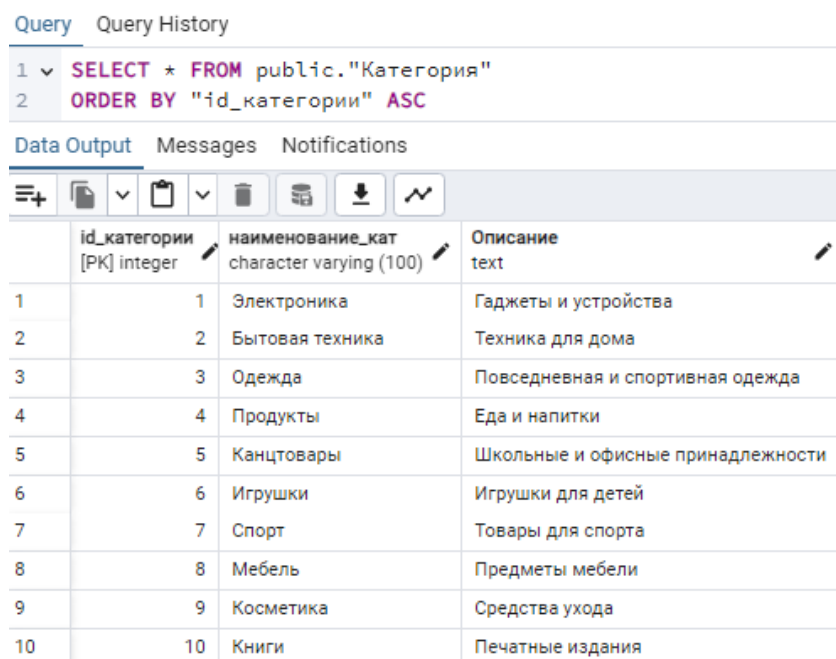
CREATE запрос:

```
CREATE TABLE Категория (  
    id_категории SERIAL PRIMARY KEY,  
    наименование_кат VARCHAR(100) NOT NULL,  
    Описание TEXT  
);
```

INSERT запрос:

```
INSERT INTO Категория (id_категории, наименование_кат, Описание)  
VALUES  
(1, 'Электроника', 'Гаджеты и устройства'),  
(2, 'Бытовая техника', 'Техника для дома'),  
(3, 'Одежда', 'Повседневная и спортивная одежда'),  
(4, 'Продукты', 'Еда и напитки'),  
(5, 'Канцтовары', 'Школьные и офисные принадлежности'),  
(6, 'Игрушки', 'Игрушки для детей'),  
(7, 'Спорт', 'Товары для спорта'),  
(8, 'Мебель', 'Предметы мебели'),  
(9, 'Косметика', 'Средства ухода'),  
(10, 'Книги', 'Печатные издания');
```

Наполнение таблицы «Категория» изображено на рисунке 4.4.



Query Query History

```
1 SELECT * FROM public."Категория"  
2 ORDER BY "id_категории" ASC
```

Data Output Messages Notifications

	id_категории [PK] integer	наименование_кат character varying (100)	Описание text
1	1	Электроника	Гаджеты и устройства
2	2	Бытовая техника	Техника для дома
3	3	Одежда	Повседневная и спортивная одежда
4	4	Продукты	Еда и напитки
5	5	Канцтовары	Школьные и офисные принадлежности
6	6	Игрушки	Игрушки для детей
7	7	Спорт	Товары для спорта
8	8	Мебель	Предметы мебели
9	9	Косметика	Средства ухода
10	10	Книги	Печатные издания

Рисунок 4.4 – Таблица «Категория»

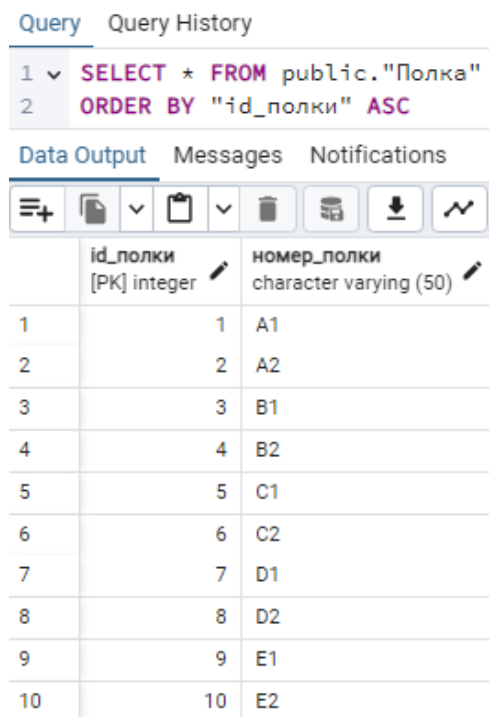
CREATE запрос:

```
CREATE TABLE Полка (  
    id_полки SERIAL PRIMARY KEY,  
    номер_полки VARCHAR(50) NOT NULL  
);
```

INSERT запрос:

```
INSERT INTO Полка (id_полки, номер_полки)  
VALUES  
(1, 'A1'),  
(2, 'A2'),  
(3, 'B1'),  
(4, 'B2'),  
(5, 'C1'),  
(6, 'C2'),  
(7, 'D1'),  
(8, 'D2'),  
(9, 'E1'),  
(10, 'E2');
```

Наполнение таблицы «Полка» изображено на рисунке 4.6.



Query Query History

```
1 SELECT * FROM public."Полка"  
2 ORDER BY "id_полки" ASC
```

Data Output Messages Notifications

	id_полки [PK] integer	номер_полки character varying (50)
1	1	A1
2	2	A2
3	3	B1
4	4	B2
5	5	C1
6	6	C2
7	7	D1
8	8	D2
9	9	E1
10	10	E2

Рисунок 4.6 – Таблица «Полка»

CREATE запрос:

```
CREATE TABLE Товар (  
    id_товара SERIAL PRIMARY KEY,  
    Наименование VARCHAR(100) NOT NULL,  
    Описание TEXT,  
    Цена_за_ед NUMERIC(10, 2) NOT NULL,  
    id_категории INT REFERENCES Категория(id_категории),  
    id_полки INT REFERENCES Полка(id_полки)  
);
```

INSERT запрос:

```
INSERT INTO Товар (id_товара, Наименование, Описание, Цена_за_ед, id_категории,  
id_полки, Количество_тов)  
VALUES  
(1, 'Смартфон', 'Iphone 15 Pro Black 256 Gb', 110000.00, 1, 1, 20),  
(2, 'Ноутбук', 'Ноутбук Apple MacBook Air 13 M2 8 core/8 core/8/256/Space Gray  
(MLXW3)', 120000.00, 1, 2, 15),  
(3, 'Телевизор', 'Samsung QLED 4K Smart TV', 80000.00, 1, 3, 25),  
(4, 'Холодильник', 'Двухкамерный холодильник Haier', 70000.00, 2, 4, 12),  
(5, 'Стиральная машина', 'LG Автоматическая стиральная машина', 30000.00, 2, 5,  
10),  
(6, 'Футболка', 'Хлопковая футболка черная', 2000.00, 3, 6, 30),  
(7, 'Джинсы', 'Синие джинсы Calvin Klein', 50.00, 3, 7, 25),  
(8, 'Сыр', 'Сыр Рокишкио', 350.00, 4, 8, 8),  
(9, 'Ручка', 'Гелевая ручка', 11.50, 5, 9, 50),  
(10, 'Тетрадь', 'Школьная тетрадь в клеточку', 20.00, 5, 10, 40),  
(11, 'Миксер', 'Кухонный миксер Tefal', 4500.00, 2, 1, 5),  
(12, 'Блендер', 'Мощный блендер', 6000.00, 2, 2, 8),  
(13, 'Кроссовки', 'Nike Airforce 1', 12000.00, 3, 3, 20),  
(14, 'Часы', 'Наручные часы Rolex', 150000.00, 3, 4, 15),  
(15, 'Шоколад', 'Молочный шоколад Milka', 90.50, 4, 5, 35),  
(16, 'Колбаса', 'Копченая колбаса', 230.00, 4, 6, 10),  
(17, 'Маркеры', 'Набор цветных маркеров', 150.00, 5, 7, 20),  
(18, 'Степлер', 'Настольный степлер', 120.00, 5, 8, 15),  
(19, 'Настольная лампа', 'Светодиодная настольная лампа Xiaomi', 3000.00, 8, 9,  
10),  
(20, 'Кресло', 'Компьютерное кресло Zone 51', 18000.00, 8, 10, 5),  
(21, 'Пылесос', 'Умный пылесос Xiaomi', 17050.00, 2, 1, 12),  
(22, 'Утюг', 'Паровой утюг Tefal', 3500.00, 2, 2, 8),  
(23, 'Куртка', 'Тёплая зимняя куртка Stone Island', 12000.00, 3, 3, 10),  
(24, 'Кепка', 'Летняя кепка', 1500.00, 3, 4, 20),  
(25, 'Кофе', 'Молотый кофе', 250.00, 4, 5, 15),
```

```
(26, 'Чай', 'Зеленый чай Greenfield', 55.50, 4, 6, 10),
(27, 'Ежедневник', 'Кожанный ежедневник', 2500.00, 5, 7, 25),
(28, 'Ластик', 'Школьный ластик', 10.50, 5, 8, 30),
(29, 'Гантели', 'Гантели 5 кг', 400.00, 7, 9, 15),
(30, 'Велотренажер', 'Домашний велотренажер', 35000.00, 7, 10, 8),
(31, 'Шкаф', 'Двухдверный шкаф', 30000.00, 8, 1, 10),
(32, 'Стул', 'Кухонный стул', 500.00, 8, 2, 20),
(33, 'Крем для лица', 'Увлажняющий крем', 200.00, 9, 3, 26),
(34, 'Шампунь', 'Для всех типов волос System4', 890.00, 9, 4, 16),
(35, 'Рюкзак', 'Школьный рюкзак', 3500.00, 3, 5, 18),
(36, 'Настольная игра', 'Игра для всей семьи', 250.00, 6, 6, 12),
(37, 'Кукла', 'Игрушечная кукла', 150.00, 6, 7, 10),
(38, 'Машинка', 'Игрушечная машинка Hotwheels', 120.00, 6, 8, 8),
(39, 'Палатка', 'Кемпинговая палатка', 10000.00, 7, 9, 6),
(40, 'Мяч', 'Футбольный мяч Puma', 2000.00, 7, 10, 16);
```

Наполнение таблицы «Товар» изображено на рисунке 4.7.

Query Query History

```
1 SELECT * FROM public."Товар"
2 ORDER BY "id_товара" ASC
```

Data Output Messages Notifications

	id_товара [PK] integer	Наименование character varying (100)	Описание text	Цена_за_ед numeric (10,2)	id_категории integer	id_полки integer	количество_тов integer
1	1	Смартфон	Iphone 15 Pro Black 256 Gb	110000.00	1	1	20
2	2	Ноутбук	Ноутбук Apple MacBook Air 13 M2 8 core/8 core/8/256/Space Gray (MLXW...	120000.00	1	2	15
3	3	Телевизор	Samsung QLED 4K Smart TV	80000.00	1	3	25
4	4	Холодильник	Двухкамерный холодильник Haier	70000.00	2	4	12
5	5	Стиральная машина	LG Автоматическая стиральная машина	30000.00	2	5	10
6	6	Футболка	Хлопковая футболка черная	2000.00	3	6	30
7	7	Джинсы	Синие джинсы Calvin Klein	50.00	3	7	25
8	8	Сыр	Сыр Рокишкьо	350.00	4	8	8
9	9	Ручка	Гелевая ручка	11.50	5	9	50
10	10	Тетрадь	Школьная тетрадь в клеточку	20.00	5	10	40
11	11	Миксер	Кухонный миксер Tefal	4500.00	2	1	5
12	12	Блендер	Мощный блендер	6000.00	2	2	8
13	13	Кроссовки	Nike Airforce 1	12000.00	3	3	20
14	14	Часы	Наручные часы Rolex	150000.00	3	4	15
15	15	Шоколад	Молочный шоколад Milka	90.50	4	5	35
16	16	Колбаса	Копченая колбаса	230.00	4	6	10
17	17	Маркеры	Набор цветных маркеров	150.00	5	7	20
18	18	Степлер	Настольный степлер	120.00	5	8	15
19	19	Настольная лампа	Светодиодная настольная лампа Xiaomi	3000.00	8	9	10
20	20	Кресло	Компьютерное кресло Zone 51	18000.00	8	10	5
21	21	Пылесос	Умный пылесос Xiaomi	17050.00	2	1	12
22	22	Утюг	Паровой утюг Tefal	3500.00	2	2	8
23	23	Куртка	Теплая зимняя куртка Stone Island	12000.00	3	3	10
24	24	Кепка	Летняя кепка	1500.00	3	4	20
25	25	Кофе	Молотый кофе	250.00	4	5	15
26	26	Чай	Зеленый чай Greenfield	55.50	4	6	10
27	27	Ежедневник	Кожанный ежедневник	2500.00	5	7	25
28	28	Ластик	Школьный ластик	10.50	5	8	30
29	29	Гантели	Гантели 5 кг	400.00	7	9	15
30	30	Велотренажер	Домашний велотренажер	35000.00	7	10	9
31	31	Шкаф	Двухдверный шкаф	30000.00	8	1	10

Рисунок 4.7 – Таблица «Товар»

CREATE запрос:

```
CREATE TABLE Позиция_заказа (  
    id_поз_заказа SERIAL PRIMARY KEY,  
    Количество_зак_тов INT NOT NULL,  
    id_товара INT REFERENCES Товар(id_товара),  
    id_заказа INT REFERENCES Заказ(id_заказа)  
);
```

INSERT запрос:

```
INSERT INTO Позиция_заказа (id_поз_заказа, Количество_зак_тов, id_товара, id_заказа)  
VALUES  
(1, 2, 1, 1),  
(2, 1, 2, 1),  
(3, 5, 3, 2),  
(4, 3, 4, 3),  
(5, 4, 6, 5),  
(6, 6, 7, 6),  
(7, 8, 8, 7),  
(8, 10, 9, 8),  
(9, 1, 10, 9),  
(10, 3, 11, 10),  
(11, 4, 13, 11),  
(12, 1, 14, 11),  
(13, 5, 15, 12),  
(14, 6, 16, 12),  
(15, 2, 17, 13),  
(16, 3, 18, 13),  
(17, 7, 19, 14),  
(18, 9, 20, 14),  
(19, 2, 21, 15),  
(20, 5, 23, 16),  
(21, 1, 24, 16),  
(22, 2, 25, 17),  
(23, 3, 26, 17),  
(24, 6, 27, 18),  
(25, 8, 28, 18),  
(26, 3, 29, 19),  
(27, 5, 31, 20),  
(28, 1, 32, 20),  
(29, 4, 33, 21),  
(30, 6, 34, 21),  
(31, 2, 35, 22),
```

(32, 3, 36, 22),
 (33, 7, 37, 23),
 (34, 9, 39, 23);

Наполнение таблицы «Позиция_заказа» изображено на рисунке 4.8.

Query Query History

```
1 SELECT * FROM public."Позиция_заказа"
2 ORDER BY "id_поз_заказа" ASC
```

Data Output Messages Notifications

	id_поз_заказа [PK] integer	Количество_зак_тов integer	id_товара integer	id_заказа integer
1	1	2	1	1
2	2	1	2	1
3	3	5	3	2
4	4	3	4	3
5	5	4	6	5
6	6	6	7	6
7	7	8	8	7
8	8	10	9	8
9	9	1	10	9
10	10	3	11	10
11	11	4	13	11
12	12	1	14	11
13	13	5	15	12
14	14	6	16	12
15	15	2	17	13
16	16	3	18	13
17	17	7	19	14
18	18	9	20	14
19	19	2	21	15
20	20	5	23	16
21	21	1	24	16
22	22	2	25	17
23	23	3	26	17
24	24	6	27	18
25	25	8	28	18
26	26	3	29	19
27	27	5	31	20
28	28	1	32	20
29	29	4	33	21
30	30	6	34	21
31	31	2	35	22

Рисунок 4.8 – Таблица «Позиция_заказа»

CREATE запрос:

```
CREATE TABLE Товары_в_отгрузке (  
    id_тов_в_отгрузке SERIAL PRIMARY KEY,  
    id_отгрузки INT REFERENCES Отгрузка(id_отгрузки),  
    id_товара INT REFERENCES Товар(id_товара),  
    количество_товара INT NOT NULL,  
    id_статуса INT REFERENCES Статус_отгрузки(id_статуса)  
);
```

INSERT запрос:

```
INSERT INTO Товары_в_отгрузке (id_тов_в_отгрузке, id_отгрузки, id_товара,  
количество_товара, id_статуса)  
VALUES  
(1, 1, 1, 2, 1),  
(2, 1, 2, 1, 2),  
(3, 2, 3, 5, 3),  
(4, 3, 4, 3, 2),  
(5, 5, 6, 4, 1),  
(6, 6, 7, 6, 2),  
(7, 7, 8, 8, 3),  
(8, 8, 9, 10, 1),  
(9, 9, 10, 1, 2),  
(10, 10, 11, 3, 3),  
(11, 11, 13, 4, 1),  
(12, 11, 14, 1, 2),  
(13, 12, 15, 5, 3),  
(14, 12, 16, 6, 1),  
(15, 13, 17, 2, 2),  
(16, 13, 18, 3, 3),  
(17, 14, 19, 7, 1),  
(18, 14, 20, 9, 2),  
(19, 15, 21, 2, 3),  
(20, 16, 23, 5, 1),  
(21, 16, 24, 1, 2),  
(22, 17, 25, 2, 3),  
(23, 17, 26, 3, 1),  
(24, 18, 27, 6, 2),  
(25, 18, 28, 8, 3),  
(26, 19, 29, 3, 1),  
(27, 20, 31, 5, 2),  
(28, 20, 32, 1, 3),  
(29, 21, 33, 4, 1),  
(30, 21, 34, 6, 2),
```

(31, 22, 35, 2, 3),
 (32, 22, 36, 3, 1),
 (33, 23, 37, 7, 2),
 (34, 23, 39, 9, 3);

Наполнение таблицы «Товары_в_отгрузке» изображено на рисунке 4.9.

Query Query History

```
1 SELECT * FROM public."Товары_в_отгрузке"
2 ORDER BY "id_тов_в_отгрузке" ASC
```

Data Output Messages Notifications

	id_тов_в_отгрузке [PK] integer	id_отгрузки integer	id_товара integer	количество_товара integer	id_статуса integer
1	1	1	1	2	1
2	2	1	2	1	2
3	3	2	3	5	3
4	4	3	4	3	2
5	5	5	6	4	1
6	6	6	7	6	2
7	7	7	8	8	3
8	8	8	9	10	1
9	9	9	10	1	2
10	10	10	11	3	3
11	11	11	13	4	1
12	12	11	14	1	2
13	13	12	15	5	3
14	14	12	16	6	1
15	15	13	17	2	2
16	16	13	18	3	3
17	17	14	19	7	1
18	18	14	20	9	2
19	19	15	21	2	3
20	20	16	23	5	1
21	21	16	24	1	2
22	22	17	25	2	3
23	23	17	26	3	1
24	24	18	27	6	2
25	25	18	28	8	3
26	26	19	29	3	1
27	27	20	31	5	2
28	28	20	32	1	3
29	29	21	33	4	1
30	30	21	34	6	2
31	31	22	35	2	3

Рисунок 4.9 – Таблица «Товары_в_отгрузке»

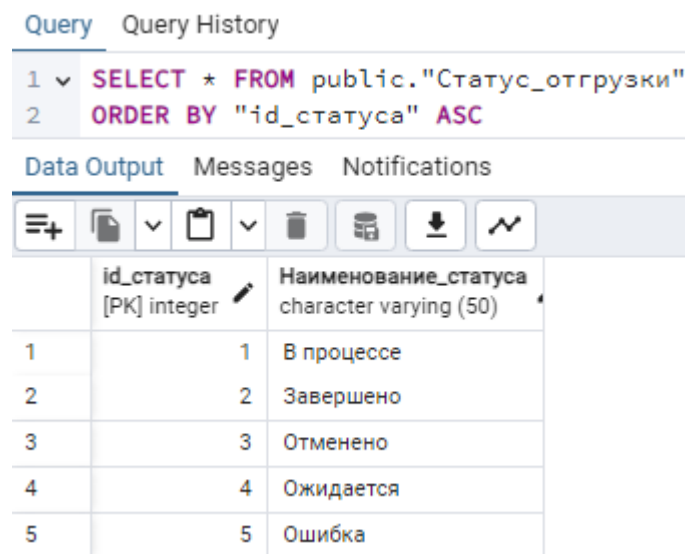
CREATE запрос:

```
CREATE TABLE Статус_отгрузки (  
    id_статуса SERIAL PRIMARY KEY,  
    Наименование_статуса VARCHAR(50) NOT NULL  
);
```

INSERT запрос:

```
INSERT INTO Статус_отгрузки (id_статуса, Наименование_статуса)  
VALUES  
(1, 'В процессе'),  
(2, 'Завершено'),  
(3, 'Отменено'),  
(4, 'Ожидается'),  
(5, 'Ошибка');
```

Наполнение таблицы «Статус_отгрузки» изображено на рисунке 4.10.



The screenshot shows a database interface with two tabs: 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query: `SELECT * FROM public."Статус_отгрузки" ORDER BY "id_статуса" ASC`. Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with two columns: 'id_статуса' (integer, primary key) and 'Наименование_статуса' (character varying (50)). The table contains five rows of data.

	id_статуса [PK] integer	Наименование_статуса character varying (50)
1	1	В процессе
2	2	Завершено
3	3	Отменено
4	4	Ожидается
5	5	Ошибка

Рисунок 4.10 – Таблица «Статус_отгрузки»

A.2 Файл Sklad.py, содержащий класс базы данных и контроллеры

```
import tkinter as tk
from tkinter import messagebox, Toplevel, font, scrolledtext
from tkinter import ttk
import psycopg2
from psycopg2._psycopg import cursor

class DatabaseApp:
    def __init__(self, root):
        # Инициализация основного окна и компонентов интерфейса
        self.root = root
        self.root.title("Графический интерфейс PostgreSQL")

        # Настройка цветов и шрифтов
        self.bg_color = "#F4E1D2"
        self.entity_button_color = "#A9CBB7"
        self.button_color = "#6C8A74"
        self.entry_bg = "#E8E8E8"
        self.button_fg = "white"
        self.label_font = font.Font(family="Verdana", size=12)

        # Настройка основных компонентов графического интерфейса
        self.root.configure(bg=self.bg_color)

        # Создание фрейма для кнопок отчетов
        self.report_frame = tk.Frame(self.root, bg=self.bg_color)
        self.report_frame.grid(row=1, column=1, sticky="nsew", padx=20, pady=10)

        # Добавление надписи "ОТЧЁТЫ"
        report_label = tk.Label(self.report_frame, text="ОТЧЁТЫ", font=self.label_font,
bg=self.bg_color)
        report_label.grid(row=0, column=0, columnspan=2, pady=10)

        # Кнопки для отчетов
        self.btn_order_volume = tk.Button(self.report_frame, text="Объемы заказов по каждому товару на
текущий месяц",
command=self.report_order_volume,
bg=self.entity_button_color, fg="black",
font=self.label_font, height=2)
        self.btn_order_volume.grid(row=1, column=0, pady=5, sticky="ew")

        self.btn_stock_left = tk.Button(self.report_frame, text="Остатки товаров на складе",
command=self.report_stock_left, bg=self.entity_button_color,
```



```

fg="black",

                                font=self.label_font, height=2)
self.btn_stock_left.grid(row=2, column=0, pady=5, sticky="ew")

self.btn_stock_needed = tk.Button(self.report_frame, text="Товары, которые необходимо завезти",
                                command=self.report_stock_needed,
bg=self.entity_button_color, fg="black",
                                font=self.label_font, height=2)
self.btn_stock_needed.grid(row=3, column=0, pady=5, sticky="ew")

self.btn_no_demand = tk.Button(self.report_frame, text="Товары, не пользующиеся спросом",
                                command=self.report_no_demand, bg=self.entity_button_color,
fg="black",
                                font=self.label_font, height=2)
self.btn_no_demand.grid(row=4, column=0, pady=5, sticky="ew")

self.btn_items_in_order = tk.Button(self.report_frame, text="Список товаров в заказах",
                                command=self.report_items_in_order,
bg=self.entity_button_color, fg="black",
                                font=self.label_font, height=2)
self.btn_items_in_order.grid(row=5, column=0, pady=5, sticky="ew")

# Определение сущностей и их атрибутов
self.entities = {
    'Категория': {'query': 'SELECT * FROM "Категория" ORDER BY "id_категории" ASC',
                  'attributes': ['id_категории', 'наименование_кат', 'Описание']},
    'Товар': {'query': 'SELECT * FROM "Товар" ORDER BY "id_товара" ASC',
              'attributes': ['id_товара', 'Наименование', 'Описание', 'Цена_за_ед',
'id_категории', 'id_полки',
                              'количество_тов']},
    'Полка': {'query': 'SELECT * FROM "Полка" ORDER BY "id_полки" ASC',
              'attributes': ['id_полки', 'номер_полки']},
    'Магазин': {'query': 'SELECT * FROM "Магазин" ORDER BY "id_магазина" ASC',
                'attributes': ['id_магазина', 'Название', 'Адрес', 'номер_тел']},
    'Заказ': {'query': 'SELECT * FROM "Заказ" ORDER BY "id_заказа" ASC',
              'attributes': ['id_заказа', 'Дата_заказа', 'id_магазина']},
    'Позиция_заказа': {'query': 'SELECT * FROM "Позиция_заказа" ORDER BY "id_поз_заказа" ASC',
                       'attributes': ['id_поз_заказа', 'Количество_зак_тов', 'id_товара',
'id_заказа']},
    'Накладная': {'query': 'SELECT * FROM "Накладная" ORDER BY "id_накладной" ASC',
                   'attributes': ['id_накладной', 'Дата_оформления', 'Общая_сумма']},
    'Отгрузка': {'query': 'SELECT * FROM "Отгрузка" ORDER BY "id_отгрузки" ASC',
                 'attributes': ['id_отгрузки', 'Дата_отгрузки', 'id_заказа', 'id_накладной']},
    'Товары_в_отгрузке': {'query': 'SELECT * FROM "Товары_в_отгрузке" ORDER BY
"id_тов_в_отгрузке" ASC',
                           'attributes': ['id_тов_в_отгрузке', 'id_отгрузки', 'id_товара',

```

```

'количество_товара',

                                'id_статуса']},

    'Статус_отгрузки': {'query': 'SELECT * FROM "Статус_отгрузки" ORDER BY "id_статуса" ASC',
                        'attributes': ['id_статуса', 'Наименование_статуса']},

}

# Создание первого фрейма для подключения к базе данных
input_frame = tk.Frame(self.root, bg=self.bg_color, padx=20, pady=20)
input_frame.grid(row=0, column=0, sticky="nsew", padx=20, pady=10)

# Поля ввода для данных подключения с автозаполнением
self.host_entry = self.create_entry("Хост:", input_frame, 0, default_value="localhost")
self.port_entry = self.create_entry("Порт:", input_frame, 1, default_value="8887")
self.dbname_entry = self.create_entry("Имя базы данных:", input_frame, 2,
default_value="Склад")
self.user_entry = self.create_entry("Пользователь:", input_frame, 3, default_value="postgres")
self.password_entry = self.create_entry("Пароль:", input_frame, 4, default_value="7521",
show="*")

# Кнопка для подключения к базе данных
self.connect_button = tk.Button(input_frame, text="Подключиться", font=self.label_font,
command=self.connect_db,

                                relief="raised", bg=self.button_color, fg=self.button_fg,
height=2)
self.connect_button.grid(row=5, column=0, columnspan=2, pady=12, sticky="ew")

# Создание фрейма для кнопок работы с сущностями
table_frame = tk.Frame(self.root, bg=self.bg_color)
table_frame.grid(row=0, column=1, sticky="nsew", padx=20, pady=10)

# Заголовок для таблиц
table_label = tk.Label(table_frame, text="Таблицы базы данных 'Склад'", font=self.label_font,
bg=self.bg_color)
table_label.grid(row=0, column=0, columnspan=2, pady=10)

# Кнопки для взаимодействия с таблицами базы данных
row = 1
col = 0
for i, (entity, info) in enumerate(self.entities.items()):
    if i % 2 == 0 and i != 0:
        row += 1
        col = 0
    button = tk.Button(table_frame, text=entity,
                        command=lambda e=entity, q=info['query'], a=info['attributes']:
self.open_entity_window(

                                e, q, a),

```

```

        bg=self.entity_button_color, fg="black", font=self.label_font)
    button.grid(row=row, column=col, sticky="ew", padx=6, pady=6, ipadx=10)
    col += 1

def create_entry(self, label_text, parent, row, show=None, default_value=None):
    # Создание поля ввода с меткой и автозаполнением
    label = tk.Label(parent, text=label_text, font=self.label_font, bg=self.bg_color)
    label.grid(row=row, column=0, sticky="w", pady=6, padx=6)

    entry = tk.Entry(parent, show=show, bg=self.entry_bg)
    entry.grid(row=row, column=1, sticky="ew", padx=12, pady=6, ipadx=10)

    # Устанавливаем значение по умолчанию, если оно передано
    if default_value:
        entry.insert(0, default_value)

    return entry

def connect_db(self):
    # Подключение к базе данных PostgreSQL с использованием введенных данных
    try:
        self.connection = psycopg2.connect(
            host=self.host_entry.get(),
            port=self.port_entry.get(),
            dbname=self.dbname_entry.get(),
            user=self.user_entry.get(),
            password=self.password_entry.get()
        )
        messagebox.showinfo("Успех", "Успешно подключено к базе данных")
    except Exception as e:
        messagebox.showerror("Ошибка", str(e))

def get_next_id(self, table_name, id_column):
    """Получает следующий id для указанной таблицы."""
    try:
        with self.connection.cursor() as cur:
            cur.execute(f'SELECT MAX("{id_column}") FROM "{table_name}";')
            result = cur.fetchone()
            max_id = result[0] if result[0] else 0
            return max_id + 1
    except Exception as e:
        messagebox.showerror("Ошибка", f"Не удалось получить следующий id: {e}")
        return None

def open_entity_window(self, entity_name, query, attributes):
    # Открывает окно для работы с сущностью базы данных

```

```

window = Toplevel(self.root)
window.title(f"Управление {entity_name}")

# Настройка таблицы для вывода результата запроса
result_tree = ttk.Treeview(window, columns=attributes, show="headings")
result_tree.grid(row=0, column=0, rowspan=6, columnspan=4, sticky='nsew', padx=6, pady=2)

# Добавление вертикальной полосы прокрутки
vsb = ttk.Scrollbar(window, orient="vertical", command=result_tree.yview)
vsb.grid(row=0, column=4, rowspan=6, sticky='ns')
result_tree.configure(yscrollcommand=vsb.set)

window.grid_rowconfigure(0, weight=1)
window.grid_columnconfigure(0, weight=1)

# Настройка заголовков для таблицы
for attribute in attributes:
    result_tree.heading(attribute, text=attribute)

# Выполнение запроса и отображение данных сразу
self.execute_query(query, result_tree)

# Поля для редактирования данных
attribute_entries = {}
for i, attribute in enumerate(attributes):
    attribute_label = tk.Label(window, text=attribute, font=self.label_font)
    attribute_label.grid(row=i + 1, column=5, padx=6, pady=2)

    attribute_entry = tk.Entry(window, width=50, bg=self.entry_bg)
    attribute_entry.grid(row=i + 1, column=6, padx=6, pady=2)

    # Заполнение только первого id (первичный ключ) автоматически
    if attribute.startswith("id_") and i == 0: # только для первого id
        table_name = entity_name
        next_id = self.get_next_id(table_name, attribute)
        if next_id is not None:
            attribute_entry.insert(0, next_id)
            attribute_entry.config(state="disabled") # Запрещаем редактировать id вручную

    # Заполнение всех остальных атрибутов (оставляем их пустыми)
    attribute_entries[attribute] = attribute_entry

# Кнопка для добавления новой записи
submit_button = tk.Button(window, text="Добавить",
                           command=lambda: self.insert_record(entity_name, query,
attribute_entries),

```

```

        bg=self.button_color, fg=self.button_fg, height=2, width=11,
        font=("Arial", 10))
submit_button.grid(row=i + 3, column=6, columnspan=2, pady=2)

# Кнопка для редактирования записи
edit_button = tk.Button(window, text="Редактировать",
                        command=lambda: self.edit_record(entity_name, result_tree,
attribute_entries),
                        bg=self.button_color, fg=self.button_fg, height=2, width=11,
                        font=("Arial", 10))
edit_button.grid(row=7, column=0, columnspan=4, pady=10)

# Привязка клавиши Delete к удалению записи
result_tree.bind("<Delete>", lambda event: self.delete_selected_record(entity_name,
attributes[0], result_tree))

def edit_record(self, entity_name, result_tree, attributes):
    # Отладочный вывод для проверки attributes
    print("Полученные attributes:", attributes)
    if isinstance(attributes, dict):
        attributes = list(attributes.keys())
    if not isinstance(attributes, list):
        messagebox.showerror("Ошибка", "Attributes должен быть списком.")
        return
    selected_item = result_tree.selection()
    if not selected_item:
        messagebox.showwarning("Выбор записи", "Выберите запись для редактирования.")
        return

    values = result_tree.item(selected_item, "values")
    edit_window = Toplevel(self.root)
    edit_window.title(f"Редактирование {entity_name}")

    entries = {}
    row = 0
    for i, attribute in enumerate(attributes):
        tk.Label(edit_window, text=attribute).grid(row=row, column=0, padx=10, pady=5)
        entry = tk.Entry(edit_window, width=40)
        entry.insert(0, values[i])
        entry.grid(row=row, column=1, padx=10, pady=5)
        entries[attribute] = entry
        row += 1

    def save_changes():
        new_values = [entry.get() for entry in entries.values()]
        record_id = values[0]

```

```

# Формирование SQL-запроса
sql_set_clause = ", ".join([f"{attr} = %s" for attr in attributes[1:]])

sql = f"UPDATE {entity_name} SET {sql_set_clause} WHERE {attributes[0]} = %s"
print("SQL запрос:", sql) # Отладочный вывод запроса
cursor = self.connection.cursor()
cursor.execute(sql, (*new_values[1:], record_id))
self.connection.commit()
result_tree.item(selected_item, values=new_values)
edit_window.destroy()
messagebox.showinfo("Успех", "Изменения сохранены!")

tk.Button(edit_window, text="Сохранить", command=save_changes).grid(row=row, column=0,
columnspan=2, pady=10)

def execute_query(self, query, result_tree):
    # Выполнение SQL-запроса и отображение результатов
    try:
        with self.connection.cursor() as cur:
            cur.execute(query)
            rows = cur.fetchall()
            columns = [desc[0] for desc in cur.description]

            # Очистка старых данных из Treeview
            result_tree.delete(*result_tree.get_children())
            result_tree["columns"] = columns
            result_tree["show"] = "headings"

            # Установка заголовков колонок
            for col in columns:
                result_tree.heading(col, text=col)
                result_tree.column(col, width=100)

            # Добавление данных в таблицу
            for row in rows:
                result_tree.insert("", "end", values=row)
    except Exception as e:
        messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")

# Функции для выполнения SQL-запросов

def execute_report_query(self, query):
    # Выполнение SQL-запроса и отображение результатов в новой вкладке
    result_window = Toplevel(self.root)
    result_window.title("Отчет")

```

```

result_tree = ttk.Treeview(result_window)
result_tree.grid(row=0, column=0, rowspan=6, columnspan=4, sticky='nsew', padx=6, pady=2)

vsb = ttk.Scrollbar(result_window, orient="vertical", command=result_tree.yview)
vsb.grid(row=0, column=4, rowspan=6, sticky='ns')
result_tree.configure(yscrollcommand=vsb.set)

cursor = self.connection.cursor()
try:
    cursor.execute(query)
    result = cursor.fetchall()
    columns = [desc[0] for desc in cursor.description]
    result_tree["columns"] = columns
    for col in columns:
        result_tree.heading(col, text=col)
        result_tree.column(col, width=100)

    for row in result:
        result_tree.insert("", "end", values=row)
except Exception as e:
    messagebox.showerror("Ошибка", f"Не удалось выполнить запрос:\n{str(e)}")

def report_order_volume(self):
    # SQL-запрос для получения объема заказов по каждому товару
    query = """
SELECT
    t.id_товара,
    t.Наименование,
    SUM(pz.Количество_зак_тов) AS Общий_объем_заказов
FROM
    Товар t
LEFT JOIN
    Позиция_заказа pz ON t.id_товара = pz.id_товара
LEFT JOIN
    Заказ z ON pz.id_заказа = z.id_заказа
WHERE
    z.Дата_заказа IS NOT NULL
    AND z.Дата_заказа BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY
    t.id_товара, t.Наименование
ORDER BY
    Общий_объем_заказов DESC;
    """

    self.execute_report_query(query)

```

```

def report_stock_left(self):
    query = """
        SELECT
            t.id_товара,
            t.Наименование,
            t.количество_тов - COALESCE(SUM(tv.количество_товара), 0) AS Остаток_на_складе
        FROM
            Товар t
        LEFT JOIN
            Товары_в_отгрузке tv ON t.id_товара = tv.id_товара
        GROUP BY
            t.id_товара, t.Наименование, t.количество_тов
        ORDER BY
            t.id_товара;
    """

```

```

self.execute_report_query(query)

```

```

def report_stock_needed(self):
    query = """
        SELECT
            t.id_товара,
            t.Наименование,
            t.количество_тов - COALESCE(tv.количество_товара, 0) AS Остаток_на_складе
        FROM
            Товар t
        LEFT JOIN
            Товары_в_отгрузке tv ON t.id_товара = tv.id_товара
        WHERE
            (t.количество_тов - COALESCE(tv.количество_товара, 0)) < 10
        ORDER BY
            t.id_товара;
    """

```

```

self.execute_report_query(query)

```

```

def report_no_demand(self):

```

```

    query = """
        SELECT
            t.id_товара,
            t.Наименование,
            t.Описание
        FROM
            Товар t
        LEFT JOIN

```



```

        Позиция_заказа pz ON t.id_товара = pz.id_товара
LEFT JOIN
        Заказ z ON pz.id_заказа = z.id_заказа
WHERE
        z.id_заказа IS NULL
ORDER BY
        t.Наименование;
"""

self.execute_report_query(query)

def report_items_in_order(self):
    query = """
SELECT
    t.id_товара,
    t.Наименование,
    t.Описание,
    COALESCE(SUM(pz.Количество_зак_тов), 0) AS Количество_в_заказах
FROM
    Товар t
LEFT JOIN
    Позиция_заказа pz ON t.id_товара = pz.id_товара
WHERE
    pz.id_заказа = 1
GROUP BY
    t.id_товара, t.Наименование, t.Описание
ORDER BY
    t.id_товара;
"""

    self.execute_report_query(query)

def delete_selected_record(self, table_name, id_column, treeview):
    """Удаляет выбранную запись из таблицы."""
    selected_item = treeview.selection()
    if not selected_item:
        messagebox.showwarning("Предупреждение", "Выберите запись для удаления")
        return

    try:
        item_id = treeview.item(selected_item, "values")[0]
        with self.connection.cursor() as cur:
            cur.execute(f'DELETE FROM "{table_name}" WHERE "{id_column}" = %s', (item_id,))
            self.connection.commit()
            treeview.delete(selected_item)
            messagebox.showinfo("Успех", "Запись успешно удалена")
    
```

```

except Exception as e:
    messagebox.showerror("Ошибка", f"Не удалось удалить запись: {e}")

def insert_record(self, entity_name, query, attribute_entries):
    # Добавление новой записи в таблицу
    try:
        columns = ', '.join(attribute_entries.keys())
        values = ', '.join([f"'{entry.get()}'" for entry in attribute_entries.values()])
        insert_query = f'INSERT INTO "{entity_name}" ({columns}) VALUES ({values});'

        with self.connection.cursor() as cur:
            cur.execute(insert_query)
            self.connection.commit()

        messagebox.showinfo("Успех", "Запись успешно добавлена")
    except Exception as e:
        messagebox.showerror("Ошибка", f"Не удалось добавить запись: {e}")

if __name__ == "__main__":
    root = tk.Tk()
    app = DatabaseApp(root)
    root.mainloop()

```