

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Зав. кафедрой \_\_ПОВТ и АС

\_\_\_\_\_\_\_\_\_\_\_\_ Долгов В.В.

подпись ФИО

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024г.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту по дисциплине Базы данных

на тему Проектирование и реализация базы данных склада запчастей

Автор проекта (работы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волкова Э.Ю.

подпись Ф.И.О.

Направление/специальность, профиль/специализация:

02.03.03 Математическое обеспечение и администрирование информационных систем

коднаправления наименование профиля (специализации)

Обозначение курсового проекта (работы) КР.350000.000 Группа ВМО31

Руководитель проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ст. преподаватель Новиков С.П. подпись (должность, ФИО)

Проект (работа) защищен (а)

дата оценка подпись

г. Ростов-на-Дону

2024 г.



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Зав. кафедрой ПОВТ и АС

\_\_\_\_\_\_\_\_\_\_\_\_ Долгов В.В.

(подпись) (ФИО.)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024г.

**ЗАДАНИЕ**

к курсовой работе по дисциплине Базы данных

Студент Волкова Э.Ю. КР.350000.000 Группа ВМО31

Тема Проектирование и реализация базы данных склада запчастей выв

Срок представления проекта (работы) к защите « » 2024г.

Исходные данные для курсового проекта (работы)

Задание на выполнение курсовой работы

Дейт, Кристофер – Введение в системы баз данных, 8-е издание / Пер. с англ. – М.: Издательский дом «Вильяме», 2005 – 1328 с.: ил. – Парал. тит. англ. – ISBN 5-8459-0788-8

Паттон Джефф. Пользовательские истории. Искусство гибкой разработки ПО. – СПб.: Питер, 2019. – 288 с.

Карпова Т.С. Базы данных: модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.

Содержание пояснительной записки

ВВЕДЕНИЕ:

В данном разделе содержится: описание ключевых понятий, которые фигурируют в контексте баз данных, постановка задачи, выполнение которой сводится к разработке базы данных для склада запчастей, краткие описания каждой главы.

Наименование и содержание разделов:

1. Анализ предметной области и проектирование структуры базы данных. Этот раздел посвящен описанию выбранной предметной области. Приведено подробное описание видения базы данных конечными пользователями, описаны их основные функции. Описана модель уровня представлений.
2. Физическая реализация разработанной модели. В данном разделе описаны основные функции, используемые для работы с базой данных, спроектированы таблицы в системе управления базой данных.
3. Разработка клиентского интерфейса для работы с базой данных. Данный раздел содержит описание классов, необходимых для создания приложения, их методов. Произведено тестирование приложения.

ЗАКЛЮЧЕНИЕ:

В заключении курсовой работы отображены результаты решения поставленной задачи – была разработана база данных склада запчастей, а также программное средство с графическим интерфейсом в виде оконного приложения, соответствующего требованиям, предъявляемыми конечными пользователями.

Руководитель проекта (работы) Новиков С.П.

подпись, дата ФИО

Задание принял(а) к исполнению Волкова Э.Ю.

подпись, дата ФИО

**Содержание**

[Введение 5](#_Toc195060821)

[1 Анализ предметной области и проектирования структуры базы данных 7](#_Toc195060822)

[1.1 Формирование уровня представлений базы данных склада запчастей 7](#_Toc195060823)

[1.1.1 Описание представлений пользователей и выделение основных сущностей 7](#_Toc195060824)

[1.1.2 Описание объектов предметной области, их атрибутов и доменов 10](#_Toc195060825)

[1.1.3 Создание схемы в виде ER-диаграммы 13](#_Toc195060826)

[1.2.2 Нормализация отношений 20](#_Toc195060827)

[1.2.2.1 Нормализация до первой нормальной формы (1НФ) 20](#_Toc195060828)

[1.2.2.2 Выбор первичных ключей и нормализация до второй нормальной формы (2НФ) 21](#_Toc195060829)

[1.2.2.3 Нормализация до третьей нормальной формы (3НФ) 22](#_Toc195060830)

[1.2.3 Логическое (даталогическое) проектирование базы данных 24](#_Toc195060831)

[2 Физическая реализация модели БД склада запчастей 25](#_Toc195060832)

[2.1 Обоснование выбора системы управления базами данных (СУБД) 25](#_Toc195060833)

[2.2 Создание таблиц данных в среде целевой СУБД 26](#_Toc195060834)

[3 Разработка клиентского интерфейса для работы с базой данных 33](#_Toc195060835)

[3.1 Выбор среды для разработки приложения 33](#_Toc195060836)

[3.2 Описание основных модулей и функций интерфейса 34](#_Toc195060837)

[3.3 Пример работы графического интерфейса 36](#_Toc195060838)

[Заключение 43](#_Toc195060839)

[Приложение А ER-диаграмма 45](#_Toc195060840)

[Приложение Б Даталогическая схема 46](#_Toc195060841)

[Приложение Г Исходный код программного средства 48](#_Toc195060842)

[Приложение Д Исходный код базы данных склада запчастей 130](#_Toc195060843)

[Приложение Е Отзыв руководителя 154](#_Toc195060844)

# **Введение**

Каждый день генерируются гигантские объемы данных, работать с которыми без систем хранения и структурирования не предоставляется возможным. Для работы с такими масштабами получаемой информации необходимы инструменты, организующие работу, взаимодействие, хранение и доступ к данным. Таким инструментом и являются базы данных.

База данных — это информационная модель, позволяющая в упорядоченном виде хранить данные о группе объектов, обладающих одинаковым набором свойств [1]. Иначе говоря, база данных – это именованная совокупность взаимосвязанных данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области, при такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений. База данных состоит из множества связанных файлов и используется в целях отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей. Данные запоминаются так, чтобы они были независимы от программ, использующих эти данные. Для добавления новых или модификации существующих данных, а также для поиска данных в базе применяется система управления базами данных (СУБД).

СУБД – это программное обеспечение, предназначенное для создания, ведения и совместного использования базы данных многими пользователями. СУБД осуществляют ввод, проверку, систематизацию, поиск и обработку данных, распечатку их в виде отчётов. Наиболее распространенными СУБД являются MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

Главным источником данных при проектировании базы данных является соответствующая предметная область, которая будет рассматриваться как совокупность знаний и данных об объектах и процессах, подлежащих проектированию и хранению в БД.

Целью данной работы является разработка базы данных и графического клиентского интерфейса для склада запчастей.

В первой главе выполнен анализ предметной области, выделены основные сущности и их атрибуты, описана база данных на внешнем и концептуальном уровнях.

Во второй главе описана реализация базы данных склада запчастей на физическом уровне.

В третьей главе описана реализация клиентского интерфейса и рассмотрена работа приложения на тестовом примере.

# **1 Анализ предметной области и проектирования структуры базы данных**

Проектирование базы данных принято разбивать на несколько уровней абстракций, чтобы точно описать необходимые элементы не только для хранения информации, но и для её обработки.

## 1.1 Формирование уровня представлений базы данных склада запчастей

### 1.1.1 Описание представлений пользователей и выделение основных сущностей

Предметная область базы данных для учета запчастей на складе включает в себя информацию о различных аспектах складского хозяйства. Она включает в себя несколько ключевых элементов для базы данных.

**Детали**: в базе данных должна быть информация о всех деталях, которые хранятся на складе. Каждая запись о детали должна содержать уникальный идентификатор, название, параметры (например, вес, размеры).

**Складские помещения и полки**: для эффективного учета деталей необходимо иметь информацию о расположении товаров на складе. Это включает в себя данные о складских помещениях и полках, на которых хранятся детали.

**Отправки и поставки**: база данных должна содержать информацию об отправках, поставках, дате и времени прихода или расхода, количестве деталей, статусе заказа и других связанных с заказами данных.

**Учет перемещений**: для отслеживания перемещения деталей на складе необходима информация о перемещениях товаров между складскими помещениями, состоянии товаров, ответственных ­­­сотрудниках, дате перемещения и других связанных с перемещением данных.

**Пользователи и права доступа**: для обеспечения безопасности и контроля доступа к данным в базе должна быть реализована система управления пользователями с различными уровнями доступа к информации в зависимости от их ролей на складе.

База данных склада запчастей необходима прежде всего для хранения информации о пользователях, находящихся на складе деталей, поступивших и выполненных накладных.

Для каждого пользователя должен быть строгий раздел доступных действий, которые может выполнить тот или иной пользователь. Иначе говоря, кладовщик не должен иметь доступ к добавлению новых накладных в базу данных, так как его задача собрать необходимые детали на отгрузку или разложить по полкам поступившие. Накладными должен заниматься менеджер склада, который также будет давать ограниченный доступ к определённым накладным кладовщику. Именно менеджер должен отслеживать этапы отгрузки запчастей со склада или их поступления. Владелец склада должен в свою очередь добавлять новых пользователей, выдавать им роль в базе данных склада, а также добавлять новых контрагентов, с которыми он сотрудничает.

На основе предметной области можно выделить следующих

пользователей: кладовщик, менеджер склада, владелец склада.

**Кладовщик:** осуществляет приемку поставок, выкладку товаров на полки, при этом происходит заполнение в таблице «Стеллаж», указывает данные в таблице «Полка», поле чего формируется ссылки на «Стеллаж», «Помещение» и «Склад». При приёме поставки к соответствующей накладной прикрепляется сотрудник с ролью Кладовщик. Данный сотрудник получает в виде документа список деталей, а также их местоположение на складе, которое предварительно сформировано с помощью запроса к таблицам: «Склад», «Помещение», «Стеллаж», «Полка», «Деталь». Сотрудник также может изменять поле «Состояние» (завершено/в процессе) в соответствии с выполнением задачи, отображаемого в таблице «Накладная». После внесения изменений проверяется количество имеющихся деталей, при установке статуса «завершено», количество деталей на складе уменьшается/увеличивается, полки освобождаются/заполняются, происходит изменения в таблицах: «Склад», «Помещение», «Стеллаж», «Полка», «Деталь».

Кладовщику важно знать, где именно на складе находится определённая деталь, поэтому необходимо реализовать поиск, по следующим таблицам: Склад, Помещение, Стеллаж, Полка, Деталь. Таким образом можно выделить следующие сущности: «Склад», «Помещение», «Стеллаж», «Полка», «Деталь», «Накладная».

**Менеджер склада:** получает в физическом виде накладные на выгрузку деталей на склад и на отгрузку деталей со склада. Сотрудник с такой ролью заполняет таблицу «Накладные», где указывает номер договора с контрагентом (табл.), дату и время выгрузки или отгрузки деталей со склада, а также статус выгрузки или отгрузки, список деталей, и закреплённого за накладного кладовщика.

Таким образом менеджеру склада необходимо вести учёт накладных на поставку и отправку деталей, что выделяется в следующие таблицы: «Накладные», «Накладные(сотрудник)», «Накладные(деталь)», «Контрагент», «Сотрудник».

**Владелец склада:** ему необходимо отслеживать информацию о сотрудниках, работающих на данном складе, а также о контрагентах, с которыми ведётся сотрудничество. Сотрудник с данной ролью может вносить изменения в таблицу «Сотрудник», заполняя (изменяя) поля «Фамилия», «Имя», «Отчество», «Роль». Владелец склада также должен иметь доступ к таблице «Контрагент», в которую он может вносить изменения: добавление новой записи о контрагенте, изменение уже имеющейся записи, удаление записи. В данных записях владелец склада заполняет (изменяет) поля «Наименование», «Контактное лицо», «Номер телефона» и «Адрес» (полный адрес, т.е. область, город, улица и т.д.).

Таким образом можно выделить следующие сущности: «Сотрудник» и «Контрагент».

Итого выделено десять сущностей: «Склад», «Помещение», «Стеллаж», «Полка», «Деталь», «Накладные», «Накладные(сотрудник)», «Накладные(деталь)», «Сотрудник», «Контрагент».

### 1.1.2 Описание объектов предметной области, их атрибутов и доменов

Далее будут описаны необходимые сущности и атрибуты для создания базы данных склада запчастей.

Таблица 1.1 – Отношение «warehouse» к сущности «Склад»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| warehouse\_id | Серийный номер |
| warehouse\_number | Целое число |
| address | Текст |

Ограничения:

warehouse\_id – первичный ключ (уникальные значения).

Таблица 1.2 – Отношение «room» к сущности «Помещение»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| room\_id | Серийный номер |
| warehouseID | Серийный номер |
| room\_number | Целое число |

Ограничения:

room\_id – первичный ключ (уникальные значения);

warehouseID – внешний ключ, ссылающийся на поле warehouse\_id в таблице «Склад».

Таблица 1.3 – Отношение «rack» к сущности «Стеллаж»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| rack\_id | Серийный номер |
| roomID | Серийный номер |
| rack\_number | Целое число |

Ограничения:

rack\_id – первичный ключ (уникальные значения);

roomID – внешний ключ, ссылающийся на поле rack\_id в таблице «Помещение».

Таблица 1.4 – Отношение «shelf» к сущности «Полка»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| shelf\_id | Серийный номер |
| rackID | Серийный номер |
| shelf\_number | Целое число |

Ограничения:

shelf\_id – первичный ключ (уникальные значения);

rackID – внешний ключ, ссылающийся на поле rack\_id в таблице «Стеллаж».

Таблица 1.5 – Отношение «details» к сущности «Деталь»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| detail\_id | Серийный номер |
| shelfID | Серийный номер |
| weight | Число с плавающей точкой |
| type\_detail | Текст |

Ограничения:

detail\_id – первичный ключ (уникальные значения);

shelfD – внешний ключ, ссылающийся на поле shelf\_id в таблице «Полка».

Таблица 1.6 – Отношение «counteragent» к сущности «Контрагент»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| counteragent\_id | Серийный номер |
| counteragent\_name | Строка (128) |
| contact\_person | Строка (128) |
| phone\_number | Большое целое число |
| address | Текст |

Ограничения:

counteragent\_id – первичный ключ (уникальные значения).

Таблица 1.7 – Отношение «invoice» к сущности «Накладная»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| invoice\_id | Серийный номер |
| counteragentID | Серийный номер |
| date\_time | Дата и время |
| type\_invoice | Булево значение |
| status | Булево значение |

Ограничения:

counteragent\_id – первичный ключ (уникальные значения);

counteragentID – внешний ключ, ссылающийся на поле counteragent\_id в таблице «Контрагент».

Таблица 1.8 – Отношение «invoice\_detail» к сущности «Накладная(деталь)»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| invoiceID | Серийный номер |
| detailID | Серийный номер |
| quantity | Целое число |

Ограничения:

invoice\_id – первичный ключ (уникальные значения);

detailID – внешний ключ, ссылающийся на поле counteragent\_id в таблице «Деталь».

Таблица 1.9 – Отношение «employee» к сущности «Сотрудник»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| employee\_id | Серийный номер |
| employee\_role | Строка (25) |
| last\_name | Строка (35) |
| first\_name | Строка (35) |
| patronymic | Строка (35) |

Ограничения:

employee\_id – первичный ключ (уникальные значения).

Таблица 1.10 – Отношение «invoice\_employee» к сущности «Накладная(сотрудник)»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| invoiceID | Серийный номер |
| responsible | Серийный номер |
| granted\_access | Серийный номер |
| when\_granted | Дата и время |

Ограничения:

invoiceID – внешний ключ, ссылающийся на поле invoice\_id в таблице «Накладная»;

responsible – внешний ключ, ссылающийся на поле employee\_id в таблице «Сотрудник»;

granted\_access – внешний ключ, ссылающийся на поле employee\_id в таблице «Сотрудник».

### 1.1.3 Создание схемы в виде ER-диаграммы

Создание логической схемы является следующим этапом в процессе разработки и проектирования базы данных магазина.

Логическое (даталогическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных даталогическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

В процессе разработки логическая модель данных постоянно тестируется и проверяется на соответствие требованиям пользователей.

Такая модель данных является источником информации для этапа физического проектирования и обеспечивает разработчика физической базы данных средствами поиска компромиссов, необходимых для достижения поставленных целей. При правильной организации сопровождения поддерживаемая модель данных позволяет точно и наглядно представить вносимые в базу данных изменения, а также оценить их влияние на прикладные программы и использование данных, уже имеющихся в базе. В настоящий момент фактическим стандартом в моделировании баз данных стала модель «сущность-связь».

Рассмотрим основные понятия, используемые для построения ER-диаграммы.

Сущность – это реальный или представляемый тип объекта, информация о котором должна сохраняться и быть доступна. В диаграммах сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности – это имя типа, а не некоторого конкретного экземпляра этого типа. Каждый экземпляр сущности (объект) должен быть отличим от любого другого экземпляра той же сущности.

Связь – это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Связь может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). Возможны связи на основе отношений: один-к-одному, один-ко-многим, многие-к-одному, многие-ко-многим.

Один-к-одному предполагает, что в каждый момент времени каждому элементу (кортежу) А соответствует 0 или 1 элементов (кортежей) В.

Один-ко-многим состоит в том, что в каждый момент времени каждому элементу (кортежу) А соответствует несколько элементов (кортежей) В.

Многие-к-одному предполагает, что в каждый момент времени множеству элементов (кортежей) А соответствует 1 элемент (кортеж) В.

Многие-ко-многим состоит в том, что в каждый момент времени множеству элементов (кортежей) А соответствует множество элементов (кортежей) В [2].

Таким образом, на основе представлений пользователей базы данных можно составить обобщенное представление предметной области, выделив основные объекты будущей базы данных и отбросив второстепенные. В качестве концептуальной модели была выбрана ER-диаграмма. ER-диаграммы удобны тем, что процесс выделения сущностей, атрибутов и связей является цикличным. ER-модель использует графическое изображение сущностей предметной области, их свойств (атрибутов), и взаимосвязей между сущностями.

В базе данных «Склад запчастей» можно выделить следующие основные сущности: «Сотрудник», «Контрагент», «Склад», «Помещение», «Стеллаж», «Полка», «Накладная», «Накладная(деталь)», «Накладная(сотрудник)», «Деталь».

Рассмотрим атрибуты каждой сущности, необходимые конечным пользователям для работы.

«Сотрудник»: фамилия, имя, отчество, роль.

«Контрагент»: наименование, контактное лицо, номер телефона, адрес.

«Склад»: номер, адрес.

«Помещение»: номер.

«Стеллаж»: номер.

«Полка»: номер.

«Накладная»: номер договора, дата/время, тип (загрузка/отгрузка), статус.

«Накладная(деталь)»: серийный номер, количество, номер накладной.

«Накладная(сотрудник)»: ответственное лицо, номер накладной.

«Деталь»: серийный номер, вид детали, вес.

Основные связи в проектируемой ER-диаграмме: каждая «Деталь» находится на «Полке», которая находится в «Стеллаже». «Стеллаж» находится в «Помещении», а оно находится на «Складе». «Накладная» оформляется «Контрагентом». «Сотрудник» имеет доступ к «Накладной(сотрудник)», которая включает в себя «Накладную». «Накладной(деталь)» включает себя «Деталь» и «Накладную».

Таким образом, на основе приведенных представлений пользователей, выделенных основных объектов, описанных атрибутов и связей была спроектирована семантическая модель предметной области склада запчастей (Приложение А: ER-диаграмма).

**1.1.4 Описание связей между объектами**

1. Склад (1) – Помещение (N): в таблице Склад каждая запись связана со множеством помещений из таблицы Помещение. Это связь один-ко-многим, так как к одному складу привязано множество помещений, но каждое помещение находится только на одном складе;
2. Помещение (1) – Стеллаж (N): в таблице Помещение каждая запись связана с множеством стеллажей из таблицы Стеллаж. Это связь один-ко-многим, так как к одному помещению привязано множество стеллажей, но каждый стеллаж находится только в одном помещении;
3. Стеллаж (1) – Полка (N): в таблице Стеллаж каждая запись связана со множеством полок из таблицы Полка. Это связь один-ко-многим, так как к одному стеллажу привязано множество полок, но каждая полка находится только на одном стеллаже;
4. Полка (1) – Деталь (N): в таблице Полка каждая запись связана со множеством деталей из таблицы Деталь. Это связь один-ко-многим, так как к одной полке привязано множество деталей, но каждая деталь находится только на одной полке;
5. Накладная(деталь) (N) – Накладная(сотрудник) (N): в таблице Накладная(деталь) каждая запись связана с каждым сотрудником ответственным за выполнение отгрузки/загрузки деталей с/на склад из таблицы Накладная(сотрудник). Это связь многие-ко-многим, так как множество записей таблицы Накладная(детали) связаны со множеством записей таблицы Накладная(сотрудник). Такое отношение стоит разделить на две части, а именно на связь многие-к-одному (Накладная(деталь) (N) – Накладная (1)) и один-ко-многим (Накладная (1) – Накладная(сотрудник) (N));
6. Накладная(деталь) (N) – Накладная (1): в таблице Накладная(деталь) каждая запись связана с одной накладной из таблицы Накладная. Это связь многие-к-одному, так как каждая накладная с количеством определённой детали привязана только к одной накладной, но каждая накладная может иметь несколько накладных с количеством определённых деталей;
7. Накладная (1) – Накладная(сотрудник) (N): в таблице Накладная каждая запись связана со множеством накладных доступных определённым сотрудникам из Накладная(сотрудник). Это связь один-ко-многим, так как к одной накладной привязано множество накладных с привязанными сотрудниками, но каждая накладная с привязанными сотрудниками включает в себя только одну накладную;
8. Деталь (1) – Накладная(деталь) (N): в таблице Деталь каждая запись связана со множеством накладных с количеством определённых деталей из таблицы Накладная(деталь). Это связь один-ко-многим, так как к одной детали привязано множество накладных количеством определённых деталей, но каждая накладная количеством определённых деталей содержит в себе только одну деталь;
9. Накладная(сотрудник) (N) – Сотрудник (1): в таблице Накладная(сотрудник) каждая запись связана с одним сотрудником из таблицы Сотрудник. Это связь многие-к-одному, так как каждая накладная с привязанными сотрудниками привязана только к одному сотруднику, но каждый сотрудник может быть привязан к нескольким накладным;
10. Накладная (N) – Контрагент (1): в таблице Накладная каждая запись связана с одним контрагентом из таблицы Контрагент. Это связь многие-к-одному, так как каждая накладная привязана только к одному контрагенту, но каждый контрагент может оформлять несколько накладных.

**1.2 Построение концептуального уровня базы данных**

**1.2.1 Описание функциональных зависимостей, имеющих место в предметной области**

**warehouse** = {warehouse\_id, warehouse\_number, address}

{warehouse\_number} -> {address}

Ограничение: по одному адресу не могут стоять несколько складов.

Каждый склад идентифицируется по уникальному warehouse\_id и атрибуту address, так как невозможно чтобы два склада находились по одному адресу. Атрибут address функционально зависит от warehouse\_number.

Таблица warehouse имеет в качестве первичного ключа warehouse\_id.

**room** = {room\_id, room\_number, warehouseID}

{warehouseID, room\_number} -> {room\_number}

Ограничение: на каждом складе может быть несколько комнат, но номер комнаты должен быть уникальным в пределах одного склада

Каждое помещение идентифицируется по уникальному room\_id, первичному ключу, а также по атрибуту warehouseID, который является внешним ключом, ссылающийся на таблицу warehouse. Атрибут room\_number функционально зависит от warehouseID и обозначает номер помещения в пределах определённого склада.

**rack** = {rack\_id, rack\_number, roomID}

{roomID, rack\_number} -> {rack\_number}

Ограничение: в каждой комнате может быть несколько стеллажей, но номер стеллажа должен быть уникальным в пределах одной комнаты

Каждый стеллаж идентифицируется по уникальному rack\_id, первичному ключу, а также по атрибуту roomID, который является внешним ключом, ссылающийся на таблицу room. Атрибут rack\_number функционально зависит от roomID и обозначает номер стеллажа в пределах определённого помещения.

**shelf** = {shelf\_id, shelf\_number, rackID}

{rackID, shelf\_number} -> {shelf\_number}

Ограничение: в одном стеллаже не могут быть несколько полок с одинаковым номером.

Каждая полка идентифицируется по уникальному shelf\_id, первичному ключу, а также по атрибуту rackID, который является внешним ключом, ссылающийся на таблицу rack. Атрибут shelf\_number функционально зависит от rackID и обозначает номер полки в пределах определённого стеллажа.

**counteragent** = {counteragent\_id, counteragent\_name, contact\_person, phone\_number, address}

{counteragent\_name} -> {contact\_person, phone\_number}

Каждый контрагент идентифицируется по уникальному counteragent\_id, первичному ключу. Атрибуты contact\_person и phone\_number функционально зависят от атрибута counteragent\_name и обозначают контактное лицо и его номер, по которому стоит звонить, чтобы связаться с контрагентом.

**invoice** = {invoice\_id, counteragentID, date\_time, type\_invoice, status}

{invoice\_id, counteragentID} -> {type\_invoice}

Каждая накладная идентифицируется по уникальному invoice\_id, первичному ключу. Атрибут type\_invoice функционально зависит от counteragentID и обозначают тип накладной, созданной определённым контрагентом.

**invoice\_detail** = {invoiceID, detailID, quantity}

{invoiceID, detailID} -> {quantity}

Каждая накладная идентифицируется по уникальному invoiceID, внешнему ключу, ссылающийся на таблицу invoice, а также по уникальному detailID, также внешнему ключу, который ссылается на таблицу details. Атрибут quantity функционально зависит от них и обозначает количество определённой детали по определённой накладной.

### 1.2.2 Нормализация отношений

### 1.2.2.1 Нормализация до первой нормальной формы (1НФ)

Отношение находится в первой нормальной форме (1НФ), если каждый его атрибут атомарен, т.е. содержит только одно значение, а не список значений.

Рассмотрим базовые отношения базы данных «Склад запчастей».

**warehouse (1)** = {warehouse\_id, warehouse\_number, address}

**room (1)** = {room\_id, room\_number, warehouseID} (с внешним ключом warehouseID)

**rack (1)** = {rack\_id, rack\_number, roomID} (с внешним ключом roomID)

**shelf (1)** = {shelf\_id, shelf\_number, rackID} (с внешним ключом rackID)

**details (1)** = {detail\_id, shelfID, weight, type\_detail} (с внешним ключом shelfID)

**counteragent (1)** = {counteragent\_id, counteragent\_name, contact\_person, phone\_number}

**invoice (1)** = {invoice\_id, counteragentID, date\_time, type\_invoice, status} (с внешним ключом counteragentID)

**invoice\_detail (1)** = {invoiceID, detailID, quantity} (составной ключ, состоящий из двух столбцов invoiceID и detailID)

**employee (1)** = {employee\_id, employee\_role, last\_name, first\_name, patronymic}

**invoice\_employee (1)** = {invoiceID, responsible, granted\_access, when\_granted} (внешние ключи invoiceID, responsible, granted\_access)

Все базовые отношения находятся в 1НФ.

### 1.2.2.2 Выбор первичных ключей и нормализация до второй нормальной формы (2НФ)

Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме и неключевые атрибуты зависят от первичного ключа. Рассмотрим подробнее таблицы базы данных «Склад запчастей».

**warehouse (2)** = {warehouse\_id, warehouse\_number, address}

В качестве первичного ключа был выбран warehouse\_id для уникальной идентификации складов в базе данных.

**room (2)** = {room\_id, room\_number, warehouseID} (с внешним ключом warehouseID)

В качестве первичного ключа был выбран room\_id для уникальной идентификации помещений в базе данных.

**rack (2)** = {rack\_id, rack\_number, roomID} (с внешним ключом roomID)

В качестве первичного ключа был выбран rack\_id для уникальной идентификации стеллажей в базе данных.

**shelf (2)** = {shelf\_id, shelf\_number, rackID} (с внешним ключом rackID)

В качестве первичного ключа был выбран shelf\_id для уникальной идентификации полок в базе данных.

**details (2)** = {detail\_id, shelfID, weight, type\_detail} (с внешним ключом shelfID)

В качестве первичного ключа был выбран detail\_id для уникальной идентификации деталей в базе данных.

**counteragent (2)** = {counteragent\_id, counteragent\_name, contact\_person, phone\_number}

В качестве первичного ключа был выбран counteragent\_id для уникальной идентификации контрагентов в базе данных.

**invoice (2)** = {invoice\_id, counteragentID, date\_time, type\_invoice, status} (с внешним ключом counteragentID)

В качестве первичного ключа был выбран invoice\_id для уникальной идентификации накладных в базе данных.

**employee (2)** = {employee\_id, employee\_role, last\_name, first\_name, patronymic}

В качестве первичного ключа был выбран employee\_id для уникальной идентификации сотрудников в базе данных.

**invoice\_employee (2)** = {invoiceID, responsible, granted\_access, when\_granted} (внешние ключи invoiceID, responsible, granted\_access)

В данной таблице первичным ключом является атрибут invoiceID, который является внешним ключом, ссылающийся на первичный ключ invoice\_id из таблицы invoice. Таким образом, неключевые атрибуты данной таблицы будут однозначно зависимы от номера накладной.

**invoice\_detail (2)** = {invoiceID, detailID, quantity} (составной ключ, состоящий из двух столбцов invoiceID и detailID)

В данной таблице первичным ключом, стоит взять составной ключ из атрибутов invoiceID и detailID, которые являются ссылками первичных ключей invoice\_id из таблицы invoice и detail\_id из таблицы details. Таким образом атрибут quantity однозначно зависит от обеих частей первичного ключа, так как количество деталей зависит от номера самой детали и от номера накладной. Следовательно, база данных приведена ко второй нормальной форме.

### 1.2.2.3 Нормализация до третьей нормальной формы (3НФ)

Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме и между атрибутами нет транзитивности. Рассмотрим подробнее таблицы базы данных «Склад запчастей».

**warehouse (3)** = {warehouse\_id, warehouse\_number, address}

Все атрибуты зависят от warehouse\_id, транзитивных зависимостей нет. **room (3)** = {room\_id, room\_number, warehouseID} (с внешним ключом warehouseID)

Все атрибуты зависят от room\_id.

**rack (3)** = {rack\_id, rack\_number, roomID} (с внешним ключом roomID)

Все атрибуты зависят от rack\_id.

**shelf (3)** = {shelf\_id, shelf\_number, rackID} (с внешним ключом rackID)

Все атрибуты зависят от shelf\_id.

**details (3)** = {detail\_id, shelfID, weight, type\_detail} (с внешним ключом shelfID)

Все атрибуты зависят от detail\_id.

**counteragent (3)** = {counteragent\_id, counteragent\_name, contact\_person, phone\_number}

Все атрибуты зависят от counteragent\_id.

**invoice (3)** = {invoice\_id, counteragentID, date\_time, type\_invoice, status} (с внешним ключом counteragentID)

Все атрибуты зависят от invoice\_id.

**employee (3)** = {employee\_id, employee\_role, last\_name, first\_name, patronymic}

Все атрибуты зависят от employee\_id.

Таблица **invoice** была разделена на две таблицы: **invoice\_employee** и **invoice\_detail**, так как в первоначальной таблице существовала транзитивность между атрибутами таблицы. Таким образом, с помощью декомпозиции таблицы **invoice** база данных «Склад запчастей» была приведена к третьей нормальной форме.

**invoice\_employee (3)** = {invoiceID, responsible, granted\_access, when\_granted} (внешние ключи invoiceID, responsible, granted\_access)

Все атрибуты зависят от составного ключа.

**invoice\_detail (3)** = {invoiceID, detailID, quantity} (составной ключ, состоящий из двух столбцов invoiceID и detailID)

Все атрибуты зависят от составного ключа.

В результате проделана нормализация отношения, а структура базы данных «Склад запчастей» приведена к виду, обеспечивающему минимальную логическую избыточность.

### 1.2.3 Логическое (даталогическое) проектирование базы данных

На этапе логического проектирования происходит разработка логической структуры будущей базы данных «Склад запчастей».

Основой для создания базы данных служит схема базы данных. Инфологическая модель данных, построенная в виде ER-диаграммы, должна быть преобразована в схему БД. Данное преобразование осуществляется путем сопоставления каждой сущности и каждой связи, имеющей атрибуты, с таблицами-отношениями.

В реляционной модели данных даталогическая модель представляет собой набор схем отношений. В ней указываются первичные ключи и связи между отношениями, которые реализованы с помощью внешних ключей.

Ранее были описаны связи между отношениями в базе данных склада запчастей. Результатом логического проектирования является даталогическая схема БД склада запчастей (Приложение Б – Даталогическая схема).

# **2 Физическая реализация модели БД склада запчастей**

## 2.1 Обоснование выбора системы управления базами данных (СУБД)

В современном мире существует широкий выбор систем управления базами данных (СУБД), каждая из которых обладает своими особенностями.

Oracle Database – это профессиональное решение для разработки сложных приложений. Его основой является высокопроизводительная база данных, способная хранить огромные объемы информации благодаря масштабируемости. Система эффективно обрабатывает запросы множества пользователей без потери производительности даже при резком росте нагрузки. Еще одним преимуществом Oracle является кроссплатформенность. Однако использование этой СУБД требует значительных финансовых затрат на серверное оборудование и техническую поддержку [3].

MySQL – популярная реляционная СУБД с открытым исходным кодом. В настоящее время разработкой и поддержкой MySQL занимается корпорация Oracle, которая приобрела права на эту систему после поглощения Sun Microsystems (ранее владевшей MySQL AB). Одним из ключевых достоинств MySQL является поддержка различных типов таблиц, включая MyISAM (с полнотекстовым поиском) и InnoDB (с транзакционной обработкой). Также в состав СУБД входит демонстрационный тип таблиц EXAMPLE, иллюстрирующий механизм создания новых форматов данных. Благодаря открытой лицензии (GPL) сообщество разработчиков активно расширяет функционал MySQL, добавляя новые возможности.

PostgreSQL – это объектно-реляционная СУБД с открытым исходным кодом. В отличие от чисто реляционных систем, PostgreSQL предлагает более гибкие и надежные механизмы работы с данными, включая поддержку их целостности. СУБД распространяется под либеральной BSD-лицензией, что позволяет свободно использовать и модифицировать ее код без юридических ограничений [5].

Для реализации физической модели была выбрана PostgreSQL, поскольку она предоставляет широкий набор инструментов для резервного копирования, мониторинга, миграции данных и администрирования (например, pgAdmin, DBeaver). Это упрощает интеграцию с другими платформами и делает процесс управления базой данных более эффективным.

## 2.2 Создание таблиц данных в среде целевой СУБД

SQL (язык структурированных запросов) — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных. [1]

В среде администрирования и разработки PostgreSQL с помощью SQL-запросов была создана база данных «Warehouse\_DB», а также все необходимые таблицы, такие как «warehouse», «room», «rack», «invoice» и другие. Данные SQL-запросы изображены на рисунках 1 и 2.

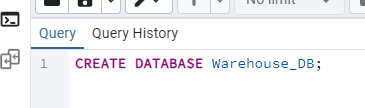


Рисунок 1 – Создание БД «Warehouse\_DB»

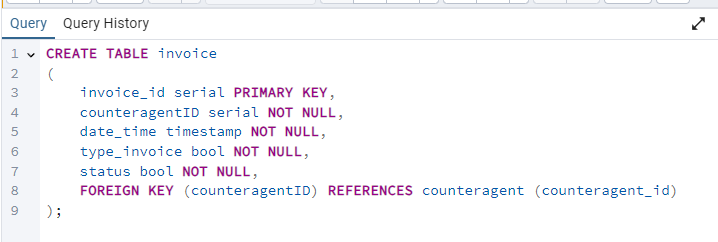


Рисунок 2 – Создание таблицы «invoice»

Для реализации темпоральности в базе данных используется таблица «log\_table», которая фиксирует изменения в других таблицах, включая тип изменения (INSERT, UPDATE, DELETE), идентификатор записи и значения до и после изменения. SQL-запрос для создания данной таблицы показан на рисунке 3.

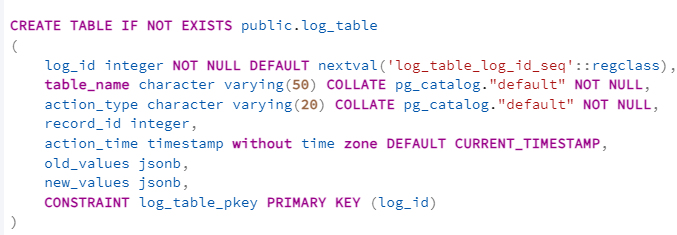


Рисунок 3 – Создание таблицы «log\_table»

Для автоматизации логирования изменений в таблицах созданы функции и триггеры. Например, функция log\_warehouse\_changes() записывает информацию об операциях в таблице «warehouse», код которой показан на рисунке 4.

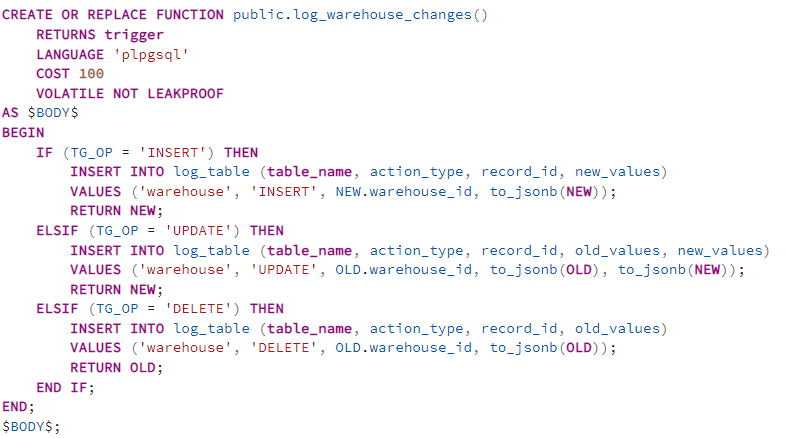


Рисунок 4 – Функция log\_warehouse\_changes()

Аналогичные функции созданы для таблиц «room», «rack», «shelf», «invoice», «invoice\_employee», «invoice\_detail», «details», «counteragent» и «employee». Эти функции вызываются соответствующими триггерами, которые срабатывают после выполнения операций INSERT, UPDATE или DELETE.

Для обеспечения целостности данных при удалении записей создана функция delete\_related\_data(), которая каскадно удаляет связанные данные. Например, при удалении склада удаляются все связанные комнаты, стеллажи, полки и детали. Этой функции соответствует триггер trg\_delete\_related\_data, который выполняется перед удалением записи в таблице «warehouse». Данная функция изображена на рисунке 5.

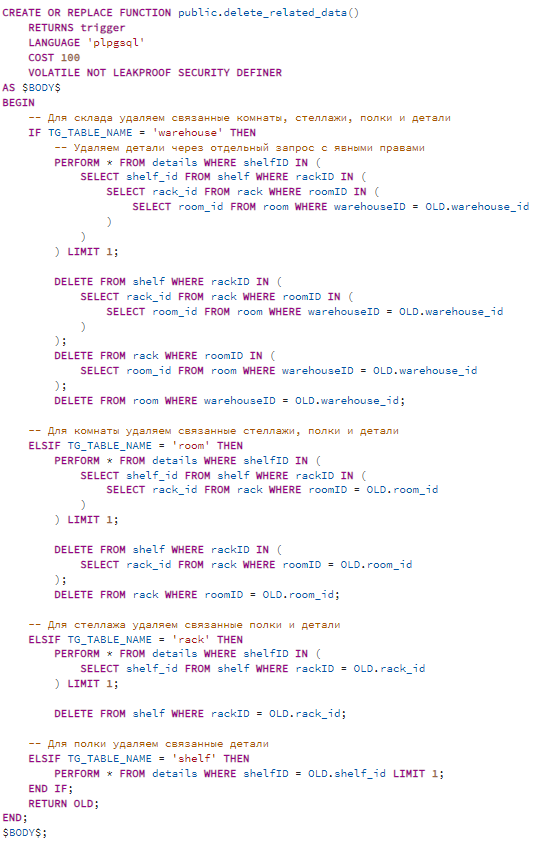


Рисунок 5 – Функция delete\_related\_data()

Для удобства работы с данными созданы представления «invoice\_details\_view» и «warehouse\_details\_view». Представление «invoice\_details\_view» объединяет информацию о накладных, включая данные о контрагенте, типе накладной, статусе, деталях и ответственных сотрудниках. Данные представления показаны на рисунках 6 и 7.

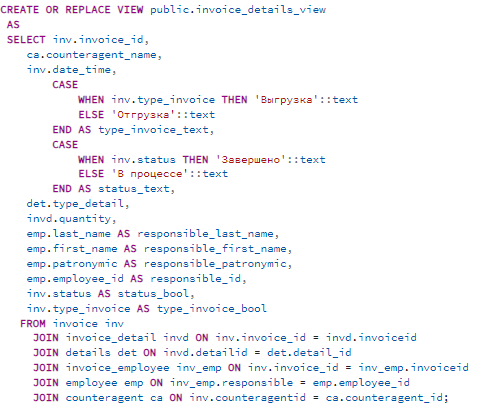


Рисунок 6 – Представление «invoice\_details\_view»

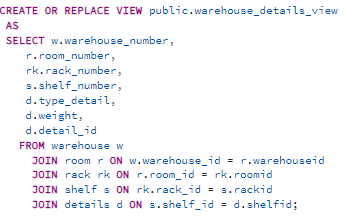


Рисунок 7 – Представление «warehouse\_details\_view»

Для поддержания актуальности данных в представлении «invoice\_details\_view» созданы функции insert\_invoice\_details\_view() и delete\_invoice\_details\_view(), которые управляют вставкой и удалением данных в связанных таблицах. Этим функциям соответствуют триггеры, срабатывающие при операциях с представлением.

Представление «warehouse\_details\_view» предоставляет информацию о деталях на складе, включая их расположение (склад, комната, стеллаж, полка). Для этого представления также созданы функции и триггеры: insert\_into\_warehouse\_details() для вставки данных, delete\_warehouse\_details() для удаления и update\_warehouse\_details\_view() для обновления. Эти функции обеспечивают согласованность данных в иерархии склада.

Для преобразования пользовательских данных в формат, удобный для хранения в базе, созданы функции convert\_text\_to\_boolean() и get\_employee\_id(), что изображены на рисунках 8 и 9.



Рисунок 8 – Функция convert\_text\_to\_boolean()

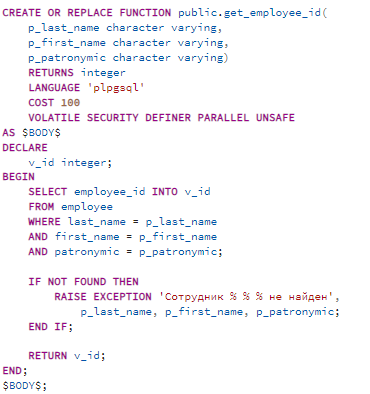


Рисунок 9 – Функция get\_employee\_id()

Функция convert\_text\_to\_boolean() преобразует текстовые значения (например, «завершено» или «выгрузка») в булевы значения (TRUE/FALSE), а функция get\_employee\_id() возвращает идентификатор сотрудника по его ФИО. Эти функции используются в триггерах и других функциях для автоматизации обработки данных.

Описанные в данном разделе функции и триггеры автоматизируют различные операции с таблицами базы данных склада запчастей и связанными данными, упрощая управление базой данных.

При помощи СУБД PostgreSQL созданы все необходимые таблицы для реализации БД склада запчастей, определены первичные ключи, проведены связи между таблицами, а также созданы таблицы и триггеры для осуществления темпоральности базы данных. Все таблицы были заполнены необходимыми данными.

Исходный код базы данных склада запчастей представлен в Приложении Д.

# **3 Разработка клиентского интерфейса для работы с базой данных**

Интерфейс для взаимодействия пользователей с базой данных склада запчастей должен соответствовать некоторым требованиям, а именно: быть интуитивно понятным и позволять каждому пользователю работать с базой в соответствии с его полномочиями. То есть, должны быть предусмотрены различные роли входа, для каждой роли должны быть реализованы свои функции. У владельца склада должен быть удобный способ добавления, изменения и удаления информации о сотрудниках склада, контрагентах и деталях, что хранятся на складе. Кладовщик должен иметь возможность изменения статуса накладной, а также изменения, добавления и удаления информации о деталях на складе. Менеджер склада должен иметь доступ к просмотру контрагентов, а также добавлению, изменению и удалению накладных.

## Выбор среды для разработки приложения

Выбор языка программирования и среды разработки является важным шагом в создании приложения для работы с СУБД PostgreSQL. Основными критериями для выбора являются: поддержка необходимых библиотек для работы с базой данных, удобство разработки и отладки, производительность и кроссплатформенность.

Одним из популярных языков, предлагающих мощные и стабильные инструменты для работы с этой СУБД, является Python. Он известен своей простотой и лаконичностью, что позволяет разработчикам быстро писать и тестировать код. Этот язык также поддерживает широкий набор инструментов для тестирования и отладки, таких как встроенный модуль unittest и сторонние библиотеки pytest. Средства отладки в Python позволяют эффективно отслеживать ошибки и находить узкие места в приложении, что критически важно при работе с базой данных.

Для работы с СУБД PostgreSQL Python предлагает множество библиотек. Например, библиотека psycopg2 предоставляет полный функционал для взаимодействия с PostgreSQL, включая транзакции, выполнение SQL-запросов и поддержку современных особенностей СУБД, таких как работа с JSONB или географическими данными (через PostGIS).

Python имеет достаточно высокий уровень производительности при работе с базами данных благодаря использованию эффективных драйверов и возможностей асинхронной обработки. Кроссплатформенность Python позволяет разворачивать приложения на различных операционных системах, что делает его идеальным выбором для проектов, рассчитанных на работу в разных окружениях (Linux, Windows, macOS).

Для обеспечения удобной разработки на Python рекомендуется использовать текстовый редактор Visual Studio Code (VS Code). Этот редактор обладает рядом преимуществ: интеграция с Python, плагины и расширяемость. Visual Studio Code также является кроссплатформенным инструментом, что обеспечивает возможность разработки и развертывания приложений на любой операционной системе без потери функциональности и удобства.

## Описание основных модулей и функций интерфейса

Для реализации графического интерфейса и взаимодействия с СУБД PostgreSQL было разработано несколько ключевых классов и методов.

Класс WindowApp является базовым классом для работы с окнами приложения. Он содержит метод auth\_window, который создает окно авторизации с полями для ввода логина и пароля, кнопкой входа и обработчиком событий. Метод create\_connection устанавливает соединение с базой данных PostgreSQL, используя переданные учетные данные. Метод start\_work обрабатывает попытку входа пользователя, проверяет учетные данные и либо открывает главное окно приложения (WarehouseApp), либо выводит сообщение об ошибке.

Класс WarehouseApp наследует от WindowApp и реализует основную логику работы приложения. В его конструкторе инициализируется главное окно, устанавливается соединение с базой данных, определяются права пользователя и создаются вкладки интерфейса. Метод determine\_user\_permissions проверяет права текущего пользователя на доступ к различным разделам приложения.

Приложение поддерживает работу с несколькими сущностями через отдельные вкладки: накладные (create\_invoice\_tab), склад (create\_warehouse\_tab), контрагенты (create\_counteragent\_tab) и сотрудники (create\_employee\_tab). Для каждой сущности реализованы CRUD-операции (добавление, редактирование, удаление, поиск).

Особое внимание уделено системе отката изменений. Методы save\_initial\_state, undo\_last\_operation и rollback\_to\_initial\_state позволяют сохранять состояние системы на момент входа пользователя и откатывать изменения, используя данные из таблицы логов. Для работы с резервными копиями реализованы методы create\_backup и restore\_from\_backup.

Для работы со структурой склада (склады, комнаты, стеллажи, полки) используется метод edit\_warehouse\_structure, который открывает отдельное окно с вкладками для каждой сущности. Методы add\_structure\_item, edit\_structure\_item и delete\_structure\_item обеспечивают редактирование структуры с проверкой целостности данных.

Приложение поддерживает систему поиска с сохранением условий (current\_search\_conditions) и возможностью сброса. Для каждой сущности реализованы отдельные методы поиска (search\_invoice, search\_warehouse\_item и т.д.).

Методы работы с накладными (add\_invoice, edit\_invoice, delete\_invoice) включают проверку доступности деталей на складе и ограничение на количество активных накладных у сотрудника (не более 5). Аналогичные методы реализованы для работы с другими сущностями (контрагентами, сотрудниками, деталями на складе).

Приложение завершает работу через метод logout, который корректно закрывает соединение с базой данных. Статусная строка в нижней части окна отображает текущее состояние системы и информацию о загруженных данных.

UML-диаграмма классов представлена в приложении B

Исходный код приложения представлен в Приложении Г.

## Пример работы графического интерфейса

После запуска программы на экране пользователя появляется окно для авторизации пользователя в системе, в котором необходимо ввести верные логин и пароль, иначе будет выдана ошибка, что показано на рисунках 10 и 11.

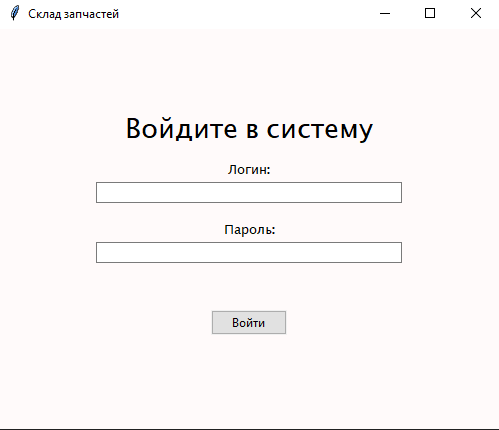


Рисунок 10 – Окно авторизации

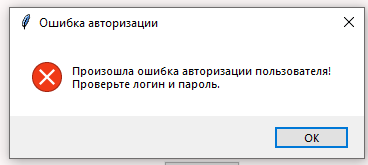
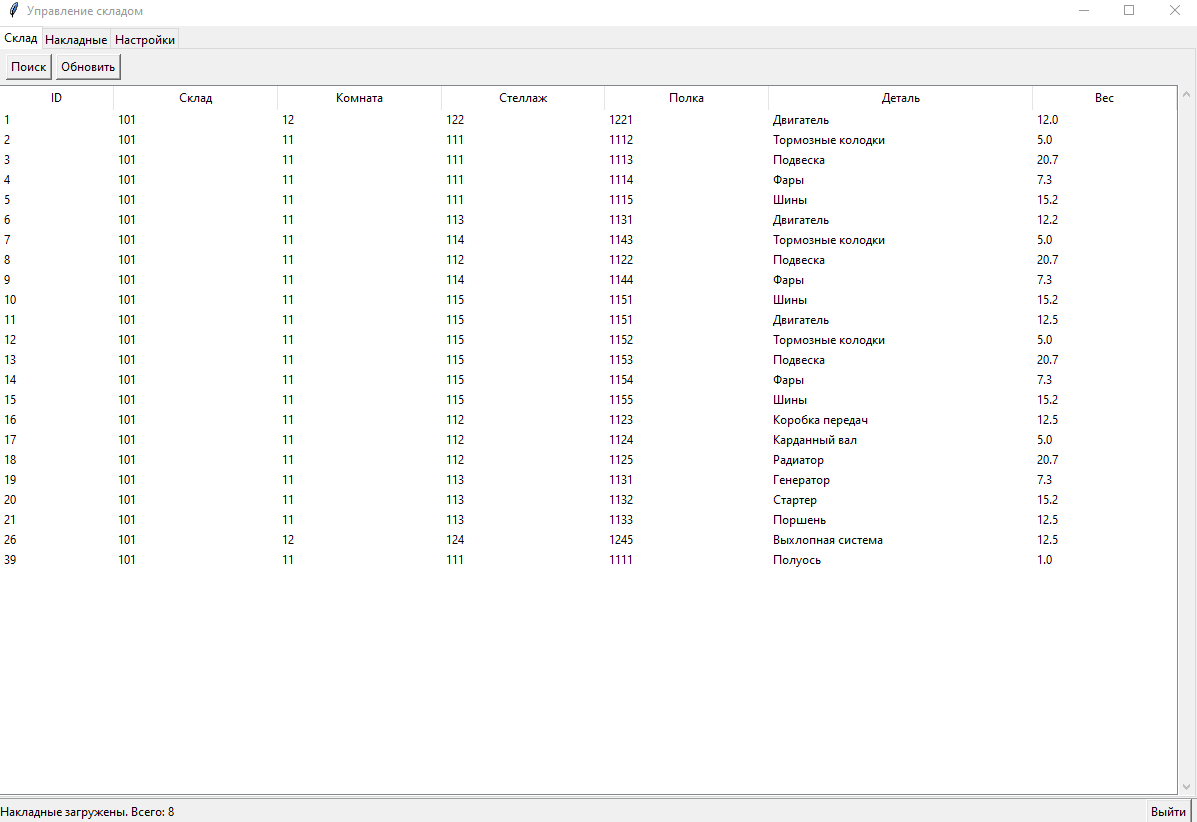


Рисунок 11 – Ошибка авторизации пользователя

Если пользователь ввёл верный логин и пароль, то на экране появиться главное окно приложения, что изображено на рисунке 12.

  
Рисунок 12 – Главное окно приложения

В верхней панели главного окна находятся вкладки таблиц базы данных, количество которых отличается от роли пользователя. Здесь же находятся кнопки «Поиск» и «Обновить». В нижней панели окна находится строка состояния приложения и кнопка «Выйти». В центральной части окна находится таблица с данными. Нажав на кнопку «Поиск», пользователь может заполнить от одного до нескольких полей для поиска необходимой записи в открытой таблице, что показано на рисунке 13 на примере поиска детали в таблице «Склад».

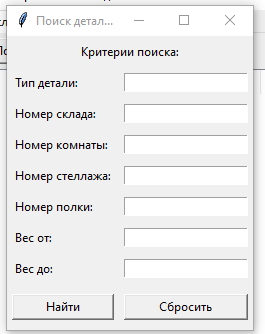


Рисунок 13 – Поиск детали в таблице «Склад»

Во вкладке «Настройки» для пользователей clerk (Кладовщик) и manager (Менеджер) находятся кнопки отката системы, что показаны на рисунке 14. Пользователь может откатить одну последнюю операцию или полностью откатить систему к началу текущего сеанса.

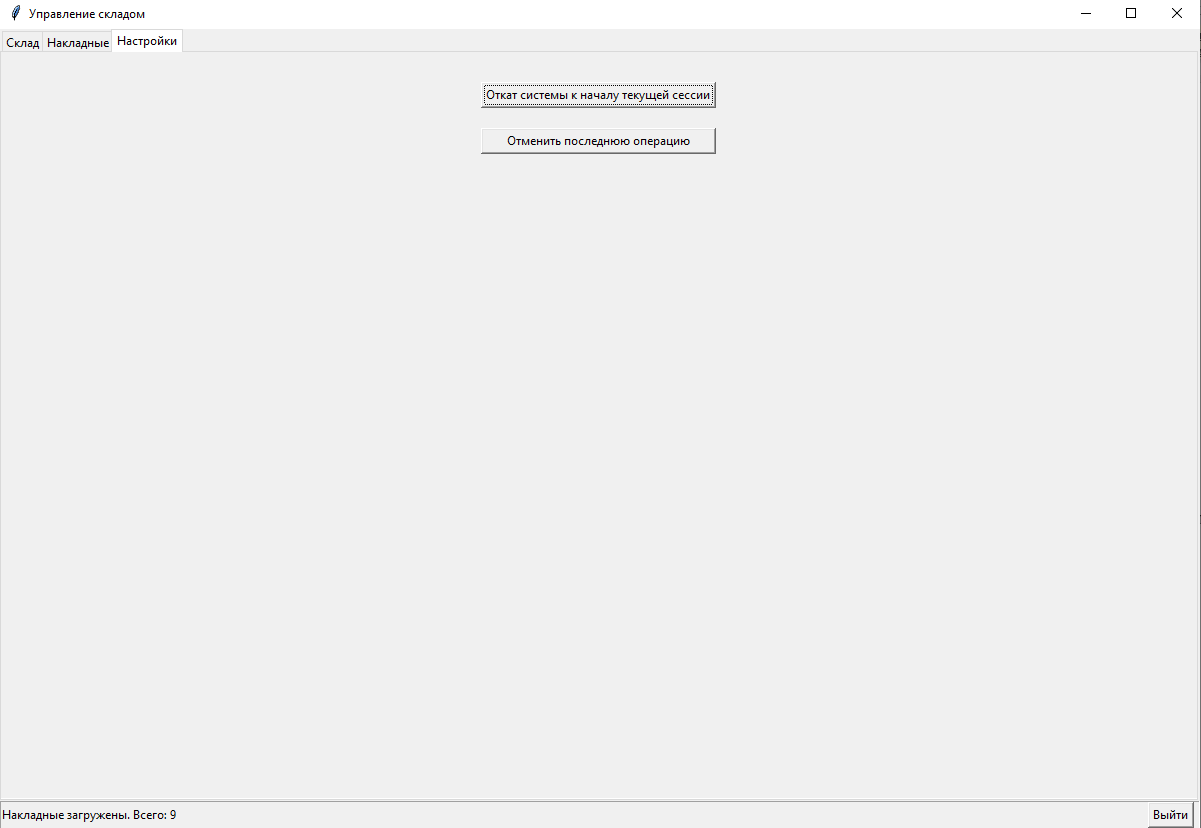


Рисунок 14 – Кнопки отката системы

Для пользователя owner (Владелец склада) находятся дополнительные кнопки резервного копирования, загрузки резервной копии и редактирования структуры склада, что показано на рисунке 15.

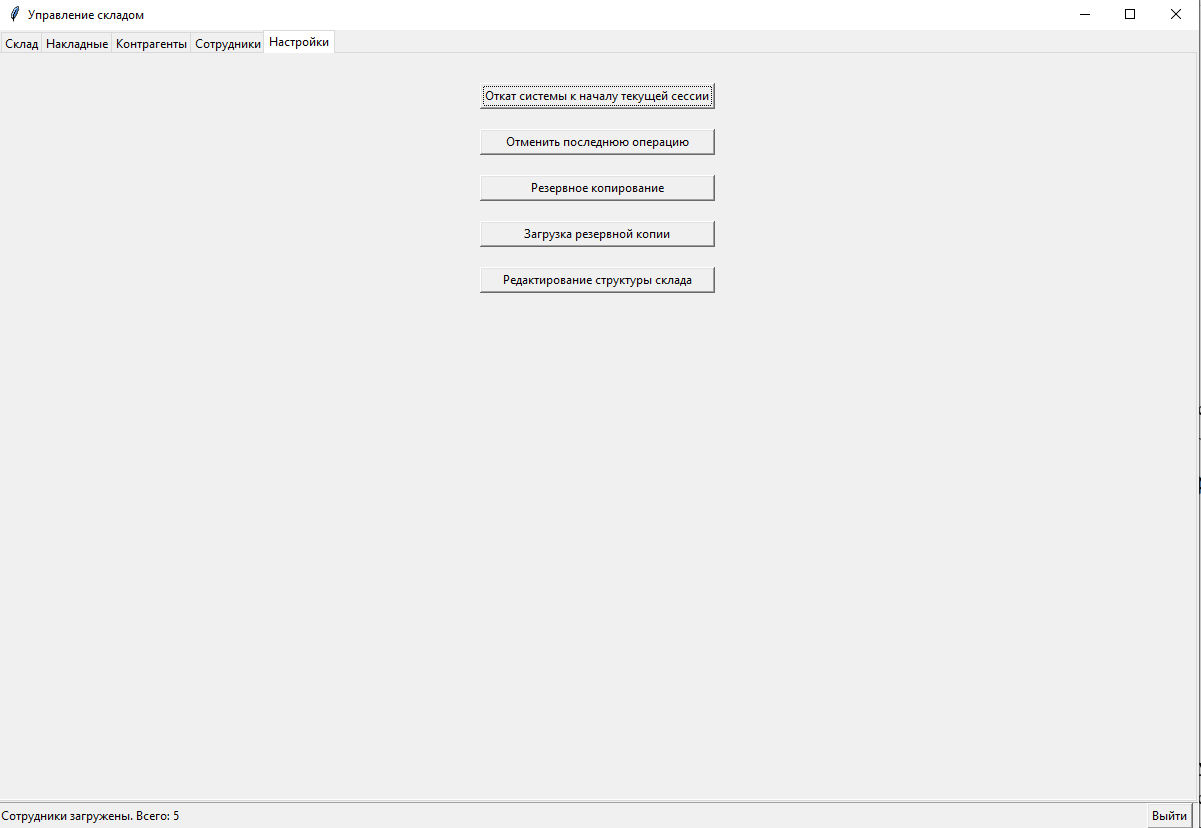


Рисунок 15 – Вкладка «Настройки» для пользователя owner

Нажав на кнопку «Редактирование структуры склада» главное окно приложения закроется и будет открыто новое окно, через которое пользователь сможет добавить, изменить или удалить существующие склады, комнаты, стеллажи и полки. Данное окно изображено на рисунке 16.

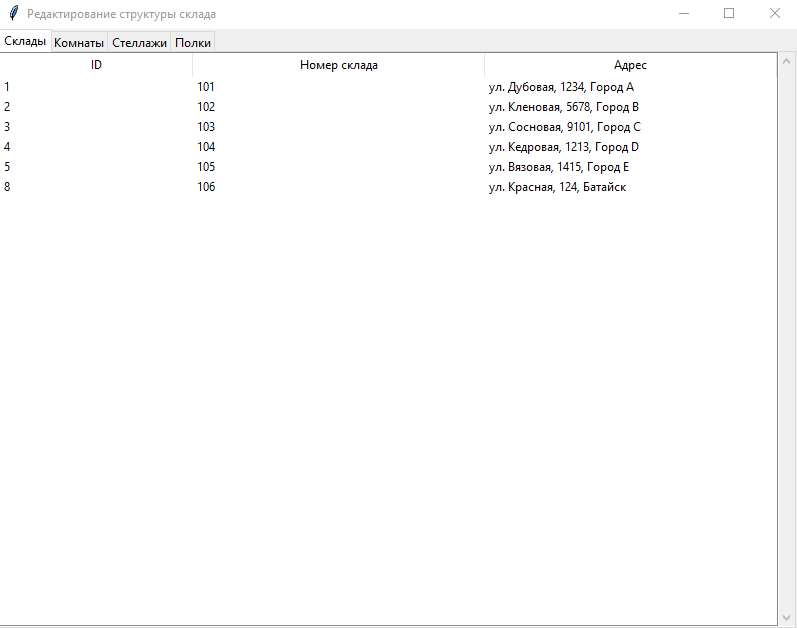


Рисунок 16 – Окно редактирования структуры склада

Нажав на крестик в правом верхнем углу окна на экране пользователя появиться главное окно приложение.

Нажав ПКМ по любой записи в любой таблице пользователю, будет представлен список возможных действий, что изображено на рисунке 17.

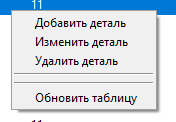


Рисунок 17 – Список доступных действий

Нажав на кнопку «Добавить» на экране пользователя появиться дополнительное окно с полями, каждое из которых обязательно к заполнению, что показано на рисунке 18.

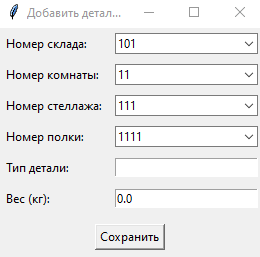


Рисунок 18 – Добавление новой записи

Если пользователь не заполнил все поля или неверно ввел данные, то на экране появиться ошибки, что изображены на рисунках 19 и 20 соответственно.

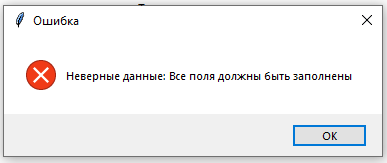


Рисунок 19 – Ошибка заполнения полей

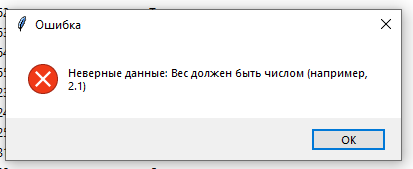


Рисунок 20 – Неверно введенные данные

Если пользователь ввел все данные правильно, то в таблице появиться новая запись.

Если пользователь нажмет на кнопку «Изменить», то на экране появиться дополнительное окно, в котором поля будут автоматически заполнены данными выделенной записи в таблице, что показано на рисунке 21.

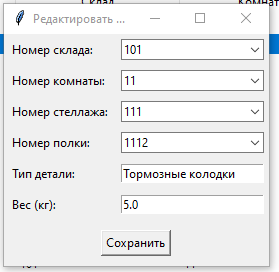


Рисунок 21 – Изменение записи в таблице

Если пользователь нажмет на кнопку «Удалить» или на клавишу Delete на клавиатуре, то на экране пользователя появиться уведомление, что изображено на рисунке 22.

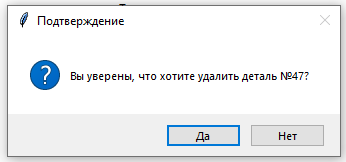


Рисунок 22 – Подтверждение удаления записи

Если пользователь нажмет «Нет», то запись не будет удалена. Если нажмет «Да», то на экране появиться уведомление об удачном или неудачном удалении записи, что показано на рисунках 23 и 24.

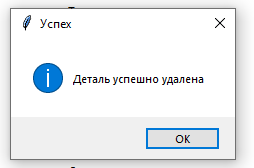


Рисунок 23 – Уведомление об успешном удалении записи

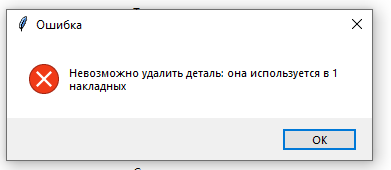


Рисунок 24 – Уведомление об ошибке удаления

Нажав на кнопку «Выйти» в нижней панели главного окна или на крестик в правом верхнем углу экрана, пользователя вернет в окно авторизации, а соединение с базой данных будет закрыто. Нажав на крестик в правом верхнем углу окна авторизации, пользователь закончит работу программы.

Исходный код программного средства представлен в Приложении В.

# **Заключение**

В рамках данной курсовой работы была поставлена задача проектирования базы данных для склада запчастей и разработки приложения для работы с ней.

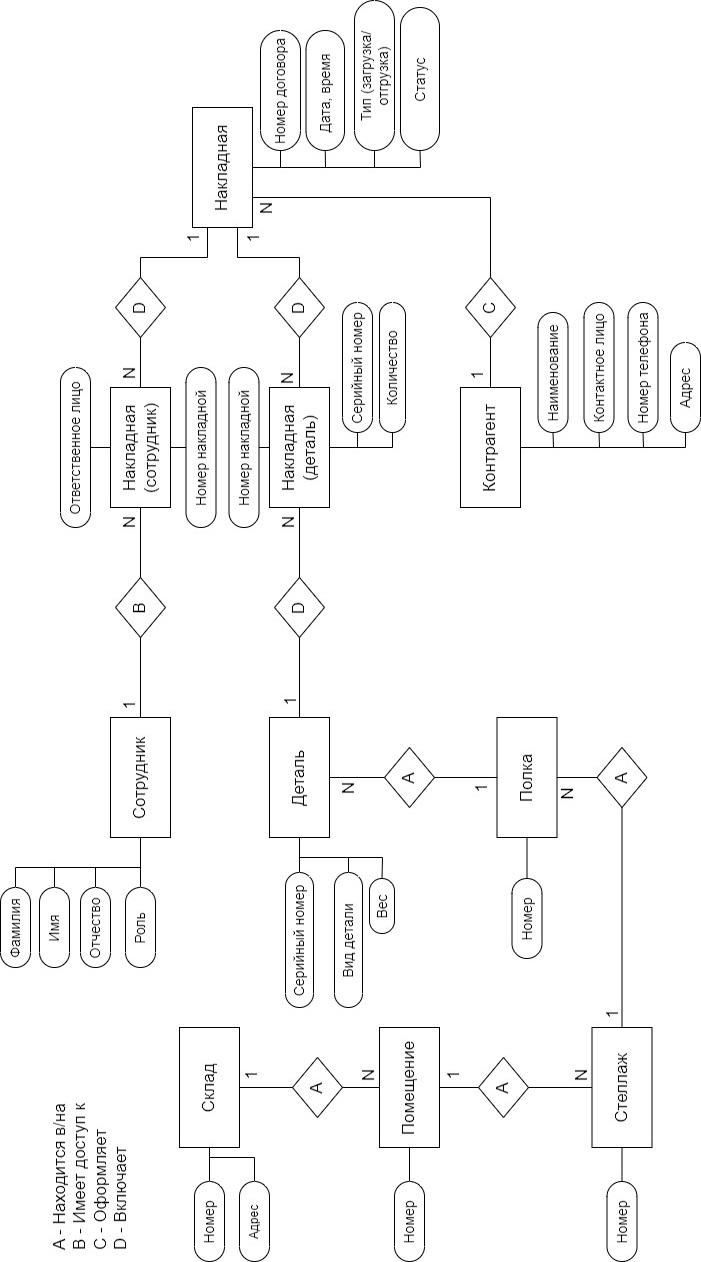
Для решения поставленной задачи была изучена предметная область, определены конечные пользователи, описаны объекты и их атрибуты. Также была разработана концептуальная модель в виде ER-диаграммы. Составлена даталогическая схема отношений. В ходе выполнения данной работы была изучена выбранная СУБД PostgreSQL, ее возможности и функционал.

В результате была спроектирована и реализована база данных склада запчастей и реализован набор интерфейсов для работы с ней. Так же были проведены необходимые тестирование и отладка разработанного программного обеспечения.

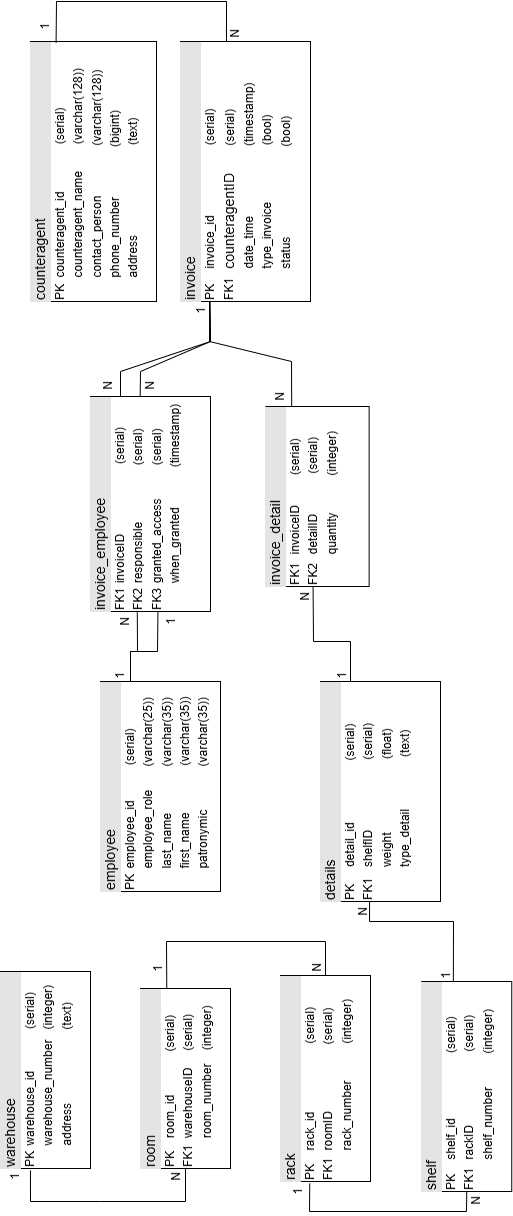
**Перечень используемых информационных источников**

1. Дейт, Кристофер – Введение в системы баз данных, 8-е издание / Пер. с англ. – М.: Издательский дом «Вильяме», 2005 – 1328 с.: ил. – Парал. тит. англ. – ISBN 5-8459-0788-8
2. Паттон Джефф. Пользовательские истории. Искусство гибкой разработки ПО. – СПб.: Питер, 2019. – 288 с.
3. Карпова Т.С. Базы данных: модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.
4. Конноли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
5. Глушаков С.В., Ломотько Д.В. – Базы данных: Учебный курс: ООО «Издательство АСТ», 2000 – 504 с. – ISBN 996-03-0992-9

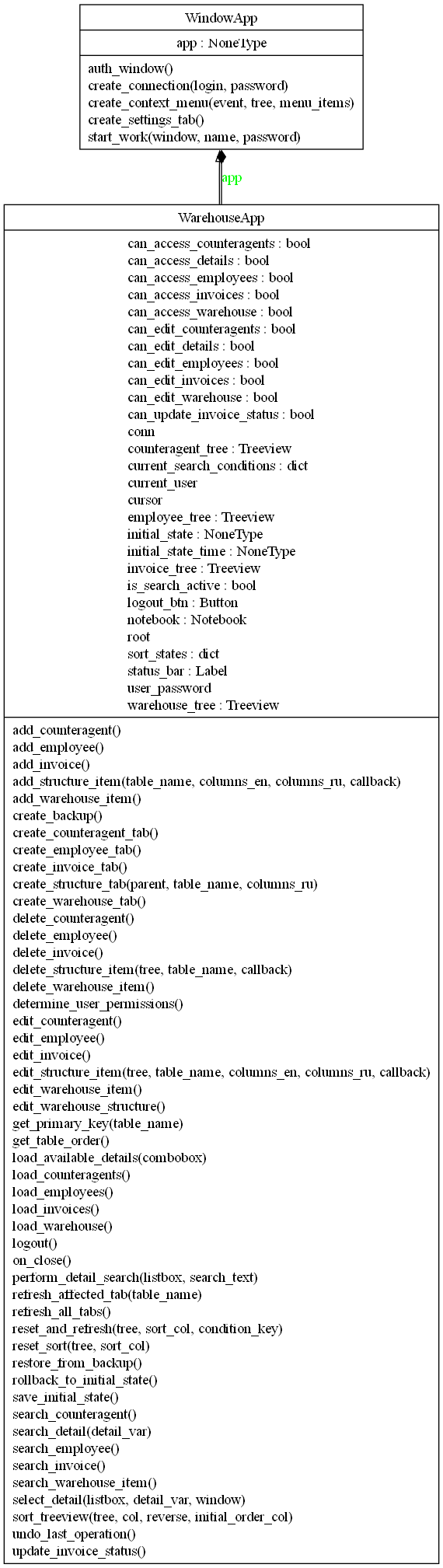
# **Приложение А ER-диаграмма**



# **Приложение Б Даталогическая схема**

****

**Приложение В UML-диаграмма классов программного средства**

****

# **Приложение Г Исходный код программного средства**

***Листинг 1 – Исходный код программного средства***

import psycopg2

from tkinter import \*

from tkinter import ttk, messagebox, filedialog

from datetime import datetime

import os

import subprocess

class WindowApp:

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

self.app = None

def auth\_window(self):

window = Tk()

window.geometry('%dx%d+%d+%d' % (500, 400,

(window.winfo\_screenwidth()/2) - (500/2),

(window.winfo\_screenheight()/2) - (400/2)))

window.title("Склад запчастей")

window.configure(background="#FFFAFA")

middle\_window\_x = 500 / 2

middle\_window\_y = 400 / 3

title\_start = ttk.Label(master=window, text="Войдите в систему",

font=("algerian", 20), background="#FFFAFA")

title\_start.place(x=middle\_window\_x, y=100, anchor="center")

title\_login = ttk.Label(text="Логин:", font=("algerian", 10), background="#FFFAFA")

title\_login.place(x=middle\_window\_x, y=140, anchor="center")

title\_password = ttk.Label(text="Пароль:", font=("algerian", 10), background="#FFFAFA")

title\_password.place(x=middle\_window\_x, y=200, anchor="center")

entry\_name = ttk.Entry(width=50)

entry\_name.place(x=middle\_window\_x, y=middle\_window\_y+30, anchor="center")

entry\_password = ttk.Entry(width=50, show="\*")

entry\_password.place(x=middle\_window\_x, y=middle\_window\_y+90, anchor="center")

btn\_in = ttk.Button(text="Войти", command=lambda: self.start\_work(window, entry\_name, entry\_password))

btn\_in.place(x=middle\_window\_x, y=middle\_window\_y+160, anchor="center")

window.mainloop()

def create\_connection(self, login, password):

try:

connection = psycopg2.connect(

host="127.0.0.1",

user=login,

password=password,

database="Warehouse\_DB"

)

print("[INFO] PostgreSQL connection open.")

return connection

except Exception as ex:

print(f"[CONNECTION ERROR] Failed to connect: {ex}")

return None

def start\_work(self, window, name, password):

login, password = name.get(), password.get()

print(f"[AUTH] Attempting login for user: {login}")

active\_user = self.create\_connection(login, password)

if active\_user is not None:

print("[AUTH] Login successful")

window.destroy()

main\_win = Tk()

self.app = WarehouseApp(main\_win, login, password, active\_user)

main\_win.mainloop()

else:

print("[AUTH] Login failed")

messagebox.showerror('Ошибка авторизации',

'Произошла ошибка авторизации пользователя! Проверьте логин и пароль.')

def create\_context\_menu(self, event, tree, menu\_items):

"""Создание контекстного меню для Treeview с добавлением кнопки обновления"""

item = tree.identify\_row(event.y)

if not item:

return

tree.selection\_set(item)

menu = Menu(self.root, tearoff=0)

for label, command in menu\_items:

if command:

menu.add\_command(label=label, command=command)

else:

menu.add\_separator()

menu.add\_separator()

if tree == self.invoice\_tree:

menu.add\_command(label="Обновить таблицу", command=lambda: [self.current\_search\_conditions.update({'invoice': None}), self.app.load\_invoices()])

elif tree == self.warehouse\_tree:

menu.add\_command(label="Обновить таблицу", command=lambda: [self.current\_search\_conditions.update({'warehouse': None}), self.app.load\_warehouse()])

elif tree == self.counteragent\_tree:

menu.add\_command(label="Обновить таблицу", command=lambda: [self.current\_search\_conditions.update({'counteragent': None}), self.app.load\_counteragents()])

elif tree == self.employee\_tree:

menu.add\_command(label="Обновить таблицу", command=lambda: [self.current\_search\_conditions.update({'employee': None}), self.app.load\_employees()])

try:

menu.tk\_popup(event.x\_root, event.y\_root)

finally:

menu.grab\_release()

def create\_settings\_tab(self):

"""Создает вкладку настроек для пользователя"""

tab = Frame(self.notebook)

self.notebook.add(tab, text="Настройки")

frame = Frame(tab)

frame.pack(pady=20)

btn\_rollback = Button(frame, text="Откат системы к началу текущей сессии",

command=self.rollback\_to\_initial\_state)

btn\_rollback.pack(pady=10, fill=X)

btn\_undo = Button(frame, text="Отменить последнюю операцию",

command=self.undo\_last\_operation)

btn\_undo.pack(pady=10, fill=X)

if self.current\_user == 'owner':

btn\_backup = Button(frame, text="Резервное копирование",

command=self.create\_backup)

btn\_backup.pack(pady=10, fill=X)

btn\_restore = Button(frame, text="Загрузка резервной копии",

command=self.restore\_from\_backup)

btn\_restore.pack(pady=10, fill=X)

btn\_edit\_warehouse = Button(frame, text="Редактирование структуры склада",

command=self.edit\_warehouse\_structure)

btn\_edit\_warehouse.pack(pady=10, fill=X)

class WarehouseApp(WindowApp):

def \_\_init\_\_(self, root, login, password, active\_user):

self.root = root

self.root.title("Управление складом")

self.root.geometry("1200x800")

self.current\_user = login

self.user\_password = password

self.initial\_state = None

self.is\_search\_active = False

self.sort\_states = {}

self.current\_search\_conditions = {

'invoice': None,

'warehouse': None,

'counteragent': None,

'employee': None

}

self.root.protocol("WM\_DELETE\_WINDOW", self.on\_close)

try:

if active\_user:

self.conn = active\_user

else:

try:

self.conn = psycopg2.connect(

host="127.0.0.1",

user=login,

password=password,

database="Warehouse\_DB"

)

print("[INFO] PostgreSQL connection open.")

except Exception as ex:

print(f"[ERROR] Connection failed: {ex}")

messagebox.showerror("Ошибка подключения", f"Не удалось подключиться к базе данных: {str(ex)}")

self.root.destroy()

return

self.cursor = self.conn.cursor()

self.determine\_user\_permissions()

self.save\_initial\_state()

self.status\_bar = Label(root, text=f"Вход выполнен как: {login} | Готово", bd=1, relief=SUNKEN, anchor=W)

self.status\_bar.pack(side=BOTTOM, fill=X)

self.logout\_btn = Button(self.status\_bar, text="Выйти", command=self.logout)

self.logout\_btn.pack(side=RIGHT, padx=5)

self.notebook = ttk.Notebook(root)

self.notebook.pack(fill=BOTH, expand=True)

if self.can\_access\_warehouse:

self.create\_warehouse\_tab()

if self.can\_access\_invoices:

self.create\_invoice\_tab()

if self.can\_access\_counteragents:

self.create\_counteragent\_tab()

if self.can\_access\_employees:

self.create\_employee\_tab()

self.create\_settings\_tab()

except psycopg2.Error as e:

print(f"[TRANSACTION ERROR] Initialization failed: {e}")

messagebox.showerror("Ошибка подключения", f"Не удалось подключиться к базе данных: {str(e)}")

self.root.destroy()

def logout(self):

"""Выход из системы и возврат к окну авторизации"""

if messagebox.askyesno("Подтверждение", "Вы уверены, что хотите выйти из системы?"):

if hasattr(self, 'conn') and self.conn:

self.conn.close()

print("[INFO] PostgreSQL connection closed.")

self.root.destroy()

self.auth\_window()

def on\_close(self):

"""Обработчик закрытия окна - выполняет выход из системы"""

self.logout()

def load\_available\_details(self, combobox):

"""Загружает список доступных деталей в комбобокс"""

try:

self.cursor.execute("""

SELECT DISTINCT type\_detail

FROM details

ORDER BY type\_detail

""")

details = [row[0] for row in self.cursor.fetchall()]

combobox['values'] = details

except Exception as e:

print(f"[ERROR] Failed to load details: {e}")

def search\_detail(self, detail\_var):

"""Открывает окно поиска деталей на складе"""

search\_window = Toplevel(self.root)

search\_window.title("Выбор детали")

search\_window.geometry("400x300")

search\_frame = Frame(search\_window)

search\_frame.pack(fill=X, padx=5, pady=5)

search\_var = StringVar()

search\_entry = Entry(search\_frame, textvariable=search\_var)

search\_entry.pack(side=LEFT, expand=True, fill=X, padx=(0, 5))

search\_btn = Button(search\_frame, text="Найти",

command=lambda: self.perform\_detail\_search(detail\_listbox, search\_var.get()))

search\_btn.pack(side=LEFT)

detail\_listbox = Listbox(search\_window)

detail\_listbox.pack(fill=BOTH, expand=True, padx=5, pady=5)

select\_btn = Button(search\_window, text="Выбрать",

command=lambda: self.select\_detail(detail\_listbox, detail\_var, search\_window))

select\_btn.pack(pady=5)

self.perform\_detail\_search(detail\_listbox, "")

search\_entry.bind("<Return>", lambda e: self.perform\_detail\_search(detail\_listbox, search\_var.get()))

detail\_listbox.bind("<Double-Button-1>", lambda e: self.select\_detail(detail\_listbox, detail\_var, search\_window))

def perform\_detail\_search(self, listbox, search\_text):

"""Выполняет поиск деталей по заданному тексту"""

listbox.delete(0, END)

try:

query = """

SELECT DISTINCT type\_detail

FROM details

WHERE type\_detail ILIKE %s

ORDER BY type\_detail

"""

self.cursor.execute(query, (f"%{search\_text}%",))

for row in self.cursor.fetchall():

listbox.insert(END, row[0])

except Exception as e:

print(f"[ERROR] Failed to search details: {e}")

def select\_detail(self, listbox, detail\_var, window):

"""Выбирает деталь из списка"""

selection = listbox.curselection()

if not selection:

messagebox.showwarning("Предупреждение", "Выберите деталь из списка")

return

selected\_detail = listbox.get(selection[0])

detail\_var.set(selected\_detail)

window.destroy()

def sort\_treeview(self, tree, col, reverse, initial\_order\_col=None):

"""Сортировка Treeview по столбцу"""

data = [(tree.set(item, col), item) for item in tree.get\_children('')]

try:

data.sort(key=lambda x: float(x[0]), reverse=reverse)

except ValueError:

data.sort(key=lambda x: x[0].lower(), reverse=reverse)

for index, (val, item) in enumerate(data):

tree.move(item, '', index)

tree.heading(col, command=lambda: self.sort\_treeview(tree, col, not reverse, initial\_order\_col))

if initial\_order\_col:

tree.heading(initial\_order\_col,

command=lambda: self.reset\_sort(tree, col))

tree.heading(col, text=col + (' ↓' if reverse else ' ↑'))

def reset\_sort(self, tree, sort\_col):

"""Сброс сортировки к первоначальному состоянию"""

for col in tree['columns']:

tree.heading(col, text=col)

items = list(tree.get\_children(''))

try:

items.sort(key=lambda x: int(tree.set(x, sort\_col)))

except ValueError:

items.sort(key=lambda x: tree.set(x, sort\_col).lower())

for index, item in enumerate(items):

tree.move(item, '', index)

for col in tree['columns']:

tree.heading(col, command=lambda c=col: self.sort\_treeview(tree, c, False, sort\_col))

def reset\_and\_refresh(self, tree, sort\_col, condition\_key):

"""Сброс сортировки и обновление данных"""

self.current\_search\_conditions.update({condition\_key: None})

self.reset\_sort(tree, sort\_col)

if tree == self.invoice\_tree:

self.load\_invoices()

elif tree == self.warehouse\_tree:

self.load\_warehouse()

elif tree == self.counteragent\_tree:

self.load\_counteragents()

elif tree == self.employee\_tree:

self.load\_employees()

def get\_table\_order(self):

"""Возвращает порядок таблиц для отката с учетом зависимостей"""

return [

'invoice\_detail',

'invoice\_employee',

'invoice',

'details',

'counteragent',

'employee',

'shelf',

'rack',

'room',

'warehouse',

'log\_table'

]

def refresh\_affected\_tab(self, table\_name):

"""Обновляет вкладку, соответствующую измененной таблице"""

if not self.is\_search\_active:

if table\_name in ['invoice', 'invoice\_detail', 'invoice\_employee'] and hasattr(self, 'invoice\_tree'):

self.load\_invoices()

elif table\_name == 'details' and hasattr(self, 'warehouse\_tree'):

self.load\_warehouse()

elif table\_name == 'counteragent' and hasattr(self, 'counteragent\_tree'):

self.load\_counteragents()

elif table\_name == 'employee' and hasattr(self, 'employee\_tree'):

self.load\_employees()

else:

pass

def refresh\_all\_tabs(self):

"""Обновляет все вкладки приложения"""

if not self.is\_search\_active:

if hasattr(self, 'invoice\_tree'):

self.load\_invoices()

if hasattr(self, 'warehouse\_tree'):

self.load\_warehouse()

if hasattr(self, 'counteragent\_tree'):

self.load\_counteragents()

if hasattr(self, 'employee\_tree'):

self.load\_employees()

else:

pass

def undo\_last\_operation(self):

"""Отменяет последнюю операцию с учетом зависимостей"""

try:

if not hasattr(self, 'initial\_state\_time') or not self.initial\_state\_time:

messagebox.showinfo("Информация", "Не удалось определить начало сессии")

return

self.cursor.execute("""

SELECT record\_id FROM log\_table

WHERE action\_type = 'UNDO'

AND action\_time >= %s

ORDER BY log\_id DESC

LIMIT 1

""", (self.initial\_state\_time,))

last\_undo = self.cursor.fetchone()

last\_undo\_id = last\_undo[0] if last\_undo else None

query = """

SELECT lt.log\_id, lt.table\_name, lt.action\_type,

lt.record\_id, lt.old\_values, lt.new\_values

FROM log\_table lt

WHERE lt.action\_type NOT IN ('UNDO', 'ROLLBACK')

AND lt.action\_time >= %s

"""

params = [self.initial\_state\_time]

if last\_undo\_id:

query += " AND lt.log\_id > %s"

params.append(last\_undo\_id)

query += """

ORDER BY

CASE table\_name

WHEN 'invoice\_detail' THEN 1

WHEN 'invoice\_employee' THEN 2

WHEN 'invoice' THEN 3

WHEN 'details' THEN 4

WHEN 'counteragent' THEN 5

WHEN 'employee' THEN 6

WHEN 'shelf' THEN 7

WHEN 'rack' THEN 8

WHEN 'room' THEN 9

WHEN 'warehouse' THEN 10

ELSE 11

END,

CASE action\_type

WHEN 'DELETE' THEN 1

WHEN 'UPDATE' THEN 2

WHEN 'INSERT' THEN 3

ELSE 4

END,

log\_id DESC

LIMIT 1

"""

self.cursor.execute(query, params)

last\_op = self.cursor.fetchone()

if not last\_op:

messagebox.showinfo("Информация", "Нет операций для отмены в текущей сессии")

return

log\_id, table, action, record\_id, old\_values, new\_values = last\_op

pk = self.get\_primary\_key(table)

if not pk:

messagebox.showerror("Ошибка", f"Не удалось определить первичный ключ для таблицы {table}")

return

self.cursor.execute("BEGIN")

try:

if action == 'INSERT':

if table in ['invoice\_detail', 'invoice\_employee']:

self.cursor.execute(f"""

SELECT invoiceid FROM {table}

WHERE {pk} = %s

""", (record\_id,))

invoice\_id = self.cursor.fetchone()[0]

self.cursor.execute("""

SELECT 1 FROM invoice

WHERE invoice\_id = %s

""", (invoice\_id,))

if not self.cursor.fetchone():

messagebox.showwarning("Предупреждение",

"Невозможно отменить операцию: связанная накладная уже удалена")

self.conn.rollback()

return

self.cursor.execute(f"""

DELETE FROM {table}

WHERE {pk} = %s

RETURNING \*

""", (record\_id,))

elif action == 'DELETE' and old\_values:

if table in ['invoice\_detail', 'invoice\_employee'] and 'invoiceid' in old\_values:

self.cursor.execute("""

SELECT 1 FROM invoice

WHERE invoice\_id = %s

""", (old\_values['invoiceid'],))

if not self.cursor.fetchone():

messagebox.showwarning("Предупреждение",

"Невозможно отменить операцию: связанная накладная не существует")

self.conn.rollback()

return

columns = ', '.join(old\_values.keys())

values = ', '.join(['%s'] \* len(old\_values))

self.cursor.execute(f"""

INSERT INTO {table} ({columns})

VALUES ({values})

RETURNING \*

""", list(old\_values.values()))

elif action == 'UPDATE' and old\_values:

set\_clause = ', '.join([f"{k} = %s" for k in old\_values.keys()])

self.cursor.execute(f"""

UPDATE {table}

SET {set\_clause}

WHERE {pk} = %s

RETURNING \*

""", list(old\_values.values()) + [record\_id])

import json

old\_values\_json = json.dumps(new\_values) if new\_values else None

new\_values\_json = json.dumps(old\_values) if old\_values else None

self.cursor.execute("""

INSERT INTO log\_table

(table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES (%s, 'UNDO', %s, %s, %s)

""", (table, log\_id, old\_values\_json, new\_values\_json))

self.conn.commit()

self.refresh\_affected\_tab(table)

messagebox.showinfo("Успех", "Последняя операция успешно отменена")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось отменить операцию: {str(e)}")

except Exception as e:

messagebox.showerror("Ошибка", f"Ошибка при отмене операции: {str(e)}")

def save\_initial\_state(self):

"""Сохраняет информацию о начальной точке отката"""

try:

self.cursor.execute("SELECT current\_timestamp")

self.initial\_state\_time = self.cursor.fetchone()[0]

print(f"[INFO] Saved initial state time: {self.initial\_state\_time}")

except Exception as e:

print(f"[ERROR] Failed to save initial state: {e}")

self.initial\_state\_time = None

def rollback\_to\_initial\_state(self):

"""Откатывает изменения с учетом зависимостей между таблицами"""

if not self.initial\_state\_time:

messagebox.showwarning("Предупреждение", "Не удалось определить начальное состояние сессии")

return

if messagebox.askyesno("Подтверждение",

"Вы уверены, что хотите откатить все изменения текущей сессии?"):

try:

self.cursor.execute("""

SELECT table\_name, action\_type, record\_id, old\_values

FROM log\_table

WHERE action\_time >= %s

AND log\_id > COALESCE((SELECT MAX(log\_id) FROM log\_table WHERE action\_type = 'ROLLBACK'), 0)

ORDER BY

CASE table\_name

WHEN 'invoice\_detail' THEN 1

WHEN 'invoice\_employee' THEN 2

WHEN 'invoice' THEN 3

WHEN 'details' THEN 4

WHEN 'counteragent' THEN 5

WHEN 'employee' THEN 6

WHEN 'shelf' THEN 7

WHEN 'rack' THEN 8

WHEN 'room' THEN 9

WHEN 'warehouse' THEN 10

ELSE 11

END,

CASE action\_type

WHEN 'DELETE' THEN 1

WHEN 'UPDATE' THEN 2

WHEN 'INSERT' THEN 3

ELSE 4

END,

log\_id DESC

""", (self.initial\_state\_time,))

changes = self.cursor.fetchall()

if not changes:

messagebox.showinfo("Информация", "Нет изменений для отката")

return

self.cursor.execute("BEGIN")

for table, action, record\_id, old\_values in changes:

try:

pk = self.get\_primary\_key(table)

if not pk:

print(f"[WARNING] Не удалось определить PK для таблицы {table}")

continue

if action == 'INSERT':

self.cursor.execute(f"""

SELECT 1 FROM {table}

WHERE {pk} = %s

""", (record\_id,))

if self.cursor.fetchone():

self.cursor.execute(f"""

DELETE FROM {table}

WHERE {pk} = %s

""", (record\_id,))

elif action == 'DELETE' and old\_values:

if table == 'invoice\_detail' or table == 'invoice\_employee':

if 'invoiceid' in old\_values:

self.cursor.execute("""

SELECT 1 FROM invoice

WHERE invoice\_id = %s

""", (old\_values['invoiceid'],))

if not self.cursor.fetchone():

print(f"[WARNING] Пропуск вставки в {table}, invoice не существует")

continue

self.cursor.execute(f"""

SELECT 1 FROM {table}

WHERE {pk} = %s

""", (record\_id,))

if not self.cursor.fetchone():

columns = ', '.join(old\_values.keys())

values = ', '.join(['%s'] \* len(old\_values))

self.cursor.execute(f"""

INSERT INTO {table} ({columns})

VALUES ({values})

""", list(old\_values.values()))

elif action == 'UPDATE' and old\_values:

set\_clause = ', '.join([f"{k} = %s" for k in old\_values.keys()])

self.cursor.execute(f"""

UPDATE {table}

SET {set\_clause}

WHERE {pk} = %s

""", list(old\_values.values()) + [record\_id])

except psycopg2.Error as e:

print(f"[WARNING] Failed to revert {action} on {table}.{record\_id}: {e}")

self.conn.rollback()

self.cursor.execute("SAVEPOINT rollback\_continue")

continue

self.cursor.execute("""

INSERT INTO log\_table (table\_name, action\_type, record\_id)

VALUES ('SYSTEM', 'ROLLBACK', %s)

""", (len(changes),))

self.conn.commit()

self.refresh\_all\_tabs()

messagebox.showinfo("Успех", "Система успешно откачена к началу сессии")

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось выполнить откат: {str(e)}")

self.conn.rollback()

def get\_primary\_key(self, table\_name):

"""Возвращает имя первичного ключа для таблицы с обработкой исключений"""

pk\_mapping = {

'warehouse': 'warehouse\_id',

'room': 'room\_id',

'rack': 'rack\_id',

'shelf': 'shelf\_id',

'details': 'detail\_id',

'counteragent': 'counteragent\_id',

'invoice': 'invoice\_id',

'invoice\_detail': 'invoiceID',

'employee': 'employee\_id',

'invoice\_employee': 'invoiceID',

'log\_table': 'log\_id'

}

if table\_name in pk\_mapping:

return pk\_mapping[table\_name]

try:

self.cursor.execute("""

SELECT a.attname

FROM pg\_index i

JOIN pg\_attribute a ON a.attrelid = i.indrelid AND a.attnum = ANY(i.indkey)

WHERE i.indrelid = %s::regclass AND i.indisprimary

""", (table\_name,))

result = self.cursor.fetchone()

return result[0] if result else None

except:

return None

def create\_backup(self):

"""Создает резервную копию базы данных"""

try:

backup\_file = filedialog.asksaveasfilename(

defaultextension=".backup",

filetypes=[("Backup files", "\*.backup"), ("All files", "\*.\*")],

title="Сохранить резервную копию как"

)

if not backup\_file:

return

env = os.environ.copy()

env['PGPASSWORD'] = '12345'

command = [

r'C:\Program Files\PostgreSQL\17\bin\pg\_dump',

'-h', '127.0.0.1',

'-U', 'postgres',

'-d', 'Warehouse\_DB',

'-F', 'c',

'-f', backup\_file

]

process = subprocess.Popen(command,

env=env,

stdout=subprocess.PIPE,

stderr=subprocess.PIPE)

stdout, stderr = process.communicate()

if process.returncode == 0:

messagebox.showinfo("Успех", f"Резервная копия успешно создана: {backup\_file}")

else:

error\_msg = stderr.decode('utf-8') if stderr else "Неизвестная ошибка"

messagebox.showerror("Ошибка", f"Не удалось создать резервную копию: {error\_msg}")

except Exception as e:

messagebox.showerror("Ошибка", f"Ошибка при создании резервной копии: {str(e)}")

def restore\_from\_backup(self):

"""Восстанавливает базу данных из резервной копии"""

try:

backup\_file = filedialog.askopenfilename(

filetypes=[("Backup files", "\*.backup"), ("All files", "\*.\*")],

title="Выберите файл резервной копии"

)

if not backup\_file:

return

if not messagebox.askyesno("Подтверждение",

"Вы уверены, что хотите восстановить базу данных из резервной копии? Все текущие данные будут потеряны!"):

return

self.conn.close()

env = os.environ.copy()

env['PGPASSWORD'] = '12345'

command = [

r'C:\Program Files\PostgreSQL\17\bin\pg\_restore',

'-h', '127.0.0.1',

'-U', 'postgres',

'-d', 'Warehouse\_DB',

'-c',

backup\_file

]

process = subprocess.Popen(command,

env=env,

stdout=subprocess.PIPE,

stderr=subprocess.PIPE)

stdout, stderr = process.communicate()

if process.returncode == 0:

messagebox.showinfo("Успех", "База данных успешно восстановлена из резервной копии")

self.root.destroy()

main\_win = Tk()

app = WarehouseApp(main\_win, self.current\_user, self.user\_password, None)

main\_win.mainloop()

else:

error\_msg = stderr.decode('utf-8') if stderr else "Неизвестная ошибка"

messagebox.showerror("Ошибка", f"Не удалось восстановить базу данных: {error\_msg}")

self.conn = psycopg2.connect(

host="127.0.0.1",

user=self.current\_user,

password=self.user\_password,

database="Warehouse\_DB"

)

self.cursor = self.conn.cursor()

except Exception as e:

messagebox.showerror("Ошибка", f"Ошибка при восстановлении из резервной копии: {str(e)}")

try:

self.conn = psycopg2.connect(

host="127.0.0.1",

user=self.current\_user,

password=self.user\_password,

database="Warehouse\_DB"

)

self.cursor = self.conn.cursor()

except:

self.root.destroy()

def create\_structure\_tab(self, parent, table\_name, columns\_ru):

"""Создает вкладку для редактирования структуры склада с русскими названиями"""

columns\_map = {

"warehouse": {

"Номер склада": "warehouse\_number",

"Адрес": "address"

},

"room": {

"Склад": "warehouseid",

"Номер комнаты": "room\_number"

},

"rack": {

"Комната": "roomid",

"Номер стеллажа": "rack\_number"

},

"shelf": {

"Стеллаж": "rackid",

"Номер полки": "shelf\_number"

}

}

columns\_en = [columns\_map[table\_name][col] for col in columns\_ru]

tree = ttk.Treeview(parent, columns=("id", \*columns\_ru), show="headings")

tree.heading("id", text="ID")

for col in columns\_ru:

tree.heading(col, text=col)

tree.column(col, width=150)

tree.column("id", width=50)

scroll = ttk.Scrollbar(parent, command=tree.yview)

scroll.pack(side=RIGHT, fill=Y)

tree.configure(yscrollcommand=scroll.set)

tree.pack(fill=BOTH, expand=True)

def load\_data(reset\_search=False):

"""Загружает данные с учетом текущих условий поиска или сбрасывает их"""

if reset\_search:

self.current\_search\_conditions[table\_name] = None

tree.delete(\*tree.get\_children())

if self.current\_search\_conditions.get(table\_name):

conditions = self.current\_search\_conditions[table\_name]['conditions']

params = self.current\_search\_conditions[table\_name]['params']

query = f"SELECT \* FROM {table\_name} WHERE " + " AND ".join(conditions) + " ORDER BY 1"

self.cursor.execute(query, params)

else:

self.cursor.execute(f"SELECT \* FROM {table\_name} ORDER BY 1")

for row in self.cursor.fetchall():

formatted\_row = [row[0]]

for i, col in enumerate(columns\_en):

if col.endswith("id"):

ref\_table = col.replace("id", "")

self.cursor.execute(f"SELECT {ref\_table}\_number FROM {ref\_table} WHERE {ref\_table}\_id = %s", (row[i+1],))

ref\_number = self.cursor.fetchone()

formatted\_row.append(ref\_number[0] if ref\_number else "N/A")

else:

formatted\_row.append(row[i+1])

tree.insert("", END, values=formatted\_row)

def search\_items():

search\_window = Toplevel(self.root)

search\_window.title(f"Поиск в {table\_name}")

Label(search\_window, text="Критерии поиска:").grid(row=0, column=0, columnspan=2, pady=5)

search\_vars = []

for i, col in enumerate(columns\_ru):

Label(search\_window, text=f"{col}:").grid(row=i+1, column=0, padx=5, pady=5, sticky=W)

var = StringVar()

if columns\_en[i].endswith("id"):

ref\_table = columns\_en[i].replace("id", "")

self.cursor.execute(f"SELECT {ref\_table}\_number FROM {ref\_table} ORDER BY {ref\_table}\_number")

options = [str(row[0]) for row in self.cursor.fetchall()]

combo = ttk.Combobox(search\_window, textvariable=var, values=options)

combo.grid(row=i+1, column=1, padx=5, pady=5, sticky=EW)

else:

Entry(search\_window, textvariable=var).grid(row=i+1, column=1, padx=5, pady=5, sticky=EW)

search\_vars.append(var)

def perform\_search():

try:

conditions = []

params = []

for i, col in enumerate(columns\_en):

search\_text = search\_vars[i].get()

if search\_text:

if col.endswith("id"):

ref\_table = col.replace("id", "")

self.cursor.execute(f"""

SELECT {ref\_table}\_id FROM {ref\_table}

WHERE {ref\_table}\_number = %s

""", (search\_text,))

ref\_id = self.cursor.fetchone()

if ref\_id:

conditions.append(f"{col} = %s")

params.append(ref\_id[0])

else:

conditions.append(f"{col}::text LIKE %s")

params.append(f"%{search\_text}%")

self.current\_search\_conditions[table\_name] = {

'conditions': conditions,

'params': params

}

load\_data(False)

search\_window.destroy()

except Exception as e:

messagebox.showerror("Ошибка", f"Ошибка поиска: {str(e)}")

self.conn.rollback()

Button(search\_window, text="Найти", command=perform\_search).grid(

row=len(columns\_ru)+1, column=0, padx=5, pady=10, sticky=EW)

Button(search\_window, text="Сбросить", command=lambda: [self.current\_search\_conditions.update({table\_name: None}), load\_data(False), search\_window.destroy()]).grid(

row=len(columns\_ru)+1, column=1, padx=5, pady=10, sticky=EW)

menu = Menu(parent, tearoff=0)

menu.add\_command(label="Добавить", command=lambda: self.add\_structure\_item(table\_name, columns\_en, columns\_ru, lambda: load\_data(False)))

menu.add\_command(label="Изменить", command=lambda: self.edit\_structure\_item(tree, table\_name, columns\_en, columns\_ru, lambda: load\_data(False)))

menu.add\_command(label="Удалить", command=lambda: self.delete\_structure\_item(tree, table\_name, lambda: load\_data(False)))

menu.add\_separator()

menu.add\_command(label="Найти", command=search\_items)

menu.add\_separator()

menu.add\_command(label="Обновить список", command=lambda: load\_data(False))

menu.add\_command(label="Обновить таблицу", command=lambda: load\_data(True))

def show\_context\_menu(event):

item = tree.identify\_row(event.y)

if item:

tree.selection\_set(item)

menu.post(event.x\_root, event.y\_root)

tree.bind("<Button-3>", show\_context\_menu)

load\_data(False)

def edit\_warehouse\_structure(self):

"""Редактирование структуры склада (склады, комнаты, стеллажи, полки)"""

edit\_window = Toplevel(self.root)

edit\_window.title("Редактирование структуры склада")

edit\_window.geometry("800x600")

self.root.withdraw()

def on\_edit\_window\_close():

self.root.deiconify()

edit\_window.destroy()

edit\_window.protocol("WM\_DELETE\_WINDOW", on\_edit\_window\_close)

notebook = ttk.Notebook(edit\_window)

notebook.pack(fill=BOTH, expand=True)

warehouse\_tab = Frame(notebook)

notebook.add(warehouse\_tab, text="Склады")

self.create\_structure\_tab(warehouse\_tab, "warehouse", ["Номер склада", "Адрес"])

room\_tab = Frame(notebook)

notebook.add(room\_tab, text="Комнаты")

self.create\_structure\_tab(room\_tab, "room", ["Склад", "Номер комнаты"])

rack\_tab = Frame(notebook)

notebook.add(rack\_tab, text="Стеллажи")

self.create\_structure\_tab(rack\_tab, "rack", ["Комната", "Номер стеллажа"])

shelf\_tab = Frame(notebook)

notebook.add(shelf\_tab, text="Полки")

self.create\_structure\_tab(shelf\_tab, "shelf", ["Стеллаж", "Номер полки"])

def add\_structure\_item(self, table\_name, columns\_en, columns\_ru, callback):

"""Добавляет новый элемент структуры склада с проверкой уникальности"""

add\_window = Toplevel(self.root)

add\_window.title(f"Добавить в {table\_name}")

entries = []

labels = []

for i, (col\_en, col\_ru) in enumerate(zip(columns\_en, columns\_ru)):

labels.append(Label(add\_window, text=col\_ru))

labels[-1].grid(row=i, column=0, padx=5, pady=5, sticky=W)

if col\_en.endswith("id"):

ref\_table = col\_en.replace("id", "")

self.cursor.execute(f"SELECT {ref\_table}\_id, {ref\_table}\_number FROM {ref\_table}")

options = [f"{num} (ID: {id})" for id, num in self.cursor.fetchall()]

var = StringVar()

combo = ttk.Combobox(add\_window, textvariable=var, values=options)

combo.grid(row=i, column=1, padx=5, pady=5, sticky=EW)

entries.append((var, True, col\_en))

else:

entry = Entry(add\_window)

entry.grid(row=i, column=1, padx=5, pady=5, sticky=EW)

entries.append((entry, False, col\_en))

def save\_item():

try:

values = []

parent\_id = None

if table\_name == "warehouse":

warehouse\_number = entries[0][0].get()

self.cursor.execute("SELECT 1 FROM warehouse WHERE warehouse\_number = %s", (warehouse\_number,))

address = entries[1][0].get()

self.cursor.execute("SELECT 1 FROM warehouse WHERE address = %s", (address,))

if self.cursor.fetchone():

raise ValueError(f"Склад с номером {warehouse\_number} уже существует")

if self.cursor.fetchone():

raise ValueError(f"Склад по адресу {address} уже существует")

elif table\_name == "room":

room\_number = entries[1][0].get()

warehouse\_id = entries[0][0].get().split("(ID: ")[1].replace(")", "").strip()

self.cursor.execute("""

SELECT 1 FROM room

WHERE room\_number = %s AND warehouseID = %s

""", (room\_number, warehouse\_id))

if self.cursor.fetchone():

raise ValueError(f"Комната с номером {room\_number} уже существует в этом складе")

elif table\_name == "rack":

rack\_number = entries[1][0].get()

room\_id = entries[0][0].get().split("(ID: ")[1].replace(")", "").strip()

self.cursor.execute("""

SELECT 1 FROM rack

WHERE rack\_number = %s AND roomID = %s

""", (rack\_number, room\_id))

if self.cursor.fetchone():

raise ValueError(f"Стеллаж с номером {rack\_number} уже существует в этой комнате")

elif table\_name == "shelf":

shelf\_number = entries[1][0].get()

rack\_id = entries[0][0].get().split("(ID: ")[1].replace(")", "").strip()

self.cursor.execute("""

SELECT 1 FROM shelf

WHERE shelf\_number = %s AND rackID = %s

""", (shelf\_number, rack\_id))

if self.cursor.fetchone():

raise ValueError(f"Полка с номером {shelf\_number} уже существует на этом стеллаже")

for entry, is\_combo, col\_en in entries:

if is\_combo:

val = entry.get().split("(ID: ")[1].replace(")", "").strip()

values.append(val)

else:

values.append(entry.get())

columns\_str = ", ".join(columns\_en)

placeholders = ", ".join(["%s"] \* len(columns\_en))

self.cursor.execute(

f"INSERT INTO {table\_name} ({columns\_str}) VALUES ({placeholders})",

values

)

self.conn.commit()

add\_window.destroy()

messagebox.showinfo("Успех", "Запись успешно добавлена")

except ValueError as ve:

messagebox.showerror("Ошибка", str(ve))

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось добавить запись: {str(e)}")

Button(add\_window, text="Сохранить", command=save\_item).grid(

row=len(columns\_en), column=0, columnspan=2, pady=10)

def edit\_structure\_item(self, tree, table\_name, columns\_en, columns\_ru, callback):

"""Редактирует элемент структуры склада без автоматического обновления"""

selected = tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите запись для редактирования")

return

item = tree.item(selected[0])

item\_id = item['values'][0]

edit\_window = Toplevel(self.root)

edit\_window.title(f"Изменить запись в {table\_name}")

self.cursor.execute(f"SELECT \* FROM {table\_name} WHERE {table\_name}\_id = %s", (item\_id,))

current\_data = self.cursor.fetchone()

entries = []

labels = []

for i, (col\_en, col\_ru) in enumerate(zip(columns\_en, columns\_ru)):

labels.append(Label(edit\_window, text=col\_ru))

labels[-1].grid(row=i, column=0, padx=5, pady=5, sticky=W)

if col\_en.endswith("id"):

ref\_table = col\_en.replace("id", "")

self.cursor.execute(f"SELECT {ref\_table}\_id, {ref\_table}\_number FROM {ref\_table}")

options = [f"{num} (ID: {id})" for id, num in self.cursor.fetchall()]

self.cursor.execute(f"SELECT {ref\_table}\_number FROM {ref\_table} WHERE {ref\_table}\_id = %s", (current\_data[i+1],))

current\_ref = self.cursor.fetchone()

current\_value = f"{current\_ref[0]} (ID: {current\_data[i+1]})" if current\_ref else ""

var = StringVar(value=current\_value)

combo = ttk.Combobox(edit\_window, textvariable=var, values=options)

combo.grid(row=i, column=1, padx=5, pady=5, sticky=EW)

entries.append((var, True, col\_en))

else:

entry = Entry(edit\_window)

entry.insert(0, str(current\_data[i+1]))

entry.grid(row=i, column=1, padx=5, pady=5, sticky=EW)

entries.append((entry, False, col\_en))

def save\_changes():

try:

if table\_name == "warehouse":

new\_address = entries[1][0].get()

self.cursor.execute("""

SELECT 1 FROM warehouse

WHERE address = %s AND warehouse\_id != %s

""", (new\_address, item\_id))

warehouse\_number = entries[0][0].get()

self.cursor.execute("""

SELECT 1 FROM warehouse

WHERE warehouse\_number = %s AND warehouse\_id != %s

""", (warehouse\_number, item\_id))

if self.cursor.fetchone():

raise ValueError(f"Склад по адресу {new\_address} уже существует")

if self.cursor.fetchone():

raise ValueError(f"Склад с номером {warehouse\_number} уже существует")

elif table\_name == "room":

room\_number = entries[1][0].get()

warehouse\_id = entries[0][0].get().split("(ID: ")[1].replace(")", "").strip()

self.cursor.execute("""

SELECT 1 FROM room

WHERE room\_number = %s AND warehouseID = %s AND room\_id != %s

""", (room\_number, warehouse\_id, item\_id))

if self.cursor.fetchone():

raise ValueError(f"Комната с номером {room\_number} уже существует в этом складе")

elif table\_name == "rack":

rack\_number = entries[1][0].get()

room\_id = entries[0][0].get().split("(ID: ")[1].replace(")", "").strip()

self.cursor.execute("""

SELECT 1 FROM rack

WHERE rack\_number = %s AND roomID = %s AND rack\_id != %s

""", (rack\_number, room\_id, item\_id))

if self.cursor.fetchone():

raise ValueError(f"Стеллаж с номером {rack\_number} уже существует в этой комнате")

elif table\_name == "shelf":

shelf\_number = entries[1][0].get()

rack\_id = entries[0][0].get().split("(ID: ")[1].replace(")", "").strip()

self.cursor.execute("""

SELECT 1 FROM shelf

WHERE shelf\_number = %s AND rackID = %s AND shelf\_id != %s

""", (shelf\_number, rack\_id, item\_id))

if self.cursor.fetchone():

raise ValueError(f"Полка с номером {shelf\_number} уже существует на этом стеллаже")

values = []

for entry, is\_combo, col\_en in entries:

if is\_combo:

val = entry.get().split("(ID: ")[1].replace(")", "").strip()

values.append(val)

else:

values.append(entry.get())

set\_clause = ", ".join([f"{col} = %s" for col in columns\_en])

self.cursor.execute(

f"UPDATE {table\_name} SET {set\_clause} WHERE {table\_name}\_id = %s",

values + [item\_id]

)

self.conn.commit()

edit\_window.destroy()

messagebox.showinfo("Успех", "Запись успешно обновлена")

except ValueError as ve:

messagebox.showerror("Ошибка", str(ve))

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось обновить запись: {str(e)}")

Button(edit\_window, text="Сохранить", command=save\_changes).grid(

row=len(columns\_en), column=0, columnspan=2, pady=10)

def delete\_structure\_item(self, tree, table\_name, callback):

"""Удаляет элемент структуры склада с обработкой ошибок"""

selected = tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите запись для удаления")

return

item = tree.item(selected[0])

item\_id = item['values'][0]

try:

if table\_name == "warehouse":

self.cursor.execute("""

SELECT 1 FROM details WHERE shelfID IN (

SELECT shelf\_id FROM shelf WHERE rackID IN (

SELECT rack\_id FROM rack WHERE roomID IN (

SELECT room\_id FROM room WHERE warehouseID = %s

)

)

) LIMIT 1

""", (item\_id,))

elif table\_name == "room":

self.cursor.execute("""

SELECT 1 FROM details WHERE shelfID IN (

SELECT shelf\_id FROM shelf WHERE rackID IN (

SELECT rack\_id FROM rack WHERE roomID = %s

)

) LIMIT 1

""", (item\_id,))

elif table\_name == "rack":

self.cursor.execute("""

SELECT 1 FROM details WHERE shelfID IN (

SELECT shelf\_id FROM shelf WHERE rackID = %s

) LIMIT 1

""", (item\_id,))

elif table\_name == "shelf":

self.cursor.execute("""

SELECT 1 FROM details WHERE shelfID = %s LIMIT 1

""", (item\_id,))

has\_details = self.cursor.fetchone()

if has\_details:

if not messagebox.askyesno(

"Подтверждение",

"Это действие удалит все связанные детали. Продолжить?"

):

return

self.conn.commit()

messagebox.showinfo("Успех", "Запись успешно удалена")

except psycopg2.Error as e:

self.conn.rollback()

error\_msg = f"Ошибка удаления: {str(e)}"

if "нет доступа" in str(e):

error\_msg += "\n\nНедостаточно прав для выполнения операции. Обратитесь к администратору."

messagebox.showerror("Ошибка", error\_msg)

def determine\_user\_permissions(self):

self.can\_access\_invoices = False

self.can\_access\_warehouse = False

self.can\_access\_counteragents = False

self.can\_access\_employees = False

self.can\_access\_details = False

self.can\_edit\_invoices = False

self.can\_edit\_warehouse = False

self.can\_edit\_counteragents = False

self.can\_edit\_employees = False

self.can\_edit\_details = False

self.can\_update\_invoice\_status = False

try:

self.cursor.execute("""

SELECT table\_name, privilege\_type

FROM information\_schema.table\_privileges

WHERE grantee = %s

""", ('warehouse\_'+self.current\_user,))

permissions = self.cursor.fetchall()

for table, privilege in permissions:

if table in ('invoice', 'invoice\_details\_view'):

self.can\_access\_invoices = True

if privilege in ('INSERT', 'UPDATE', 'DELETE'):

self.can\_edit\_invoices = True

if table in ('warehouse', 'room', 'rack', 'shelf', 'warehouse\_details\_view'):

self.can\_access\_warehouse = True

if privilege in ('INSERT', 'UPDATE', 'DELETE'):

self.can\_edit\_warehouse = True

if table == 'counteragent':

self.can\_access\_counteragents = True

if privilege in ('INSERT', 'UPDATE', 'DELETE'):

self.can\_edit\_counteragents = True

if table == 'employee':

self.can\_access\_employees = True

if privilege in ('INSERT', 'UPDATE', 'DELETE'):

self.can\_edit\_employees = True

if table == 'details':

self.can\_access\_details = True

if privilege in ('INSERT', 'UPDATE', 'DELETE'):

self.can\_edit\_details = True

if self.current\_user == 'clerk':

self.can\_edit\_invoices = False

self.can\_update\_invoice\_status = True

except psycopg2.Error as e:

print(f"[PERMISSION ERROR] Failed to check permissions: {e}")

self.status\_bar.config(text=f"Ошибка проверки прав доступа: {str(e)}")

except psycopg2.Error as e:

print(f"[VIEW ERROR] Failed to access view: {e}")

messagebox.showerror("Ошибка", "Не удалось получить доступ к данным. Проверьте права доступа.")

def create\_invoice\_tab(self):

"""Создание вкладки для работы с накладными с учетом прав доступа"""

tab = Frame(self.notebook)

self.notebook.add(tab, text="Накладные")

button\_frame = Frame(tab)

button\_frame.pack(fill=X, padx=5, pady=5)

search\_btn = Button(button\_frame, text="Поиск", command=self.search\_invoice)

search\_btn.pack(side=LEFT, padx=2)

refresh\_btn = Button(button\_frame, text="Обновить",

command=lambda: self.reset\_and\_refresh(self.invoice\_tree, "ID", 'invoice'))

refresh\_btn.pack(side=LEFT, padx=2)

columns = ("ID", "Контрагент", "Дата", "Тип", "Статус", "Деталь", "Кол-во", "Ответственный")

self.invoice\_tree = ttk.Treeview(tab, columns=columns, show="headings")

for col in columns:

self.invoice\_tree.heading(col, text=col,

command=lambda c=col: self.sort\_treeview(

self.invoice\_tree, c, False, "ID"))

self.invoice\_tree.column(col, width=100)

self.invoice\_tree.column("ID", width=50)

self.invoice\_tree.column("Контрагент", width=150)

self.invoice\_tree.column("Дата", width=120)

self.invoice\_tree.column("Тип", width=100)

self.invoice\_tree.column("Статус", width=100)

self.invoice\_tree.column("Деталь", width=150)

self.invoice\_tree.column("Кол-во", width=70)

self.invoice\_tree.column("Ответственный", width=150)

scroll = ttk.Scrollbar(tab, command=self.invoice\_tree.yview)

scroll.pack(side=RIGHT, fill=Y)

self.invoice\_tree.configure(yscrollcommand=scroll.set)

self.invoice\_tree.pack(fill=BOTH, expand=True)

menu\_items = []

if self.can\_edit\_invoices:

menu\_items.extend([

("Добавить накладную", self.add\_invoice),

("Изменить накладную", self.edit\_invoice),

("Удалить накладную", self.delete\_invoice),

(None, None)

])

elif self.can\_update\_invoice\_status:

menu\_items.extend([

("Обновить статус", self.update\_invoice\_status),

(None, None)

])

self.invoice\_tree.bind("<Button-3>", lambda e: self.create\_context\_menu(e, self.invoice\_tree, menu\_items))

self.invoice\_tree.bind("<Delete>", lambda e: self.delete\_invoice())

self.load\_invoices()

def update\_invoice\_status(self):

selected = self.invoice\_tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите накладную для обновления статуса")

return

try:

item = self.invoice\_tree.item(selected[0])

invoice\_id = item['values'][0]

current\_status = item['values'][4] == 'Завершено'

status\_window = Toplevel(self.root)

status\_window.title("Обновление статуса накладной")

Label(status\_window, text="Новый статус:").pack(padx=5, pady=5)

status\_var = StringVar()

status\_var.set("Завершено" if current\_status else "В процессе")

status\_combobox = ttk.Combobox(status\_window, textvariable=status\_var,

values=["В процессе", "Завершено"])

status\_combobox.pack(padx=5, pady=5)

def save\_status():

try:

new\_status = status\_var.get() == "Завершено"

if not new\_status:

self.cursor.execute("""

SELECT responsible FROM invoice\_employee

WHERE invoiceid = %s

""", (invoice\_id,))

employee\_id = self.cursor.fetchone()

if employee\_id:

employee\_id = employee\_id[0]

self.cursor.execute("""

SELECT COUNT(\*) FROM invoice\_employee

WHERE responsible = %s

AND invoiceid IN (SELECT invoice\_id FROM invoice WHERE status = FALSE)

AND invoiceid != %s

""", (employee\_id, invoice\_id))

active\_invoices\_count = self.cursor.fetchone()[0]

if active\_invoices\_count >= 5:

raise ValueError(

f"У сотрудника уже {active\_invoices\_count} активных накладных. "

"Максимум - 5. Нельзя изменить статус."

)

self.cursor.execute("""

UPDATE invoice SET status = %s WHERE invoice\_id = %s

""", (new\_status, invoice\_id))

self.conn.commit()

self.load\_invoices()

status\_window.destroy()

messagebox.showinfo("Успех", "Статус накладной обновлен")

except ValueError as ve:

messagebox.showerror("Ошибка", str(ve))

except Exception as e:

print(f"[UPDATE ERROR] Failed to update invoice status: {e}")

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось обновить статус: {str(e)}")

Button(status\_window, text="Сохранить", command=save\_status).pack(pady=10)

except Exception as e:

print(f"[STATUS UPDATE ERROR] Initial error: {e}")

messagebox.showerror("Ошибка", f"Ошибка при обновлении статуса: {str(e)}")

def create\_warehouse\_tab(self):

"""Создание вкладки для работы со складом с учетом прав доступа"""

tab = Frame(self.notebook)

self.notebook.add(tab, text="Склад")

button\_frame = Frame(tab)

button\_frame.pack(fill=X, padx=5, pady=5)

search\_btn = Button(button\_frame, text="Поиск", command=self.search\_warehouse\_item)

search\_btn.pack(side=LEFT, padx=2)

refresh\_btn = Button(button\_frame, text="Обновить",

command=lambda: self.reset\_and\_refresh(self.warehouse\_tree, "ID", 'warehouse'))

refresh\_btn.pack(side=LEFT, padx=2)

columns = ("ID", "Склад", "Комната", "Стеллаж", "Полка", "Деталь", "Вес")

self.warehouse\_tree = ttk.Treeview(tab, columns=columns, show="headings")

for col in columns:

self.warehouse\_tree.heading(col, text=col,

command=lambda c=col: self.sort\_treeview(

self.warehouse\_tree, c, False, "ID"))

self.warehouse\_tree.column(col, width=100)

self.warehouse\_tree.column("ID", width=50)

self.warehouse\_tree.column("Склад", width=100)

self.warehouse\_tree.column("Комната", width=100)

self.warehouse\_tree.column("Стеллаж", width=100)

self.warehouse\_tree.column("Полка", width=100)

self.warehouse\_tree.column("Деталь", width=200)

self.warehouse\_tree.column("Вес", width=80)

scroll = ttk.Scrollbar(tab, command=self.warehouse\_tree.yview)

scroll.pack(side=RIGHT, fill=Y)

self.warehouse\_tree.configure(yscrollcommand=scroll.set)

self.warehouse\_tree.pack(fill=BOTH, expand=True)

menu\_items = []

if self.can\_edit\_warehouse:

menu\_items.extend([

("Добавить деталь", self.add\_warehouse\_item),

("Изменить деталь", self.edit\_warehouse\_item),

("Удалить деталь", self.delete\_warehouse\_item),

(None, None)

])

self.warehouse\_tree.bind("<Button-3>", lambda e: self.create\_context\_menu(e, self.warehouse\_tree, menu\_items))

self.warehouse\_tree.bind("<Delete>", lambda e: self.delete\_warehouse\_item())

self.load\_warehouse()

def create\_counteragent\_tab(self):

"""Создание вкладки для работы с контрагентами"""

tab = Frame(self.notebook)

self.notebook.add(tab, text="Контрагенты")

button\_frame = Frame(tab)

button\_frame.pack(fill=X, padx=5, pady=5)

search\_btn = Button(button\_frame, text="Поиск", command=self.search\_counteragent)

search\_btn.pack(side=LEFT, padx=2)

refresh\_btn = Button(button\_frame, text="Обновить",

command=lambda: self.reset\_and\_refresh(self.counteragent\_tree, "ID", 'counteragent'))

refresh\_btn.pack(side=LEFT, padx=2)

columns = ("ID", "Название", "Контакт", "Телефон", "Адрес")

self.counteragent\_tree = ttk.Treeview(tab, columns=("ID", "Название", "Контакт", "Телефон", "Адрес"), show="headings")

for col in columns:

self.counteragent\_tree.heading(col, text=col,

command=lambda c=col: self.sort\_treeview(

self.counteragent\_tree, c, False, "ID"))

self.counteragent\_tree.column(col, width=100)

self.counteragent\_tree.heading("ID", text="ID")

self.counteragent\_tree.heading("Название", text="Название")

self.counteragent\_tree.heading("Контакт", text="Контактное лицо")

self.counteragent\_tree.heading("Телефон", text="Телефон")

self.counteragent\_tree.heading("Адрес", text="Адрес")

self.counteragent\_tree.column("ID", width=50)

self.counteragent\_tree.column("Название", width=200)

self.counteragent\_tree.column("Контакт", width=150)

self.counteragent\_tree.column("Телефон", width=120)

self.counteragent\_tree.column("Адрес", width=250)

scroll = ttk.Scrollbar(tab, command=self.counteragent\_tree.yview)

scroll.pack(side=RIGHT, fill=Y)

self.counteragent\_tree.configure(yscrollcommand=scroll.set)

self.counteragent\_tree.pack(fill=BOTH, expand=True)

menu\_items = []

if self.can\_edit\_counteragents:

menu\_items.extend([

("Добавить контрагента", self.add\_counteragent),

("Изменить контрагента", self.edit\_counteragent),

("Удалить контрагента", self.delete\_counteragent),

(None, None)

])

self.counteragent\_tree.bind("<Button-3>", lambda e: self.create\_context\_menu(e, self.counteragent\_tree, menu\_items))

self.counteragent\_tree.bind("<Delete>", lambda e: self.delete\_counteragent())

self.load\_counteragents()

def create\_employee\_tab(self):

"""Создание вкладки для работы с сотрудниками"""

tab = Frame(self.notebook)

self.notebook.add(tab, text="Сотрудники")

button\_frame = Frame(tab)

button\_frame.pack(fill=X, padx=5, pady=5)

search\_btn = Button(button\_frame, text="Поиск", command=self.search\_employee)

search\_btn.pack(side=LEFT, padx=2)

refresh\_btn = Button(button\_frame, text="Обновить",

command=lambda: self.reset\_and\_refresh(self.employee\_tree, "ID", 'employee'))

refresh\_btn.pack(side=LEFT, padx=2)

columns = ("ID", "Роль", "Фамилия", "Имя", "Отчество")

self.employee\_tree = ttk.Treeview(tab, columns=("ID", "Роль", "Фамилия", "Имя", "Отчество"), show="headings")

for col in columns:

self.employee\_tree.heading(col, text=col,

command=lambda c=col: self.sort\_treeview(

self.employee\_tree, c, False, "ID"))

self.employee\_tree.column(col, width=100)

self.employee\_tree.heading("ID", text="ID")

self.employee\_tree.heading("Роль", text="Роль")

self.employee\_tree.heading("Фамилия", text="Фамилия")

self.employee\_tree.heading("Имя", text="Имя")

self.employee\_tree.heading("Отчество", text="Отчество")

self.employee\_tree.column("ID", width=50)

self.employee\_tree.column("Роль", width=150)

self.employee\_tree.column("Фамилия", width=120)

self.employee\_tree.column("Имя", width=120)

self.employee\_tree.column("Отчество", width=120)

scroll = ttk.Scrollbar(tab, command=self.employee\_tree.yview)

scroll.pack(side=RIGHT, fill=Y)

self.employee\_tree.configure(yscrollcommand=scroll.set)

self.employee\_tree.pack(fill=BOTH, expand=True)

menu\_items = []

if self.can\_edit\_employees:

menu\_items.extend([

("Добавить сотрудника", self.add\_employee),

("Изменить сотрудника", self.edit\_employee),

("Удалить сотрудника", self.delete\_employee),

(None, None)

])

self.employee\_tree.bind("<Button-3>", lambda e: self.create\_context\_menu(e, self.employee\_tree, menu\_items))

self.employee\_tree.bind("<Delete>", lambda e: self.delete\_employee())

self.load\_employees()

def load\_invoices(self):

"""Загрузка данных о накладных"""

try:

self.invoice\_tree.delete(\*self.invoice\_tree.get\_children())

if self.current\_search\_conditions['invoice']:

conditions = self.current\_search\_conditions['invoice']['conditions']

params = self.current\_search\_conditions['invoice']['params']

query = """

SELECT

invoice\_id,

counteragent\_name,

date\_time,

type\_invoice\_text,

status\_text,

type\_detail,

quantity,

responsible\_last\_name || ' ' ||

responsible\_first\_name || ' ' ||

COALESCE(responsible\_patronymic, '') as responsible

FROM invoice\_details\_view

WHERE """ + " AND ".join(conditions) + """

ORDER BY invoice\_id

"""

self.cursor.execute(query, params)

else:

self.cursor.execute("""

SELECT

invoice\_id,

counteragent\_name,

date\_time,

type\_invoice\_text,

status\_text,

type\_detail,

quantity,

responsible\_last\_name || ' ' ||

responsible\_first\_name || ' ' ||

COALESCE(responsible\_patronymic, '') as responsible

FROM invoice\_details\_view

ORDER BY invoice\_id

""")

for row in self.cursor.fetchall():

self.invoice\_tree.insert("", END, values=row)

count = len(self.invoice\_tree.get\_children())

if self.current\_search\_conditions['invoice']:

self.status\_bar.config(text=f"Найдено накладных: {count}")

else:

self.status\_bar.config(text=f"Накладные загружены. Всего: {count}")

except Exception as e:

print(f"[LOAD ERROR] Failed to load invoices: {e}")

self.status\_bar.config(text=f"Ошибка загрузки накладных: {str(e)}")

self.conn.rollback()

def search\_invoice(self):

"""Поиск накладных по различным критериям"""

search\_window = Toplevel(self.root)

search\_window.title("Поиск накладных")

Label(search\_window, text="Критерии поиска:").grid(row=0, column=0, columnspan=2, pady=5)

Label(search\_window, text="ID накладной:").grid(row=1, column=0, padx=5, pady=5, sticky=W)

id\_var = StringVar()

id\_entry = Entry(search\_window, textvariable=id\_var)

id\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Контрагент:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

counteragent\_var = StringVar()

counteragent\_entry = Entry(search\_window, textvariable=counteragent\_var)

counteragent\_entry.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Дата (ГГГГ-ММ-ДД):").grid(row=3, column=0, padx=5, pady=5, sticky=W)

date\_from\_var = StringVar()

date\_from\_entry = Entry(search\_window, textvariable=date\_from\_var)

date\_from\_entry.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Тип накладной:").grid(row=5, column=0, padx=5, pady=5, sticky=W)

type\_var = StringVar()

type\_combobox = ttk.Combobox(search\_window, textvariable=type\_var,

values=["", "Отгрузка", "Выгрузка"])

type\_combobox.grid(row=5, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Статус:").grid(row=6, column=0, padx=5, pady=5, sticky=W)

status\_var = StringVar()

status\_combobox = ttk.Combobox(search\_window, textvariable=status\_var,

values=["", "В процессе", "Завершено"])

status\_combobox.grid(row=6, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Деталь:").grid(row=7, column=0, padx=5, pady=5, sticky=W)

detail\_var = StringVar()

detail\_entry = Entry(search\_window, textvariable=detail\_var)

detail\_entry.grid(row=7, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Ответственный:").grid(row=8, column=0, padx=5, pady=5, sticky=W)

responsible\_var = StringVar()

responsible\_entry = Entry(search\_window, textvariable=responsible\_var)

responsible\_entry.grid(row=8, column=1, padx=5, pady=5, sticky=EW)

def perform\_search():

try:

conditions = []

params = []

if id\_var.get():

conditions.append("invoice\_id = %s")

params.append(int(id\_var.get()))

if counteragent\_var.get():

conditions.append("counteragent\_name ILIKE %s")

params.append(f"%{counteragent\_var.get()}%")

if date\_from\_var.get():

conditions.append("date\_time >= %s")

params.append(date\_from\_var.get())

if type\_var.get():

conditions.append("type\_invoice\_text = %s")

params.append(type\_var.get())

if status\_var.get():

conditions.append("status\_text = %s")

params.append(status\_var.get())

if detail\_var.get():

conditions.append("type\_detail ILIKE %s")

params.append(f"%{detail\_var.get()}%")

if responsible\_var.get():

conditions.append("""

(responsible\_last\_name ILIKE %s OR

responsible\_first\_name ILIKE %s OR

COALESCE(responsible\_patronymic, '') ILIKE %s)

""")

params.extend([

f"%{responsible\_var.get()}%",

f"%{responsible\_var.get()}%",

f"%{responsible\_var.get()}%"

])

self.current\_search\_conditions['invoice'] = {

'conditions': conditions,

'params': params

}

self.load\_invoices()

search\_window.destroy()

except ValueError as ve:

messagebox.showerror("Ошибка", f"Некорректные данные: {str(ve)}")

except Exception as e:

self.status\_bar.config(text=f"Ошибка поиска: {str(e)}")

self.conn.rollback()

def reset\_search():

self.current\_search\_conditions['invoice'] = None

self.load\_invoices()

search\_window.destroy()

Button(search\_window, text="Найти", command=perform\_search).grid(

row=9, column=0, padx=5, pady=10, sticky=EW)

Button(search\_window, text="Сбросить", command=reset\_search).grid(

row=9, column=1, padx=5, pady=10, sticky=EW)

def load\_warehouse(self):

"""Загрузка данных о складе"""

try:

self.warehouse\_tree.delete(\*self.warehouse\_tree.get\_children())

if self.current\_search\_conditions['warehouse']:

conditions = self.current\_search\_conditions['warehouse']['conditions']

params = self.current\_search\_conditions['warehouse']['params']

query = """

SELECT

detail\_id,

warehouse\_number,

room\_number,

rack\_number,

shelf\_number,

type\_detail,

weight

FROM warehouse\_details\_view

WHERE """ + " AND ".join(conditions) + """

ORDER BY detail\_id, warehouse\_number, room\_number, rack\_number, shelf\_number

"""

self.cursor.execute(query, params)

else:

self.cursor.execute("""

SELECT

detail\_id,

warehouse\_number,

room\_number,

rack\_number,

shelf\_number,

type\_detail,

weight

FROM warehouse\_details\_view

ORDER BY detail\_id, warehouse\_number, room\_number, rack\_number, shelf\_number

""")

for row in self.cursor.fetchall():

self.warehouse\_tree.insert("", END, values=row)

count = len(self.warehouse\_tree.get\_children())

if self.current\_search\_conditions['warehouse']:

self.status\_bar.config(text=f"Найдено деталей: {count}")

else:

self.status\_bar.config(text=f"Данные склада загружены. Всего: {count}")

except Exception as e:

print(f"[LOAD ERROR] Failed to load warehouse data: {e}")

self.status\_bar.config(text=f"Ошибка загрузки данных склада: {str(e)}")

self.conn.rollback()

def search\_warehouse\_item(self):

"""Поиск деталей на складе по различным критериям"""

search\_window = Toplevel(self.root)

search\_window.title("Поиск деталей на складе")

Label(search\_window, text="Критерии поиска:").grid(row=0, column=0, columnspan=2, pady=5)

Label(search\_window, text="Тип детали:").grid(row=1, column=0, padx=5, pady=5, sticky=W)

type\_var = StringVar()

type\_entry = Entry(search\_window, textvariable=type\_var)

type\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Номер склада:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

warehouse\_var = StringVar()

warehouse\_entry = Entry(search\_window, textvariable=warehouse\_var)

warehouse\_entry.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Номер комнаты:").grid(row=3, column=0, padx=5, pady=5, sticky=W)

room\_var = StringVar()

room\_entry = Entry(search\_window, textvariable=room\_var)

room\_entry.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Номер стеллажа:").grid(row=4, column=0, padx=5, pady=5, sticky=W)

rack\_var = StringVar()

rack\_entry = Entry(search\_window, textvariable=rack\_var)

rack\_entry.grid(row=4, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Номер полки:").grid(row=5, column=0, padx=5, pady=5, sticky=W)

shelf\_var = StringVar()

shelf\_entry = Entry(search\_window, textvariable=shelf\_var)

shelf\_entry.grid(row=5, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Вес от:").grid(row=6, column=0, padx=5, pady=5, sticky=W)

weight\_from\_var = StringVar()

weight\_from\_entry = Entry(search\_window, textvariable=weight\_from\_var)

weight\_from\_entry.grid(row=6, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Вес до:").grid(row=7, column=0, padx=5, pady=5, sticky=W)

weight\_to\_var = StringVar()

weight\_to\_entry = Entry(search\_window, textvariable=weight\_to\_var)

weight\_to\_entry.grid(row=7, column=1, padx=5, pady=5, sticky=EW)

def perform\_search():

try:

conditions = []

params = []

if type\_var.get():

conditions.append("type\_detail ILIKE %s")

params.append(f"%{type\_var.get()}%")

if warehouse\_var.get():

conditions.append("warehouse\_number = %s")

params.append(warehouse\_var.get())

if room\_var.get():

conditions.append("room\_number = %s")

params.append(room\_var.get())

if rack\_var.get():

conditions.append("rack\_number = %s")

params.append(rack\_var.get())

if shelf\_var.get():

conditions.append("shelf\_number = %s")

params.append(shelf\_var.get())

if weight\_from\_var.get():

try:

weight\_from = float(weight\_from\_var.get())

conditions.append("weight >= %s")

params.append(weight\_from)

except ValueError:

messagebox.showwarning("Предупреждение", "Некорректное значение веса 'от'")

if weight\_to\_var.get():

try:

weight\_to = float(weight\_to\_var.get())

conditions.append("weight <= %s")

params.append(weight\_to)

except ValueError:

messagebox.showwarning("Предупреждение", "Некорректное значение веса 'до'")

self.current\_search\_conditions['warehouse'] = {

'conditions': conditions,

'params': params

}

self.load\_warehouse()

search\_window.destroy()

except Exception as e:

self.status\_bar.config(text=f"Ошибка поиска: {str(e)}")

self.conn.rollback()

def reset\_search():

self.current\_search\_conditions['warehouse'] = None

self.load\_warehouse()

search\_window.destroy()

Button(search\_window, text="Найти", command=perform\_search).grid(

row=8, column=0, padx=5, pady=10, sticky=EW)

Button(search\_window, text="Сбросить", command=reset\_search).grid(

row=8, column=1, padx=5, pady=10, sticky=EW)

def load\_counteragents(self):

"""Загрузка данных о контрагентах"""

try:

self.counteragent\_tree.delete(\*self.counteragent\_tree.get\_children())

if self.current\_search\_conditions['counteragent']:

conditions = self.current\_search\_conditions['counteragent']['conditions']

params = self.current\_search\_conditions['counteragent']['params']

query = "SELECT \* FROM counteragent WHERE " + " AND ".join(conditions) + " ORDER BY counteragent\_id"

self.cursor.execute(query, params)

else:

self.cursor.execute("SELECT \* FROM counteragent ORDER BY counteragent\_id")

for row in self.cursor.fetchall():

self.counteragent\_tree.insert("", END, values=row)

count = len(self.counteragent\_tree.get\_children())

if self.current\_search\_conditions['counteragent']:

self.status\_bar.config(text=f"Найдено контрагентов: {count}")

else:

self.status\_bar.config(text=f"Контрагенты загружены. Всего: {count}")

except Exception as e:

self.status\_bar.config(text=f"Ошибка: {str(e)}")

def search\_counteragent(self):

"""Поиск контрагентов по различным критериям"""

search\_window = Toplevel(self.root)

search\_window.title("Поиск контрагентов")

Label(search\_window, text="Критерии поиска:").grid(row=0, column=0, columnspan=2, pady=5)

Label(search\_window, text="ID контрагента:").grid(row=1, column=0, padx=5, pady=5, sticky=W)

id\_var = StringVar()

id\_entry = Entry(search\_window, textvariable=id\_var)

id\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Название:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

name\_var = StringVar()

name\_entry = Entry(search\_window, textvariable=name\_var)

name\_entry.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Контактное лицо:").grid(row=3, column=0, padx=5, pady=5, sticky=W)

contact\_var = StringVar()

contact\_entry = Entry(search\_window, textvariable=contact\_var)

contact\_entry.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Телефон:").grid(row=4, column=0, padx=5, pady=5, sticky=W)

phone\_var = StringVar()

phone\_entry = Entry(search\_window, textvariable=phone\_var)

phone\_entry.grid(row=4, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Адрес:").grid(row=5, column=0, padx=5, pady=5, sticky=W)

address\_var = StringVar()

address\_entry = Entry(search\_window, textvariable=address\_var)

address\_entry.grid(row=5, column=1, padx=5, pady=5, sticky=EW)

def perform\_search():

try:

conditions = []

params = []

if id\_var.get():

conditions.append("counteragent\_id = %s")

params.append(int(id\_var.get()))

if name\_var.get():

conditions.append("counteragent\_name ILIKE %s")

params.append(f"%{name\_var.get()}%")

if contact\_var.get():

conditions.append("contact\_person ILIKE %s")

params.append(f"%{contact\_var.get()}%")

if phone\_var.get():

conditions.append("phone\_number::text LIKE %s")

params.append(f"%{phone\_var.get()}%")

if address\_var.get():

conditions.append("address ILIKE %s")

params.append(f"%{address\_var.get()}%")

self.current\_search\_conditions['counteragent'] = {

'conditions': conditions,

'params': params

}

self.load\_counteragents()

search\_window.destroy()

except ValueError as ve:

messagebox.showerror("Ошибка", f"Некорректные данные: {str(ve)}")

except Exception as e:

self.status\_bar.config(text=f"Ошибка поиска: {str(e)}")

self.conn.rollback()

def reset\_search():

self.current\_search\_conditions['counteragent'] = None

self.load\_counteragents()

search\_window.destroy()

Button(search\_window, text="Найти", command=perform\_search).grid(

row=6, column=0, padx=5, pady=10, sticky=EW)

Button(search\_window, text="Сбросить", command=reset\_search).grid(

row=6, column=1, padx=5, pady=10, sticky=EW)

def load\_employees(self):

"""Загрузка данных о сотрудниках"""

try:

self.employee\_tree.delete(\*self.employee\_tree.get\_children())

if self.current\_search\_conditions['employee']:

conditions = self.current\_search\_conditions['employee']['conditions']

params = self.current\_search\_conditions['employee']['params']

query = "SELECT \* FROM employee WHERE " + " AND ".join(conditions) + " ORDER BY employee\_id"

self.cursor.execute(query, params)

else:

self.cursor.execute("SELECT \* FROM employee ORDER BY employee\_id")

for row in self.cursor.fetchall():

self.employee\_tree.insert("", END, values=row)

count = len(self.employee\_tree.get\_children())

if self.current\_search\_conditions['employee']:

self.status\_bar.config(text=f"Найдено сотрудников: {count}")

else:

self.status\_bar.config(text=f"Сотрудники загружены. Всего: {count}")

except Exception as e:

self.status\_bar.config(text=f"Ошибка: {str(e)}")

def search\_employee(self):

"""Поиск сотрудников по различным критериям"""

search\_window = Toplevel(self.root)

search\_window.title("Поиск сотрудников")

Label(search\_window, text="Критерии поиска:").grid(row=0, column=0, columnspan=2, pady=5)

Label(search\_window, text="ID сотрудника:").grid(row=1, column=0, padx=5, pady=5, sticky=W)

id\_var = StringVar()

id\_entry = Entry(search\_window, textvariable=id\_var)

id\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Роль:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

role\_var = StringVar()

role\_combobox = ttk.Combobox(search\_window, textvariable=role\_var,

values=["", "Кладовщик", "Менеджер склада", "Владелец"])

role\_combobox.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Фамилия:").grid(row=3, column=0, padx=5, pady=5, sticky=W)

last\_name\_var = StringVar()

last\_name\_entry = Entry(search\_window, textvariable=last\_name\_var)

last\_name\_entry.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Имя:").grid(row=4, column=0, padx=5, pady=5, sticky=W)

first\_name\_var = StringVar()

first\_name\_entry = Entry(search\_window, textvariable=first\_name\_var)

first\_name\_entry.grid(row=4, column=1, padx=5, pady=5, sticky=EW)

Label(search\_window, text="Отчество:").grid(row=5, column=0, padx=5, pady=5, sticky=W)

patronymic\_var = StringVar()

patronymic\_entry = Entry(search\_window, textvariable=patronymic\_var)

patronymic\_entry.grid(row=5, column=1, padx=5, pady=5, sticky=EW)

def perform\_search():

try:

conditions = []

params = []

if id\_var.get():

conditions.append("employee\_id = %s")

params.append(int(id\_var.get()))

if role\_var.get():

conditions.append("employee\_role = %s")

params.append(role\_var.get())

if last\_name\_var.get():

conditions.append("last\_name ILIKE %s")

params.append(f"%{last\_name\_var.get()}%")

if first\_name\_var.get():

conditions.append("first\_name ILIKE %s")

params.append(f"%{first\_name\_var.get()}%")

if patronymic\_var.get():

conditions.append("patronymic ILIKE %s")

params.append(f"%{patronymic\_var.get()}%")

self.current\_search\_conditions['employee'] = {

'conditions': conditions,

'params': params

}

self.load\_employees()

search\_window.destroy()

except ValueError as ve:

messagebox.showerror("Ошибка", f"Некорректные данные: {str(ve)}")

except Exception as e:

self.status\_bar.config(text=f"Ошибка поиска: {str(e)}")

self.conn.rollback()

def reset\_search():

self.current\_search\_conditions['employee'] = None

self.load\_employees()

search\_window.destroy()

Button(search\_window, text="Найти", command=perform\_search).grid(

row=6, column=0, padx=5, pady=10, sticky=EW)

Button(search\_window, text="Сбросить", command=reset\_search).grid(

row=6, column=1, padx=5, pady=10, sticky=EW)

def add\_invoice(self):

"""Добавление новой накладной с проверкой прав"""

if not self.can\_edit\_invoices:

messagebox.showerror("Ошибка", "У вас нет прав на добавление накладных")

return

try:

add\_window = Toplevel(self.root)

add\_window.title("Добавить накладную")

if self.can\_access\_counteragents:

self.cursor.execute("SELECT counteragent\_id, counteragent\_name FROM counteragent")

counteragents = self.cursor.fetchall()

counteragent\_names = [name for id, name in counteragents]

counteragent\_ids = {name: id for id, name in counteragents}

else:

counteragents = []

counteragent\_names = []

counteragent\_ids = {}

messagebox.showwarning("Предупреждение", "Нет доступа к списку контрагентов")

self.cursor.execute("""

SELECT DISTINCT

responsible\_id,

responsible\_last\_name || ' ' ||

responsible\_first\_name || ' ' ||

COALESCE(responsible\_patronymic, '') as responsible\_name

FROM invoice\_details\_view

ORDER BY responsible\_name

""")

employees = self.cursor.fetchall()

employee\_names = [name for id, name in employees]

employee\_ids = {name: id for id, name in employees}

Label(add\_window, text="Контрагент:").grid(row=0, column=0, padx=5, pady=5, sticky=W)

counteragent\_var = StringVar()

counteragent\_combobox = ttk.Combobox(add\_window, textvariable=counteragent\_var, values=counteragent\_names)

counteragent\_combobox.grid(row=0, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Дата и время (ГГГГ-ММ-ДД ЧЧ:ММ):").grid(row=1, column=0, padx=5, pady=5, sticky=W)

date\_entry = Entry(add\_window)

current\_datetime = datetime.now().strftime("%Y-%m-%d %H:%M")

date\_entry.insert(0, current\_datetime)

date\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Тип накладной:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

type\_var = StringVar()

type\_combobox = ttk.Combobox(add\_window, textvariable=type\_var, values=["Отгрузка", "Выгрузка"])

type\_combobox.current(0)

type\_combobox.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Статус:").grid(row=3, column=0, padx=5, pady=5, sticky=W)

status\_var = StringVar()

status\_combobox = ttk.Combobox(add\_window, textvariable=status\_var, values=["В процессе", "Завершено"])

status\_combobox.current(0)

status\_combobox.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

detail\_frame = Frame(add\_window)

detail\_frame.grid(row=4, column=0, columnspan=2, padx=5, pady=5, sticky=EW)

Label(detail\_frame, text="Деталь:").pack(side=LEFT, padx=(0, 5))

detail\_var = StringVar()

detail\_entry = Entry(detail\_frame, textvariable=detail\_var)

detail\_entry.pack(side=LEFT, expand=True, fill=X)

search\_btn = Button(detail\_frame, text="🔍", command=lambda: self.search\_detail(detail\_var))

search\_btn.pack(side=LEFT, padx=(5, 0))

Label(add\_window, text="Количество:").grid(row=5, column=0, padx=5, pady=5, sticky=W)

quantity\_entry = Entry(add\_window)

quantity\_entry.grid(row=5, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Ответственный:").grid(row=6, column=0, padx=5, pady=5, sticky=W)

employee\_var = StringVar()

employee\_combobox = ttk.Combobox(add\_window, textvariable=employee\_var, values=employee\_names)

employee\_combobox.grid(row=6, column=1, padx=5, pady=5, sticky=EW)

def save\_invoice():

try:

input\_datetime\_str = date\_entry.get()

try:

input\_datetime = datetime.strptime(input\_datetime\_str, "%Y-%m-%d %H:%M")

except ValueError:

raise ValueError("Некорректный формат даты. Используйте ГГГГ-ММ-ДД ЧЧ:ММ")

current\_datetime = datetime.now()

if input\_datetime > current\_datetime:

raise ValueError("Дата накладной не может быть в будущем. Укажите текущую или прошедшую дату.")

counteragent\_id = counteragent\_ids.get(counteragent\_var.get())

employee\_id = employee\_ids.get(employee\_var.get())

if employee\_id:

self.cursor.execute("""

SELECT COUNT(\*) FROM invoice\_employee

WHERE responsible = %s AND invoiceid IN (

SELECT invoice\_id FROM invoice WHERE status = FALSE

)

""", (employee\_id,))

active\_invoices\_count = self.cursor.fetchone()[0]

if active\_invoices\_count >= 5:

raise ValueError(f"У сотрудника уже {active\_invoices\_count} активных накладных. Максимум - 5.")

if None in (counteragent\_id, employee\_id):

raise ValueError("Не все обязательные поля заполнены")

detail\_name = detail\_var.get().strip()

if not detail\_name:

raise ValueError("Название детали не может быть пустым")

self.cursor.execute("""

SELECT detail\_id FROM details

WHERE type\_detail = %s

LIMIT 1

""", (detail\_name,))

detail\_data = self.cursor.fetchone()

if not detail\_data:

raise ValueError(f"Деталь '{detail\_name}' не найдена на складе")

detail\_id = detail\_data[0]

type\_invoice = type\_var.get() == "Выгрузка"

status = status\_var.get() == "Завершено"

try:

quantity = int(quantity\_entry.get())

if quantity <= 0:

raise ValueError("Количество должно быть положительным числом")

except ValueError:

raise ValueError("Количество должно быть целым числом")

if not type\_invoice:

self.cursor.execute("""

SELECT COUNT(\*)

FROM details

WHERE type\_detail = %s

""", (detail\_name,))

total\_available = self.cursor.fetchone()[0]

self.cursor.execute("""

SELECT COALESCE(SUM(id.quantity), 0)

FROM invoice\_detail id

JOIN invoice i ON id.invoiceid = i.invoice\_id

JOIN details d ON id.detailid = d.detail\_id

WHERE d.type\_detail = %s AND i.type\_invoice = FALSE AND i.status = FALSE

""", (detail\_name,))

reserved = self.cursor.fetchone()[0]

if total\_available < (reserved + quantity):

raise ValueError(

f"Недостаточно деталей на складе для отгрузки. Доступно: {total\_available}, "

f"уже зарезервировано для отгрузки: {reserved}, требуется: {quantity}"

)

self.cursor.execute("""

INSERT INTO invoice (counteragentid, date\_time, type\_invoice, status)

VALUES (%s, %s, %s, %s)

RETURNING invoice\_id

""", (counteragent\_id, input\_datetime\_str, type\_invoice, status))

invoice\_id = self.cursor.fetchone()[0]

self.cursor.execute("""

INSERT INTO invoice\_detail (invoiceid, detailid, quantity)

VALUES (%s, %s, %s)

""", (invoice\_id, detail\_id, quantity))

self.cursor.execute("""

INSERT INTO invoice\_employee (invoiceid, responsible, granted\_access, when\_granted)

VALUES (%s, %s, %s, NOW())

""", (invoice\_id, employee\_id, employee\_id))

self.conn.commit()

self.load\_invoices()

add\_window.destroy()

messagebox.showinfo("Успех", "Накладная успешно добавлена")

except ValueError as ve:

messagebox.showerror("Ошибка", f"Неверные данные: {str(ve)}")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось добавить накладную: {str(e)}")

Button(add\_window, text="Сохранить", command=save\_invoice).grid(row=7, column=0, columnspan=2, pady=10)

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось открыть форму: {str(e)}")

def edit\_invoice(self):

"""Редактирование накладной с проверкой прав"""

if not self.can\_edit\_invoices:

messagebox.showerror("Ошибка", "У вас нет прав на редактирование накладных")

return

selected = self.invoice\_tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите накладную для редактирования")

return

item = self.invoice\_tree.item(selected[0])

invoice\_id = item['values'][0]

try:

self.cursor.execute("""

SELECT

invoice\_id,

counteragent\_name,

date\_time,

type\_invoice\_text,

status\_text,

type\_detail,

quantity,

responsible\_last\_name || ' ' ||

responsible\_first\_name || ' ' ||

COALESCE(responsible\_patronymic, '') as responsible

FROM invoice\_details\_view

WHERE invoice\_id = %s

""", (invoice\_id,))

invoice\_data = self.cursor.fetchone()

if not invoice\_data:

messagebox.showerror("Ошибка", "Накладная не найдена")

return

edit\_window = Toplevel(self.root)

edit\_window.title("Редактировать накладную")

self.cursor.execute("SELECT counteragent\_id, counteragent\_name FROM counteragent")

counteragents = self.cursor.fetchall()

counteragent\_names = [name for id, name in counteragents]

counteragent\_ids = {name: id for id, name in counteragents}

self.cursor.execute("""

SELECT DISTINCT

responsible\_id,

responsible\_last\_name || ' ' ||

responsible\_first\_name || ' ' ||

COALESCE(responsible\_patronymic, '') as responsible\_name

FROM invoice\_details\_view

ORDER BY responsible\_name

""")

employees = self.cursor.fetchall()

employee\_names = [name for id, name in employees]

employee\_ids = {name: id for id, name in employees}

Label(edit\_window, text="Контрагент:").grid(row=0, column=0, padx=5, pady=5, sticky=W)

counteragent\_var = StringVar()

counteragent\_combobox = ttk.Combobox(edit\_window, textvariable=counteragent\_var,

values=counteragent\_names)

counteragent\_combobox.grid(row=0, column=1, padx=5, pady=5, sticky=EW)

for id, name in counteragents:

if name == invoice\_data[1]:

counteragent\_var.set(invoice\_data[1])

break

Label(edit\_window, text="Дата и время:").grid(row=1, column=0, padx=5, pady=5, sticky=W)

date\_entry = Entry(edit\_window)

date\_entry.insert(0, invoice\_data[2].strftime("%Y-%m-%d %H:%M"))

date\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(edit\_window, text="Тип накладной:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

type\_var = StringVar()

type\_combobox = ttk.Combobox(edit\_window, textvariable=type\_var,

values=["Отгрузка", "Выгрузка"])

type\_combobox.current(1 if invoice\_data[3] == "Выгрузка" else 0)

type\_combobox.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(edit\_window, text="Статус:").grid(row=3, column=0, padx=5, pady=5, sticky=W)

status\_var = StringVar()

status\_combobox = ttk.Combobox(edit\_window, textvariable=status\_var,

values=["В процессе", "Завершено"])

status\_combobox.current(1 if invoice\_data[4] == "Завершено" else 0)

status\_combobox.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

detail\_frame = Frame(edit\_window)

detail\_frame.grid(row=4, column=0, columnspan=2, padx=5, pady=5, sticky=EW)

Label(detail\_frame, text="Деталь:").pack(side=LEFT, padx=(0, 5))

detail\_var = StringVar()

detail\_entry = Entry(detail\_frame, textvariable=detail\_var)

detail\_entry.pack(side=LEFT, expand=True, fill=X)

search\_btn = Button(detail\_frame, text="🔍", command=lambda: self.search\_detail(detail\_var))

search\_btn.pack(side=LEFT, padx=(5, 0))

Label(edit\_window, text="Количество:").grid(row=5, column=0, padx=5, pady=5, sticky=W)

quantity\_entry = Entry(edit\_window)

quantity\_entry.insert(0, str(invoice\_data[6]))

quantity\_entry.grid(row=5, column=1, padx=5, pady=5, sticky=EW)

Label(edit\_window, text="Ответственный:").grid(row=6, column=0, padx=5, pady=5, sticky=W)

employee\_var = StringVar()

employee\_combobox = ttk.Combobox(edit\_window, textvariable=employee\_var,

values=employee\_names)

employee\_combobox.grid(row=6, column=1, padx=5, pady=5, sticky=EW)

for id, name in employees:

if name == invoice\_data[7]:

employee\_var.set(name)

break

def save\_changes():

try:

detail\_name = detail\_var.get().strip()

if not detail\_name:

raise ValueError("Тип детали не может быть пустым")

self.cursor.execute("""

SELECT detail\_id FROM details

WHERE type\_detail = %s

LIMIT 1

""", (detail\_name,))

detail\_data = self.cursor.fetchone()

if not detail\_data:

raise ValueError(f"Деталь '{detail\_name}' не найдена на складе")

detail\_id = detail\_data[0]

try:

quantity = int(quantity\_entry.get())

if quantity <= 0:

raise ValueError("Количество должно быть положительным числом!")

except ValueError:

raise ValueError("Количество должно быть целым числом!")

counteragent\_name = counteragent\_var.get()

counteragent\_id = counteragent\_ids.get(counteragent\_name)

if counteragent\_id is None:

raise ValueError("Выберите корректного контрагента")

employee\_name = employee\_var.get()

employee\_id = employee\_ids.get(employee\_name)

if employee\_id is None:

raise ValueError("Выберите корректного сотрудника")

if employee\_id:

self.cursor.execute("""

SELECT COUNT(\*) FROM invoice\_employee

WHERE responsible = %s

AND invoiceid IN (SELECT invoice\_id FROM invoice WHERE status = FALSE)

AND invoiceid != %s

""", (employee\_id, invoice\_id))

active\_invoices\_count = self.cursor.fetchone()[0]

if active\_invoices\_count >= 5:

raise ValueError(f"У сотрудника уже {active\_invoices\_count} активных накладных. Максимум - 5.")

type\_invoice = type\_var.get() == "Выгрузка"

status = status\_var.get() == "Завершено"

if not type\_invoice:

self.cursor.execute("""

SELECT quantity FROM invoice\_detail

WHERE invoiceid = %s

""", (invoice\_id,))

old\_quantity\_row = self.cursor.fetchone()

old\_quantity = old\_quantity\_row[0] if old\_quantity\_row else 0

self.cursor.execute("""

SELECT COUNT(\*)

FROM details d

WHERE d.type\_detail = %s

AND NOT EXISTS (

SELECT 1 FROM invoice\_detail id

JOIN invoice i ON id.invoiceid = i.invoice\_id

WHERE id.detailid = d.detail\_id

AND i.type\_invoice = FALSE --- отгрузка

AND i.status = TRUE --- завершённые

)

""", (detail\_name,))

total\_available = self.cursor.fetchone()[0]

self.cursor.execute("""

SELECT COALESCE(SUM(id.quantity), 0)

FROM invoice\_detail id

JOIN invoice i ON id.invoiceid = i.invoice\_id

JOIN details d ON id.detailid = d.detail\_id

WHERE d.type\_detail = %s

AND i.type\_invoice = FALSE --- отгрузка

AND i.status = FALSE --- в процессе

AND i.invoice\_id != %s

""", (detail\_name, invoice\_id))

reserved = self.cursor.fetchone()[0]

if total\_available < (reserved + quantity - old\_quantity):

raise ValueError(

f"Недостаточно деталей на складе для отгрузки. "

f"Доступно: {total\_available}, "

f"уже зарезервировано в других накладных: {reserved}, "

f"новое количество: {quantity} (было: {old\_quantity})"

)

self.cursor.execute("""

UPDATE invoice

SET counteragentid = %s, date\_time = %s, type\_invoice = %s, status = %s

WHERE invoice\_id = %s

""", (counteragent\_id, date\_entry.get(), type\_invoice, status, invoice\_id))

self.cursor.execute("""

UPDATE invoice\_detail

SET detailid = %s, quantity = %s

WHERE invoiceid = %s

""", (detail\_id, quantity, invoice\_id))

self.cursor.execute("""

UPDATE invoice\_employee

SET responsible = %s

WHERE invoiceid = %s

""", (employee\_id, invoice\_id))

self.conn.commit()

self.load\_invoices()

edit\_window.destroy()

messagebox.showinfo("Успех", "Накладная успешно обновлена")

except ValueError as ve:

messagebox.showerror("Ошибка", f"Неверные данные: {str(ve)}")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось обновить накладную: {str(e)}")

Button(edit\_window, text="Сохранить", command=save\_changes).grid(row=7, column=0, columnspan=2, pady=10)

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось открыть форму: {str(e)}")

def delete\_invoice(self):

"""Удаление накладной с предварительным удалением связанных записей"""

selected = self.invoice\_tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите накладную для удаления")

return

item = self.invoice\_tree.item(selected[0])

invoice\_id = item['values'][0]

if messagebox.askyesno("Подтверждение", f"Вы уверены, что хотите удалить накладную №{invoice\_id}?"):

try:

self.cursor.execute("DELETE FROM invoice\_detail WHERE invoiceid = %s", (invoice\_id,))

self.cursor.execute("DELETE FROM invoice\_employee WHERE invoiceid = %s", (invoice\_id,))

self.cursor.execute("DELETE FROM invoice WHERE invoice\_id = %s", (invoice\_id,))

self.conn.commit()

self.load\_invoices()

if len(self.invoice\_tree.get\_children()) == 0 and self.current\_search\_conditions['invoice']:

messagebox.showinfo("Информация", "Таблица пуста. Сброс условий поиска.")

self.current\_search\_conditions['invoice'] = None

self.load\_invoices()

messagebox.showinfo("Успех", "Накладная успешно удалена")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось удалить накладную: {str(e)}")

def add\_warehouse\_item(self):

"""Добавление детали на склад с динамическими выпадающими списками"""

if not self.can\_edit\_warehouse:

messagebox.showerror("Ошибка", "У вас нет прав на добавление деталей")

return

try:

add\_window = Toplevel(self.root)

add\_window.title("Добавить деталь на склад")

self.cursor.execute("SELECT warehouse\_id, warehouse\_number FROM warehouse ORDER BY warehouse\_number")

warehouses = self.cursor.fetchall()

warehouse\_options = [str(number) for id, number in warehouses]

warehouse\_ids = {str(number): id for id, number in warehouses}

row = 0

Label(add\_window, text="Номер склада:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

warehouse\_var = StringVar()

warehouse\_combobox = ttk.Combobox(add\_window, textvariable=warehouse\_var,

values=warehouse\_options, state="readonly")

warehouse\_combobox.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(add\_window, text="Номер комнаты:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

room\_var = StringVar()

room\_combobox = ttk.Combobox(add\_window, textvariable=room\_var,

values=[], state="readonly")

room\_combobox.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(add\_window, text="Номер стеллажа:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

rack\_var = StringVar()

rack\_combobox = ttk.Combobox(add\_window, textvariable=rack\_var,

values=[], state="readonly")

rack\_combobox.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(add\_window, text="Номер полки:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

shelf\_var = StringVar()

shelf\_combobox = ttk.Combobox(add\_window, textvariable=shelf\_var,

values=[], state="readonly")

shelf\_combobox.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(add\_window, text="Тип детали:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

type\_entry = Entry(add\_window)

type\_entry.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(add\_window, text="Вес (кг):").grid(row=row, column=0, padx=5, pady=5, sticky=W)

weight\_entry = Entry(add\_window)

weight\_entry.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

weight\_entry.insert(0, "0.0")

row += 1

def update\_rooms(event=None):

selected\_warehouse = warehouse\_var.get()

if selected\_warehouse in warehouse\_ids:

warehouse\_id = warehouse\_ids[selected\_warehouse]

try:

self.cursor.execute("""

SELECT room\_id, room\_number

FROM room

WHERE warehouseID = %s

ORDER BY room\_number

""", (warehouse\_id,))

rooms = self.cursor.fetchall()

room\_combobox['values'] = [str(number) for id, number in rooms]

if rooms:

room\_combobox.current(0)

room\_var.set(str(rooms[0][1]))

update\_racks()

else:

room\_var.set('')

rack\_var.set('')

shelf\_var.set('')

rack\_combobox['values'] = []

shelf\_combobox['values'] = []

except Exception as e:

print(f"Error updating rooms: {e}")

room\_combobox['values'] = []

else:

room\_combobox['values'] = []

rack\_combobox['values'] = []

shelf\_combobox['values'] = []

def update\_racks(event=None):

selected\_warehouse = warehouse\_var.get()

selected\_room = room\_var.get()

if selected\_warehouse in warehouse\_ids and selected\_room:

warehouse\_id = warehouse\_ids[selected\_warehouse]

try:

self.cursor.execute("""

SELECT rack\_id, rack\_number

FROM rack

WHERE roomID = (

SELECT room\_id FROM room

WHERE room\_number = %s AND warehouseID = %s

)

ORDER BY rack\_number

""", (int(selected\_room), warehouse\_id))

racks = self.cursor.fetchall()

rack\_combobox['values'] = [str(number) for id, number in racks]

if racks:

rack\_combobox.current(0)

rack\_var.set(str(racks[0][1]))

update\_shelves()

else:

rack\_var.set('')

shelf\_var.set('')

shelf\_combobox['values'] = []

except Exception as e:

print(f"Error updating racks: {e}")

rack\_combobox['values'] = []

else:

rack\_combobox['values'] = []

shelf\_combobox['values'] = []

def update\_shelves(event=None):

selected\_warehouse = warehouse\_var.get()

selected\_room = room\_var.get()

selected\_rack = rack\_var.get()

if (selected\_warehouse in warehouse\_ids and selected\_room and selected\_rack):

warehouse\_id = warehouse\_ids[selected\_warehouse]

try:

self.cursor.execute("""

SELECT shelf\_id, shelf\_number

FROM shelf

WHERE rackID = (

SELECT rack\_id FROM rack

WHERE rack\_number = %s AND roomID = (

SELECT room\_id FROM room

WHERE room\_number = %s AND warehouseID = %s

)

)

ORDER BY shelf\_number

""", (int(selected\_rack), int(selected\_room), warehouse\_id))

shelves = self.cursor.fetchall()

shelf\_combobox['values'] = [str(number) for id, number in shelves]

if shelves:

shelf\_combobox.current(0)

shelf\_var.set(str(shelves[0][1]))

else:

shelf\_var.set('')

except Exception as e:

print(f"Error updating shelves: {e}")

shelf\_combobox['values'] = []

else:

shelf\_combobox['values'] = []

warehouse\_combobox.bind("<<ComboboxSelected>>", update\_rooms)

room\_combobox.bind("<<ComboboxSelected>>", update\_racks)

rack\_combobox.bind("<<ComboboxSelected>>", update\_shelves)

if warehouse\_options:

warehouse\_combobox.current(0)

warehouse\_var.set(warehouse\_options[0])

update\_rooms()

def save\_item():

try:

if not all([warehouse\_var.get(), room\_var.get(),

rack\_var.get(), shelf\_var.get(),

type\_entry.get(), weight\_entry.get()]):

raise ValueError("Все поля должны быть заполнены")

self.cursor.execute("""

SELECT shelf\_id FROM shelf

WHERE shelf\_number = %s AND rackID = (

SELECT rack\_id FROM rack

WHERE rack\_number = %s AND roomID = (

SELECT room\_id FROM room

WHERE room\_number = %s AND warehouseID = (

SELECT warehouse\_id FROM warehouse

WHERE warehouse\_number = %s

)

)

)

""", (int(shelf\_var.get()), int(rack\_var.get()), int(room\_var.get()), int(warehouse\_var.get())))

shelf\_data = self.cursor.fetchone()

if not shelf\_data:

raise ValueError("Полка не найдена в базе данных")

shelf\_id = shelf\_data[0]

try:

weight = float(weight\_entry.get())

if weight <= 0.0:

raise ValueError("Вес не может быть отрицательным")

except ValueError as e:

raise ValueError("Вес должен быть числом (например, 2.1)")

self.cursor.execute("""

SELECT COUNT(\*) FROM details WHERE shelfid = %s

""", (shelf\_id,))

details\_count = self.cursor.fetchone()[0]

if details\_count >= 5:

raise ValueError(f"На полке уже {details\_count} деталей. Максимум - 5.")

self.cursor.execute("""

INSERT INTO details (shelfid, type\_detail, weight)

VALUES (%s, %s, %s)

""", (shelf\_id, type\_entry.get(), weight))

self.conn.commit()

self.load\_warehouse()

add\_window.destroy()

messagebox.showinfo("Успех", "Деталь успешно добавлена")

except ValueError as ve:

messagebox.showerror("Ошибка", f"Неверные данные: {str(ve)}")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось добавить деталь: {str(e)}")

Button(add\_window, text="Сохранить", command=save\_item).grid(

row=row, column=0, columnspan=2, pady=10)

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось открыть форму: {str(e)}")

def edit\_warehouse\_item(self):

"""Редактирование детали на складе с динамическими выпадающими списками"""

if not self.can\_edit\_warehouse:

messagebox.showerror("Ошибка", "У вас нет прав на редактирование деталей")

return

selected = self.warehouse\_tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите деталь для редактирования")

return

item = self.warehouse\_tree.item(selected[0])

detail\_id = item['values'][0]

try:

self.cursor.execute("""

SELECT

d.detail\_id, d.type\_detail, d.weight,

s.shelf\_id, s.shelf\_number,

rk.rack\_id, rk.rack\_number,

r.room\_id, r.room\_number,

w.warehouse\_id, w.warehouse\_number

FROM details d

JOIN shelf s ON d.shelfid = s.shelf\_id

JOIN rack rk ON s.rackid = rk.rack\_id

JOIN room r ON rk.roomid = r.room\_id

JOIN warehouse w ON r.warehouseid = w.warehouse\_id

WHERE d.detail\_id = %s

""", (detail\_id,))

detail\_data = self.cursor.fetchone()

if not detail\_data:

messagebox.showerror("Ошибка", "Деталь не найдена")

return

edit\_window = Toplevel(self.root)

edit\_window.title("Редактировать деталь")

self.cursor.execute("SELECT warehouse\_id, warehouse\_number FROM warehouse ORDER BY warehouse\_number")

warehouses = self.cursor.fetchall()

warehouse\_options = [str(number) for id, number in warehouses]

warehouse\_ids = {str(number): id for id, number in warehouses}

row = 0

Label(edit\_window, text="Номер склада:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

warehouse\_var = StringVar(value=str(detail\_data[10]))

warehouse\_combobox = ttk.Combobox(edit\_window, textvariable=warehouse\_var,

values=warehouse\_options, state="readonly")

warehouse\_combobox.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(edit\_window, text="Номер комнаты:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

room\_var = StringVar(value=str(detail\_data[8]))

room\_combobox = ttk.Combobox(edit\_window, textvariable=room\_var,

values=[], state="readonly")

room\_combobox.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(edit\_window, text="Номер стеллажа:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

rack\_var = StringVar(value=str(detail\_data[6]))

rack\_combobox = ttk.Combobox(edit\_window, textvariable=rack\_var,

values=[], state="readonly")

rack\_combobox.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(edit\_window, text="Номер полки:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

shelf\_var = StringVar(value=str(detail\_data[4]))

shelf\_combobox = ttk.Combobox(edit\_window, textvariable=shelf\_var,

values=[], state="readonly")

shelf\_combobox.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(edit\_window, text="Тип детали:").grid(row=row, column=0, padx=5, pady=5, sticky=W)

type\_var = StringVar(value=detail\_data[1])

type\_entry = Entry(edit\_window, textvariable=type\_var)

type\_entry.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

Label(edit\_window, text="Вес (кг):").grid(row=row, column=0, padx=5, pady=5, sticky=W)

weight\_var = StringVar(value=str(detail\_data[2]))

weight\_entry = Entry(edit\_window, textvariable=weight\_var)

weight\_entry.grid(row=row, column=1, padx=5, pady=5, sticky=EW)

row += 1

def update\_rooms(event=None):

selected\_warehouse = warehouse\_var.get()

if selected\_warehouse in warehouse\_ids:

warehouse\_id = warehouse\_ids[selected\_warehouse]

try:

self.cursor.execute("""

SELECT room\_id, room\_number

FROM room

WHERE warehouseID = %s

ORDER BY room\_number

""", (warehouse\_id,))

rooms = self.cursor.fetchall()

room\_combobox['values'] = [str(number) for id, number in rooms]

current\_room = str(detail\_data[8])

if current\_room in [str(number) for id, number in rooms]:

room\_var.set(current\_room)

elif rooms:

room\_combobox.current(0)

room\_var.set(str(rooms[0][1]))

update\_racks()

except Exception as e:

print(f"Error updating rooms: {e}")

room\_combobox['values'] = []

else:

room\_combobox['values'] = []

def update\_racks(event=None):

selected\_warehouse = warehouse\_var.get()

selected\_room = room\_var.get()

if selected\_warehouse in warehouse\_ids and selected\_room:

warehouse\_id = warehouse\_ids[selected\_warehouse]

try:

self.cursor.execute("""

SELECT rack\_id, rack\_number

FROM rack

WHERE roomID = (

SELECT room\_id FROM room

WHERE room\_number = %s AND warehouseID = %s

)

ORDER BY rack\_number

""", (int(selected\_room), warehouse\_id))

racks = self.cursor.fetchall()

rack\_combobox['values'] = [str(number) for id, number in racks]

current\_rack = str(detail\_data[6])

if current\_rack in [str(number) for id, number in racks]:

rack\_var.set(current\_rack)

elif racks:

rack\_combobox.current(0)

rack\_var.set(str(racks[0][1]))

update\_shelves()

except Exception as e:

print(f"Error updating racks: {e}")

rack\_combobox['values'] = []

else:

rack\_combobox['values'] = []

def update\_shelves(event=None):

selected\_warehouse = warehouse\_var.get()

selected\_room = room\_var.get()

selected\_rack = rack\_var.get()

if (selected\_warehouse in warehouse\_ids and selected\_room and selected\_rack):

warehouse\_id = warehouse\_ids[selected\_warehouse]

try:

self.cursor.execute("""

SELECT shelf\_id, shelf\_number

FROM shelf

WHERE rackID = (

SELECT rack\_id FROM rack

WHERE rack\_number = %s AND roomID = (

SELECT room\_id FROM room

WHERE room\_number = %s AND warehouseID = %s

)

)

ORDER BY shelf\_number

""", (int(selected\_rack), int(selected\_room), warehouse\_id))

shelves = self.cursor.fetchall()

shelf\_combobox['values'] = [str(number) for id, number in shelves]

current\_shelf = str(detail\_data[4])

if current\_shelf in [str(number) for id, number in shelves]:

shelf\_var.set(current\_shelf)

elif shelves:

shelf\_combobox.current(0)

shelf\_var.set(str(shelves[0][1]))

except Exception as e:

print(f"Error updating shelves: {e}")

shelf\_combobox['values'] = []

else:

shelf\_combobox['values'] = []

warehouse\_combobox.bind("<<ComboboxSelected>>", update\_rooms)

room\_combobox.bind("<<ComboboxSelected>>", update\_racks)

rack\_combobox.bind("<<ComboboxSelected>>", update\_shelves)

update\_rooms()

def save\_changes():

try:

if not all([warehouse\_var.get(), room\_var.get(),

rack\_var.get(), shelf\_var.get(),

type\_var.get(), weight\_var.get()]):

raise ValueError("Все поля должны быть заполнены")

self.cursor.execute("""

SELECT shelf\_id FROM shelf

WHERE shelf\_number = %s AND rackID = (

SELECT rack\_id FROM rack

WHERE rack\_number = %s AND roomID = (

SELECT room\_id FROM room

WHERE room\_number = %s AND warehouseID = (

SELECT warehouse\_id FROM warehouse

WHERE warehouse\_number = %s

)

)

)

""", (int(shelf\_var.get()), int(rack\_var.get()), int(room\_var.get()), int(warehouse\_var.get())))

shelf\_data = self.cursor.fetchone()

if not shelf\_data:

raise ValueError("Полка не найдена в базе данных")

new\_shelf\_id = shelf\_data[0]

try:

weight = float(weight\_var.get())

if weight <= 0.0:

raise ValueError("Вес не может быть отрицательным")

except ValueError as e:

raise ValueError("Вес должен быть числом (например, 2.1)")

if new\_shelf\_id != detail\_data[3]:

self.cursor.execute("""

SELECT COUNT(\*) FROM details WHERE shelfid = %s

""", (new\_shelf\_id,))

details\_count = self.cursor.fetchone()[0]

if details\_count >= 5:

raise ValueError(f"На новой полке уже {details\_count} деталей. Максимум - 5.")

self.cursor.execute("""

UPDATE details

SET shelfid = %s, type\_detail = %s, weight = %s

WHERE detail\_id = %s

""", (new\_shelf\_id, type\_var.get(), weight, detail\_id))

self.conn.commit()

self.load\_warehouse()

edit\_window.destroy()

messagebox.showinfo("Успех", "Деталь успешно обновлена")

except ValueError as ve:

messagebox.showerror("Ошибка", f"Неверные данные: {str(ve)}")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось обновить деталь: {str(e)}")

Button(edit\_window, text="Сохранить", command=save\_changes).grid(

row=row, column=0, columnspan=2, pady=10)

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось открыть форму редактирования: {str(e)}")

def delete\_warehouse\_item(self):

"""Удаление детали со склада"""

selected = self.warehouse\_tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите деталь для удаления")

return

item = self.warehouse\_tree.item(selected[0])

detail\_id = item['values'][0]

self.cursor.execute("SELECT COUNT(\*) FROM invoice\_detail WHERE detailid = %s", (detail\_id,))

reference\_count = self.cursor.fetchone()[0]

if reference\_count > 0:

messagebox.showerror("Ошибка",

f"Невозможно удалить деталь: она используется в {reference\_count} накладных")

return

if messagebox.askyesno("Подтверждение", f"Вы уверены, что хотите удалить деталь №{detail\_id}?"):

try:

self.cursor.execute("DELETE FROM details WHERE detail\_id = %s", (detail\_id,))

self.conn.commit()

self.load\_warehouse()

if len(self.warehouse\_tree.get\_children()) == 0 and self.current\_search\_conditions['warehouse']:

messagebox.showinfo("Информация", "Таблица пуста. Сброс условий поиска.")

self.current\_search\_conditions['warehouse'] = None

self.load\_warehouse()

messagebox.showinfo("Успех", "Деталь успешно удалена")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось удалить деталь: {str(e)}")

def add\_counteragent(self):

"""Добавление нового контрагента"""

if not self.can\_edit\_counteragents:

messagebox.showerror("Ошибка", "У вас нет прав на добавление накладных")

return

try:

add\_window = Toplevel(self.root)

add\_window.title("Добавить контрагента")

Label(add\_window, text="Название:").grid(row=0, column=0, padx=5, pady=5, sticky=W)

name\_entry = Entry(add\_window)

name\_entry.grid(row=0, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Контактное лицо:").grid(row=1, column=0, padx=5, pady=5, sticky=W)

contact\_entry = Entry(add\_window)

contact\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Телефон:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

phone\_entry = Entry(add\_window)

phone\_entry.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Адрес:").grid(row=3, column=0, padx=5, pady=5, sticky=W)

address\_entry = Entry(add\_window)

address\_entry.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

def save\_counteragent():

try:

self.cursor.execute("""

INSERT INTO counteragent (counteragent\_name, contact\_person, phone\_number, address)

VALUES (%s, %s, %s, %s)

""", (

name\_entry.get(),

contact\_entry.get(),

int(phone\_entry.get()),

address\_entry.get()

))

self.conn.commit()

self.load\_counteragents()

add\_window.destroy()

messagebox.showinfo("Успех", "Контрагент успешно добавлен")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось добавить контрагента: {str(e)}")

Button(add\_window, text="Сохранить", command=save\_counteragent).grid(row=4, column=0, columnspan=2, pady=10)

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось открыть форму: {str(e)}")

def edit\_counteragent(self):

"""Редактирование контрагента"""

if not self.can\_edit\_counteragents:

messagebox.showerror("Ошибка", "У вас нет прав на добавление накладных")

return

selected = self.counteragent\_tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите контрагента для редактирования")

return

item = self.counteragent\_tree.item(selected[0])

counteragent\_id = item['values'][0]

try:

self.cursor.execute("SELECT \* FROM counteragent WHERE counteragent\_id = %s", (counteragent\_id,))

counteragent\_data = self.cursor.fetchone()

if not counteragent\_data:

messagebox.showerror("Ошибка", "Контрагент не найден")

return

edit\_window = Toplevel(self.root)

edit\_window.title("Редактировать контрагента")

Label(edit\_window, text="Название:").grid(row=0, column=0, padx=5, pady=5, sticky=W)

name\_entry = Entry(edit\_window)

name\_entry.insert(0, counteragent\_data[1])

name\_entry.grid(row=0, column=1, padx=5, pady=5, sticky=EW)

Label(edit\_window, text="Контактное лицо:").grid(row=1, column=0, padx=5, pady=5, sticky=W)

contact\_entry = Entry(edit\_window)

contact\_entry.insert(0, counteragent\_data[2])

contact\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(edit\_window, text="Телефон:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

phone\_entry = Entry(edit\_window)

phone\_entry.insert(0, str(counteragent\_data[3]))

phone\_entry.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(edit\_window, text="Адрес:").grid(row=3, column=0, padx=5, pady=5, sticky=W)

address\_entry = Entry(edit\_window)

address\_entry.insert(0, counteragent\_data[4])

address\_entry.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

def save\_changes():

try:

self.cursor.execute("""

UPDATE counteragent

SET

counteragent\_name = %s,

contact\_person = %s,

phone\_number = %s,

address = %s

WHERE counteragent\_id = %s

""", (

name\_entry.get(),

contact\_entry.get(),

int(phone\_entry.get()),

address\_entry.get(),

counteragent\_id

))

self.conn.commit()

self.load\_counteragents()

edit\_window.destroy()

messagebox.showinfo("Успех", "Контрагент успешно обновлен")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось обновить контрагента: {str(e)}")

Button(edit\_window, text="Сохранить", command=save\_changes).grid(row=4, column=0, columnspan=2, pady=10)

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось открыть форму: {str(e)}")

def delete\_counteragent(self):

"""Удаление контрагента"""

selected = self.counteragent\_tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите контрагента для удаления")

return

item = self.counteragent\_tree.item(selected[0])

counteragent\_id = item['values'][0]

counteragent\_name = item['values'][1]

if messagebox.askyesno("Подтверждение", f"Вы уверены, что хотите удалить контрагента '{counteragent\_name}'?"):

try:

self.cursor.execute("DELETE FROM counteragent WHERE counteragent\_id = %s", (counteragent\_id,))

self.conn.commit()

self.load\_counteragents()

if len(self.counteragent\_tree.get\_children()) == 0 and self.current\_search\_conditions['counteragent']:

messagebox.showinfo("Информация", "Таблица пуста. Сброс условий поиска.")

self.current\_search\_conditions['counteragent'] = None

self.load\_counteragents()

messagebox.showinfo("Успех", "Контрагент успешно удален")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось удалить контрагента: {str(e)}")

def add\_employee(self):

"""Добавление нового сотрудника"""

if not self.can\_edit\_employees:

messagebox.showerror("Ошибка", "У вас нет прав на добавление накладных")

return

try:

add\_window = Toplevel(self.root)

add\_window.title("Добавить сотрудника")

Label(add\_window, text="Роль:").grid(row=0, column=0, padx=5, pady=5, sticky=W)

role\_var = StringVar()

role\_combobox = ttk.Combobox(add\_window, textvariable=role\_var,

values=["Кладовщик", "Менеджер склада", "Владелец"])

role\_combobox.grid(row=0, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Фамилия:").grid(row=1, column=0, padx=5, pady=5, sticky=W)

last\_name\_entry = Entry(add\_window)

last\_name\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Имя:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

first\_name\_entry = Entry(add\_window)

first\_name\_entry.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(add\_window, text="Отчество:").grid(row=3, column=0, padx=5, pady=5, sticky=W)

patronymic\_entry = Entry(add\_window)

patronymic\_entry.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

def save\_employee():

try:

self.cursor.execute("""

INSERT INTO employee (employee\_role, last\_name, first\_name, patronymic)

VALUES (%s, %s, %s, %s)

""", (

role\_var.get(),

last\_name\_entry.get(),

first\_name\_entry.get(),

patronymic\_entry.get()

))

self.conn.commit()

self.load\_employees()

add\_window.destroy()

messagebox.showinfo("Успех", "Сотрудник успешно добавлен")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось добавить сотрудника: {str(e)}")

Button(add\_window, text="Сохранить", command=save\_employee).grid(row=4, column=0, columnspan=2, pady=10)

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось открыть форму: {str(e)}")

def edit\_employee(self):

"""Редактирование сотрудника"""

if not self.can\_edit\_employees:

messagebox.showerror("Ошибка", "У вас нет прав на добавление накладных")

return

selected = self.employee\_tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите сотрудника для редактирования")

return

item = self.employee\_tree.item(selected[0])

employee\_id = item['values'][0]

try:

self.cursor.execute("SELECT \* FROM employee WHERE employee\_id = %s", (employee\_id,))

employee\_data = self.cursor.fetchone()

if not employee\_data:

messagebox.showerror("Ошибка", "Сотрудник не найден")

return

edit\_window = Toplevel(self.root)

edit\_window.title("Редактировать сотрудника")

Label(edit\_window, text="Роль:").grid(row=0, column=0, padx=5, pady=5, sticky=W)

role\_var = StringVar()

role\_combobox = ttk.Combobox(edit\_window, textvariable=role\_var,

values=["Кладовщик", "Менеджер склада", "Владелец"])

role\_combobox.set(employee\_data[1])

role\_combobox.grid(row=0, column=1, padx=5, pady=5, sticky=EW)

Label(edit\_window, text="Фамилия:").grid(row=1, column=0, padx=5, pady=5, sticky=W)

last\_name\_entry = Entry(edit\_window)

last\_name\_entry.insert(0, employee\_data[2])

last\_name\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)

Label(edit\_window, text="Имя:").grid(row=2, column=0, padx=5, pady=5, sticky=W)

first\_name\_entry = Entry(edit\_window)

first\_name\_entry.insert(0, employee\_data[3])

first\_name\_entry.grid(row=2, column=1, padx=5, pady=5, sticky=EW)

Label(edit\_window, text="Отчество:").grid(row=3, column=0, padx=5, pady=5, sticky=W)

patronymic\_entry = Entry(edit\_window)

patronymic\_entry.insert(0, employee\_data[4])

patronymic\_entry.grid(row=3, column=1, padx=5, pady=5, sticky=EW)

def save\_changes():

try:

self.cursor.execute("""

UPDATE employee

SET

employee\_role = %s,

last\_name = %s,

first\_name = %s,

patronymic = %s

WHERE employee\_id = %s

""", (

role\_var.get(),

last\_name\_entry.get(),

first\_name\_entry.get(),

patronymic\_entry.get(),

employee\_id

))

self.conn.commit()

self.load\_employees()

edit\_window.destroy()

messagebox.showinfo("Успех", "Сотрудник успешно обновлен")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось обновить сотрудника: {str(e)}")

Button(edit\_window, text="Сохранить", command=save\_changes).grid(row=4, column=0, columnspan=2, pady=10)

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось открыть форму: {str(e)}")

def delete\_employee(self):

"""Удаление сотрудника"""

selected = self.employee\_tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите сотрудника для удаления")

return

item = self.employee\_tree.item(selected[0])

employee\_id = item['values'][0]

employee\_name = f"{item['values'][2]} {item['values'][3]} {item['values'][4]}"

if messagebox.askyesno("Подтверждение", f"Вы уверены, что хотите удалить сотрудника '{employee\_name}'?"):

try:

self.cursor.execute("DELETE FROM employee WHERE employee\_id = %s", (employee\_id,))

self.conn.commit()

self.load\_employees()

if len(self.employee\_tree.get\_children()) == 0 and self.current\_search\_conditions['employee']:

messagebox.showinfo("Информация", "Таблица пуста. Сброс условий поиска.")

self.current\_search\_conditions['employee'] = None

self.load\_employees()

messagebox.showinfo("Успех", "Сотрудник успешно удален")

except Exception as e:

self.conn.rollback()

messagebox.showerror("Ошибка", f"Не удалось удалить сотрудника: {str(e)}")

WindowApp().auth\_window()

**Приложение Д Исходный код базы данных склада запчастей**

***Листинг 1 – Исходный код базы данных склада запчастей***

SET statement\_timeout = 0;

SET lock\_timeout = 0;

SET idle\_in\_transaction\_session\_timeout = 0;

SET transaction\_timeout = 0;

SET client\_encoding = 'UTF8';

SET standard\_conforming\_strings = on;

SELECT pg\_catalog.set\_config('search\_path', '', false);

SET check\_function\_bodies = false;

SET xmloption = content;

SET client\_min\_messages = warning;

SET row\_security = off;

CREATE FUNCTION public.convert\_text\_to\_boolean(text\_value text, field\_type text DEFAULT 'status'::text) RETURNS boolean

LANGUAGE plpgsql IMMUTABLE

AS $$

BEGIN

text\_value := LOWER(TRIM(text\_value));

IF field\_type = 'type' THEN

RETURN text\_value IN ('выгрузка', 'выгрузить', 'отправка', 'true', '1', 'да', 'yes', 'y');

ELSE

RETURN text\_value IN ('завершено', 'готово', 'выполнено', 'done', 'true', '1', 'да', 'yes', 'y');

END IF;

END;

$$;

ALTER FUNCTION public.convert\_text\_to\_boolean(text\_value text, field\_type text) OWNER TO postgres;

CREATE FUNCTION public.delete\_invoice\_details\_view() RETURNS trigger

LANGUAGE plpgsql SECURITY DEFINER

AS $$

BEGIN

DELETE FROM invoice\_employee WHERE invoiceID = OLD.invoice\_id;

DELETE FROM invoice\_detail WHERE invoiceID = OLD.invoice\_id;

DELETE FROM invoice WHERE invoice\_id = OLD.invoice\_id;

RETURN OLD;

END;

$$;

ALTER FUNCTION public.delete\_invoice\_details\_view() OWNER TO postgres;

CREATE FUNCTION public.delete\_related\_data() RETURNS trigger

LANGUAGE plpgsql SECURITY DEFINER

AS $$

BEGIN

IF TG\_TABLE\_NAME = 'warehouse' THEN

PERFORM \* FROM details WHERE shelfID IN (

SELECT shelf\_id FROM shelf WHERE rackID IN (

SELECT rack\_id FROM rack WHERE roomID IN (

SELECT room\_id FROM room WHERE warehouseID = OLD.warehouse\_id

)

)

) LIMIT 1;

DELETE FROM shelf WHERE rackID IN (

SELECT rack\_id FROM rack WHERE roomID IN (

SELECT room\_id FROM room WHERE warehouseID = OLD.warehouse\_id

)

);

DELETE FROM rack WHERE roomID IN (

SELECT room\_id FROM room WHERE warehouseID = OLD.warehouse\_id

);

DELETE FROM room WHERE warehouseID = OLD.warehouse\_id;

ELSIF TG\_TABLE\_NAME = 'room' THEN

PERFORM \* FROM details WHERE shelfID IN (

SELECT shelf\_id FROM shelf WHERE rackID IN (

SELECT rack\_id FROM rack WHERE roomID = OLD.room\_id

)

) LIMIT 1;

DELETE FROM shelf WHERE rackID IN (

SELECT rack\_id FROM rack WHERE roomID = OLD.room\_id

);

DELETE FROM rack WHERE roomID = OLD.room\_id;

ELSIF TG\_TABLE\_NAME = 'rack' THEN

PERFORM \* FROM details WHERE shelfID IN (

SELECT shelf\_id FROM shelf WHERE rackID = OLD.rack\_id

) LIMIT 1;

DELETE FROM shelf WHERE rackID = OLD.rack\_id;

ELSIF TG\_TABLE\_NAME = 'shelf' THEN

PERFORM \* FROM details WHERE shelfID = OLD.shelf\_id LIMIT 1;

END IF;

RETURN OLD;

END;

$$;

ALTER FUNCTION public.delete\_related\_data() OWNER TO postgres;

CREATE FUNCTION public.delete\_warehouse\_details() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE

warehouse\_id\_value integer;

room\_id\_value integer;

rack\_id\_value integer;

shelf\_id\_value integer;

BEGIN

SELECT w.warehouse\_id INTO warehouse\_id\_value

FROM warehouse w

WHERE w.warehouse\_number = OLD.warehouse\_number;

SELECT r.room\_id INTO room\_id\_value

FROM room r

WHERE r.warehouseID = warehouse\_id\_value AND r.room\_number = OLD.room\_number;

SELECT ra.rack\_id INTO rack\_id\_value

FROM rack ra

WHERE ra.roomID = room\_id\_value AND ra.rack\_number = OLD.rack\_number;

SELECT s.shelf\_id INTO shelf\_id\_value

FROM shelf s

WHERE s.rackID = rack\_id\_value AND s.shelf\_number = OLD.shelf\_number;

DELETE FROM details WHERE detail\_id = OLD.detail\_id;

DELETE FROM shelf

WHERE shelf.shelf\_id = shelf\_id\_value

AND NOT EXISTS (SELECT 1 FROM details WHERE details.shelfID = shelf.shelf\_id);

DELETE FROM rack

WHERE rack.rack\_id = rack\_id\_value

AND NOT EXISTS (SELECT 1 FROM shelf WHERE shelf.rackID = rack.rack\_id);

DELETE FROM room

WHERE room.room\_id = room\_id\_value

AND NOT EXISTS (SELECT 1 FROM rack WHERE rack.roomID = room.room\_id);

DELETE FROM warehouse

WHERE warehouse.warehouse\_id = warehouse\_id\_value

AND NOT EXISTS (SELECT 1 FROM room WHERE room.warehouseID = warehouse.warehouse\_id);

RETURN OLD;

END;

$$;

ALTER FUNCTION public.delete\_warehouse\_details() OWNER TO postgres;

CREATE FUNCTION public.get\_employee\_id(p\_last\_name character varying, p\_first\_name character varying, p\_patronymic character varying) RETURNS integer

LANGUAGE plpgsql SECURITY DEFINER

AS $$

DECLARE

v\_id integer;

BEGIN

SELECT employee\_id INTO v\_id

FROM employee

WHERE last\_name = p\_last\_name

AND first\_name = p\_first\_name

AND patronymic = p\_patronymic;

IF NOT FOUND THEN

RAISE EXCEPTION 'Сотрудник % % % не найден',

p\_last\_name, p\_first\_name, p\_patronymic;

END IF;

RETURN v\_id;

END;

$$;

ALTER FUNCTION public.get\_employee\_id(p\_last\_name character varying, p\_first\_name character varying, p\_patronymic character varying) OWNER TO postgres;

CREATE FUNCTION public.insert\_into\_warehouse\_details() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE

warehouse\_id integer;

room\_id integer;

rack\_id integer;

shelf\_id integer;

BEGIN

IF NOT EXISTS (SELECT 1 FROM warehouse WHERE warehouse\_number = NEW.warehouse\_number) THEN

INSERT INTO warehouse (warehouse\_number, address)

VALUES (NEW.warehouse\_number, 'default address')

RETURNING warehouse.warehouse\_id INTO warehouse\_id;

ELSE

SELECT warehouse.warehouse\_id INTO warehouse\_id FROM warehouse WHERE warehouse\_number = NEW.warehouse\_number;

END IF;

IF NOT EXISTS (SELECT 1 FROM room WHERE room\_number = NEW.room\_number AND warehouseID = warehouse\_id) THEN

INSERT INTO room (room\_number, warehouseID)

VALUES (NEW.room\_number, warehouse\_id)

RETURNING room.room\_id INTO room\_id;

ELSE

SELECT room.room\_id INTO room\_id FROM room WHERE room\_number = NEW.room\_number AND warehouseID = warehouse\_id;

END IF;

IF NOT EXISTS (SELECT 1 FROM rack WHERE rack\_number = NEW.rack\_number AND roomID = room\_id) THEN

INSERT INTO rack (rack\_number, roomID)

VALUES (NEW.rack\_number, room\_id)

RETURNING rack.rack\_id INTO rack\_id;

ELSE

SELECT rack.rack\_id INTO rack\_id FROM rack WHERE rack\_number = NEW.rack\_number AND roomID = room\_id;

END IF;

IF NOT EXISTS (SELECT 1 FROM shelf WHERE shelf\_number = NEW.shelf\_number AND rackID = rack\_id) THEN

INSERT INTO shelf (shelf\_number, rackID)

VALUES (NEW.shelf\_number, rack\_id)

RETURNING shelf.shelf\_id INTO shelf\_id;

ELSE

SELECT shelf.shelf\_id INTO shelf\_id FROM shelf WHERE shelf\_number = NEW.shelf\_number AND rackID = rack\_id;

END IF;

INSERT INTO details (shelfID, weight, type\_detail)

VALUES (shelf\_id, NEW.weight, NEW.type\_detail);

RETURN NEW;

END;

$$;

ALTER FUNCTION public.insert\_into\_warehouse\_details() OWNER TO postgres;

CREATE FUNCTION public.insert\_invoice\_details\_view() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE

v\_invoice\_id INTEGER;

v\_counteragent\_id INTEGER;

v\_detail\_id INTEGER;

v\_employee\_id INTEGER;

v\_type\_invoice BOOLEAN;

v\_status BOOLEAN;

BEGIN

SELECT counteragent\_id INTO v\_counteragent\_id

FROM counteragent

WHERE counteragent\_name = NEW.counteragent\_name;

IF NOT FOUND THEN

RAISE EXCEPTION 'Контрагент с именем % не найден', NEW.counteragent\_name;

END IF;

IF NEW.type\_invoice\_text IS NOT NULL THEN

v\_type\_invoice := convert\_text\_to\_boolean(NEW.type\_invoice\_text, 'type');

ELSE

v\_type\_invoice := COALESCE(NEW.type\_invoice\_bool, FALSE);

END IF;

IF NEW.status\_text IS NOT NULL THEN

v\_status := convert\_text\_to\_boolean(NEW.status\_text, 'status');

ELSE

v\_status := COALESCE(NEW.status\_bool, FALSE);

END IF;

IF NEW.invoice\_id IS NOT NULL THEN

PERFORM 1 FROM invoice WHERE invoice\_id = NEW.invoice\_id;

IF FOUND THEN

UPDATE invoice SET

counteragentID = v\_counteragent\_id,

date\_time = NEW.date\_time,

type\_invoice = v\_type\_invoice,

status = v\_status

WHERE invoice\_id = NEW.invoice\_id;

v\_invoice\_id := NEW.invoice\_id;

ELSE

INSERT INTO invoice (invoice\_id, counteragentID, date\_time, type\_invoice, status)

VALUES (NEW.invoice\_id, v\_counteragent\_id, NEW.date\_time, v\_type\_invoice, v\_status)

RETURNING invoice\_id INTO v\_invoice\_id;

END IF;

ELSE

INSERT INTO invoice (counteragentID, date\_time, type\_invoice, status)

VALUES (v\_counteragent\_id, NEW.date\_time, v\_type\_invoice, v\_status)

RETURNING invoice\_id INTO v\_invoice\_id;

END IF;

SELECT detail\_id INTO v\_detail\_id

FROM details

WHERE type\_detail = NEW.type\_detail;

IF NOT FOUND THEN

RAISE EXCEPTION 'Деталь типа % не найдена', NEW.type\_detail;

END IF;

INSERT INTO invoice\_detail (invoiceID, detailID, quantity)

VALUES (v\_invoice\_id, v\_detail\_id, NEW.quantity)

ON CONFLICT (invoiceID, detailID)

DO UPDATE SET quantity = invoice\_detail.quantity + NEW.quantity;

v\_employee\_id := get\_employee\_id(

NEW.responsible\_last\_name,

NEW.responsible\_first\_name,

NEW.responsible\_patronymic

);

INSERT INTO invoice\_employee (invoiceID, responsible, granted\_access, when\_granted)

VALUES (v\_invoice\_id, v\_employee\_id, v\_employee\_id, NOW())

ON CONFLICT (invoiceID, responsible) DO NOTHING;

NEW.invoice\_id := v\_invoice\_id;

RETURN NEW;

END;

$$;

ALTER FUNCTION public.insert\_invoice\_details\_view() OWNER TO postgres;

CREATE FUNCTION public.log\_counteragent\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('counteragent', 'INSERT', NEW.counteragent\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('counteragent', 'UPDATE', OLD.counteragent\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('counteragent', 'DELETE', OLD.counteragent\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_counteragent\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_details\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('details', 'INSERT', NEW.detail\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('details', 'UPDATE', OLD.detail\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('details', 'DELETE', OLD.detail\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_details\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_employee\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('employee', 'INSERT', NEW.employee\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('employee', 'UPDATE', OLD.employee\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('employee', 'DELETE', OLD.employee\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_employee\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_invoice\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('invoice', 'INSERT', NEW.invoice\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('invoice', 'UPDATE', OLD.invoice\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('invoice', 'DELETE', OLD.invoice\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_invoice\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_invoice\_detail\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('invoice\_detail', 'INSERT', NEW.invoiceID, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('invoice\_detail', 'UPDATE', OLD.invoiceID, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('invoice\_detail', 'DELETE', OLD.invoiceID, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_invoice\_detail\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_invoice\_employee\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('invoice\_employee', 'INSERT', NEW.invoiceID, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('invoice\_employee', 'UPDATE', OLD.invoiceID, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('invoice\_employee', 'DELETE', OLD.invoiceID, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_invoice\_employee\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_rack\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('rack', 'INSERT', NEW.rack\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('rack', 'UPDATE', OLD.rack\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('rack', 'DELETE', OLD.rack\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_rack\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_room\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('room', 'INSERT', NEW.room\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('room', 'UPDATE', OLD.room\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('room', 'DELETE', OLD.room\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_room\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_shelf\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('shelf', 'INSERT', NEW.shelf\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('shelf', 'UPDATE', OLD.shelf\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('shelf', 'DELETE', OLD.shelf\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_shelf\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_warehouse\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, new\_values)

VALUES ('warehouse', 'INSERT', NEW.warehouse\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values, new\_values)

VALUES ('warehouse', 'UPDATE', OLD.warehouse\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO log\_table (table\_name, action\_type, record\_id, old\_values)

VALUES ('warehouse', 'DELETE', OLD.warehouse\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_warehouse\_changes() OWNER TO postgres;

CREATE FUNCTION public.update\_invoice\_details\_view() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE

v\_updated BOOLEAN := FALSE;

BEGIN

IF NEW.type\_invoice\_text IS DISTINCT FROM OLD.type\_invoice\_text THEN

UPDATE invoice SET

type\_invoice = convert\_text\_to\_boolean(NEW.type\_invoice\_text, 'type')

WHERE invoice\_id = NEW.invoice\_id;

v\_updated := TRUE;

END IF;

IF NEW.status\_text IS DISTINCT FROM OLD.status\_text THEN

UPDATE invoice SET

status = convert\_text\_to\_boolean(NEW.status\_text, 'status')

WHERE invoice\_id = NEW.invoice\_id;

v\_updated := TRUE;

END IF;

IF NOT v\_updated AND (

NEW.invoice\_id IS DISTINCT FROM OLD.invoice\_id OR

NEW.counteragent\_name IS DISTINCT FROM OLD.counteragent\_name OR

NEW.date\_time IS DISTINCT FROM OLD.date\_time OR

NEW.type\_detail IS DISTINCT FROM OLD.type\_detail OR

NEW.quantity IS DISTINCT FROM OLD.quantity OR

NEW.responsible\_last\_name IS DISTINCT FROM OLD.responsible\_last\_name OR

NEW.responsible\_first\_name IS DISTINCT FROM OLD.responsible\_first\_name OR

NEW.responsible\_patronymic IS DISTINCT FROM OLD.responsible\_patronymic

) THEN

RAISE EXCEPTION 'Разрешено обновлять только поля type\_invoice\_text и status\_text';

END IF;

RETURN NEW;

END;

$$;

ALTER FUNCTION public.update\_invoice\_details\_view() OWNER TO postgres;

CREATE FUNCTION public.update\_invoice\_status() RETURNS trigger

LANGUAGE plpgsql SECURITY DEFINER

AS $$

BEGIN

IF TG\_OP = 'UPDATE' AND (

OLD.invoice\_id IS DISTINCT FROM NEW.invoice\_id OR

OLD.counteragent\_name IS DISTINCT FROM NEW.counteragent\_name OR

OLD.date\_time IS DISTINCT FROM NEW.date\_time OR

OLD.type\_invoice IS DISTINCT FROM NEW.type\_invoice OR

OLD.type\_detail IS DISTINCT FROM NEW.type\_detail OR

OLD.quantity IS DISTINCT FROM NEW.quantity OR

OLD.responsible\_last\_name IS DISTINCT FROM NEW.responsible\_last\_name OR

OLD.responsible\_first\_name IS DISTINCT FROM NEW.responsible\_first\_name OR

OLD.responsible\_patronymic IS DISTINCT FROM NEW.responsible\_patronymic OR

OLD.responsible\_id IS DISTINCT FROM NEW.responsible\_id

) THEN

RAISE EXCEPTION 'Разрешено обновлять только поле status. Попытка изменить другие поля запрещена.';

END IF;

IF OLD.status IS NOT DISTINCT FROM NEW.status THEN

RETURN NEW;

END IF;

UPDATE invoice SET status = NEW.status

WHERE invoice\_id = NEW.invoice\_id;

RETURN NEW;

END;

$$;

ALTER FUNCTION public.update\_invoice\_status() OWNER TO postgres;

CREATE FUNCTION public.update\_warehouse\_details\_view() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE details

SET type\_detail = NEW.type\_detail, weight = NEW.weight

WHERE detail\_id = OLD.detail\_id;

UPDATE shelf

SET shelf\_number = NEW.shelf\_number

WHERE shelf\_id = (SELECT shelfID FROM details WHERE detail\_id = OLD.detail\_id);

UPDATE rack

SET rack\_number = NEW.rack\_number

WHERE rack\_id = (SELECT rackID FROM shelf WHERE shelf\_id = (SELECT shelfID FROM details WHERE detail\_id = OLD.detail\_id));

UPDATE room

SET room\_number = NEW.room\_number

WHERE room\_id = (SELECT roomID FROM rack WHERE rack\_id = (SELECT rackID FROM shelf WHERE shelf\_id = (SELECT shelfID FROM details WHERE detail\_id = OLD.detail\_id)));

UPDATE warehouse

SET warehouse\_number = NEW.warehouse\_number

WHERE warehouse\_id = (SELECT warehouseID FROM room WHERE room\_id = (SELECT roomID FROM rack WHERE rack\_id = (SELECT rackID FROM shelf WHERE shelf\_id = (SELECT shelfID FROM details WHERE detail\_id = OLD.detail\_id))));

RETURN NEW;

END;

$$;

ALTER FUNCTION public.update\_warehouse\_details\_view() OWNER TO postgres;

SET default\_tablespace = '';

SET default\_table\_access\_method = heap;

CREATE TABLE public.counteragent (

counteragent\_id integer NOT NULL,

counteragent\_name character varying(128) NOT NULL,

contact\_person character varying(128) NOT NULL,

phone\_number bigint NOT NULL,

address text NOT NULL

);

ALTER TABLE public.counteragent OWNER TO postgres;

CREATE SEQUENCE public.counteragent\_counteragent\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.counteragent\_counteragent\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.counteragent\_counteragent\_id\_seq OWNED BY public.counteragent.counteragent\_id;

CREATE TABLE public.details (

detail\_id integer NOT NULL,

shelfid integer NOT NULL,

weight double precision NOT NULL,

type\_detail text NOT NULL

);

ALTER TABLE public.details OWNER TO postgres;

CREATE SEQUENCE public.details\_detail\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.details\_detail\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.details\_detail\_id\_seq OWNED BY public.details.detail\_id;

CREATE SEQUENCE public.details\_shelfid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.details\_shelfid\_seq OWNER TO postgres;

ALTER SEQUENCE public.details\_shelfid\_seq OWNED BY public.details.shelfid;

CREATE TABLE public.employee (

employee\_id integer NOT NULL,

employee\_role character varying(25) NOT NULL,

last\_name character varying(35) NOT NULL,

first\_name character varying(35) NOT NULL,

patronymic character varying(35) NOT NULL

);

ALTER TABLE public.employee OWNER TO postgres;

CREATE SEQUENCE public.employee\_employee\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.employee\_employee\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.employee\_employee\_id\_seq OWNED BY public.employee.employee\_id;

CREATE TABLE public.invoice (

invoice\_id integer NOT NULL,

counteragentid integer NOT NULL,

date\_time timestamp without time zone NOT NULL,

type\_invoice boolean NOT NULL,

status boolean NOT NULL

);

ALTER TABLE public.invoice OWNER TO postgres;

CREATE SEQUENCE public.invoice\_counteragentid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_counteragentid\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_counteragentid\_seq OWNED BY public.invoice.counteragentid;

CREATE TABLE public.invoice\_detail (

invoiceid integer NOT NULL,

detailid integer NOT NULL,

quantity integer NOT NULL

);

ALTER TABLE public.invoice\_detail OWNER TO postgres;

CREATE SEQUENCE public.invoice\_detail\_detailid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_detail\_detailid\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_detail\_detailid\_seq OWNED BY public.invoice\_detail.detailid;

CREATE SEQUENCE public.invoice\_detail\_invoiceid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_detail\_invoiceid\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_detail\_invoiceid\_seq OWNED BY public.invoice\_detail.invoiceid;

CREATE TABLE public.invoice\_employee (

invoiceid integer NOT NULL,

responsible integer NOT NULL,

granted\_access integer NOT NULL,

when\_granted timestamp without time zone NOT NULL

);

ALTER TABLE public.invoice\_employee OWNER TO postgres;

CREATE VIEW public.invoice\_details\_view AS

SELECT inv.invoice\_id,

ca.counteragent\_name,

inv.date\_time,

CASE

WHEN inv.type\_invoice THEN 'Выгрузка'::text

ELSE 'Отгрузка'::text

END AS type\_invoice\_text,

CASE

WHEN inv.status THEN 'Завершено'::text

ELSE 'В процессе'::text

END AS status\_text,

det.type\_detail,

invd.quantity,

emp.last\_name AS responsible\_last\_name,

emp.first\_name AS responsible\_first\_name,

emp.patronymic AS responsible\_patronymic,

emp.employee\_id AS responsible\_id,

inv.status AS status\_bool,

inv.type\_invoice AS type\_invoice\_bool

FROM (((((public.invoice inv

JOIN public.invoice\_detail invd ON ((inv.invoice\_id = invd.invoiceid)))

JOIN public.details det ON ((invd.detailid = det.detail\_id)))

JOIN public.invoice\_employee inv\_emp ON ((inv.invoice\_id = inv\_emp.invoiceid)))

JOIN public.employee emp ON ((inv\_emp.responsible = emp.employee\_id)))

JOIN public.counteragent ca ON ((inv.counteragentid = ca.counteragent\_id)));

ALTER VIEW public.invoice\_details\_view OWNER TO postgres;

CREATE SEQUENCE public.invoice\_employee\_granted\_access\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_employee\_granted\_access\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_employee\_granted\_access\_seq OWNED BY public.invoice\_employee.granted\_access;

CREATE SEQUENCE public.invoice\_employee\_invoiceid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_employee\_invoiceid\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_employee\_invoiceid\_seq OWNED BY public.invoice\_employee.invoiceid;

CREATE SEQUENCE public.invoice\_employee\_responsible\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_employee\_responsible\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_employee\_responsible\_seq OWNED BY public.invoice\_employee.responsible;

CREATE SEQUENCE public.invoice\_invoice\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_invoice\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_invoice\_id\_seq OWNED BY public.invoice.invoice\_id;

CREATE TABLE public.log\_table (

log\_id integer NOT NULL,

table\_name character varying(50) NOT NULL,

action\_type character varying(20) NOT NULL,

record\_id integer,

action\_time timestamp without time zone DEFAULT CURRENT\_TIMESTAMP,

old\_values jsonb,

new\_values jsonb

);

ALTER TABLE public.log\_table OWNER TO postgres;

CREATE SEQUENCE public.log\_table\_log\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.log\_table\_log\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.log\_table\_log\_id\_seq OWNED BY public.log\_table.log\_id;

CREATE TABLE public.rack (

rack\_id integer NOT NULL,

roomid integer NOT NULL,

rack\_number integer NOT NULL

);

ALTER TABLE public.rack OWNER TO postgres;

CREATE SEQUENCE public.rack\_rack\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.rack\_rack\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.rack\_rack\_id\_seq OWNED BY public.rack.rack\_id;

CREATE SEQUENCE public.rack\_roomid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.rack\_roomid\_seq OWNER TO postgres;

ALTER SEQUENCE public.rack\_roomid\_seq OWNED BY public.rack.roomid;

CREATE TABLE public.room (

room\_id integer NOT NULL,

warehouseid integer NOT NULL,

room\_number integer NOT NULL

);

ALTER TABLE public.room OWNER TO postgres;

CREATE SEQUENCE public.room\_room\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.room\_room\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.room\_room\_id\_seq OWNED BY public.room.room\_id;

CREATE SEQUENCE public.room\_warehouseid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.room\_warehouseid\_seq OWNER TO postgres;

ALTER SEQUENCE public.room\_warehouseid\_seq OWNED BY public.room.warehouseid;

CREATE TABLE public.shelf (

shelf\_id integer NOT NULL,

rackid integer NOT NULL,

shelf\_number integer NOT NULL

);

ALTER TABLE public.shelf OWNER TO postgres;

CREATE SEQUENCE public.shelf\_rackid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.shelf\_rackid\_seq OWNER TO postgres;

ALTER SEQUENCE public.shelf\_rackid\_seq OWNED BY public.shelf.rackid;

CREATE SEQUENCE public.shelf\_shelf\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.shelf\_shelf\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.shelf\_shelf\_id\_seq OWNED BY public.shelf.shelf\_id;

CREATE TABLE public.warehouse (

warehouse\_id integer NOT NULL,

warehouse\_number integer NOT NULL,

address text NOT NULL

);

ALTER TABLE public.warehouse OWNER TO postgres;

CREATE VIEW public.warehouse\_details\_view AS

SELECT w.warehouse\_number,

r.room\_number,

rk.rack\_number,

s.shelf\_number,

d.type\_detail,

d.weight,

d.detail\_id

FROM ((((public.warehouse w

JOIN public.room r ON ((w.warehouse\_id = r.warehouseid)))

JOIN public.rack rk ON ((r.room\_id = rk.roomid)))

JOIN public.shelf s ON ((rk.rack\_id = s.rackid)))

JOIN public.details d ON ((s.shelf\_id = d.shelfid)));

ALTER VIEW public.warehouse\_details\_view OWNER TO postgres;

CREATE SEQUENCE public.warehouse\_warehouse\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.warehouse\_warehouse\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.warehouse\_warehouse\_id\_seq OWNED BY public.warehouse.warehouse\_id;

ALTER TABLE ONLY public.counteragent ALTER COLUMN counteragent\_id SET DEFAULT nextval('public.counteragent\_counteragent\_id\_seq'::regclass);

ALTER TABLE ONLY public.details ALTER COLUMN detail\_id SET DEFAULT nextval('public.details\_detail\_id\_seq'::regclass);

ALTER TABLE ONLY public.details ALTER COLUMN shelfid SET DEFAULT nextval('public.details\_shelfid\_seq'::regclass);

ALTER TABLE ONLY public.employee ALTER COLUMN employee\_id SET DEFAULT nextval('public.employee\_employee\_id\_seq'::regclass);

ALTER TABLE ONLY public.invoice ALTER COLUMN invoice\_id SET DEFAULT nextval('public.invoice\_invoice\_id\_seq'::regclass);

ALTER TABLE ONLY public.invoice ALTER COLUMN counteragentid SET DEFAULT nextval('public.invoice\_counteragentid\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_detail ALTER COLUMN invoiceid SET DEFAULT nextval('public.invoice\_detail\_invoiceid\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_detail ALTER COLUMN detailid SET DEFAULT nextval('public.invoice\_detail\_detailid\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_employee ALTER COLUMN invoiceid SET DEFAULT nextval('public.invoice\_employee\_invoiceid\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_employee ALTER COLUMN responsible SET DEFAULT nextval('public.invoice\_employee\_responsible\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_employee ALTER COLUMN granted\_access SET DEFAULT nextval('public.invoice\_employee\_granted\_access\_seq'::regclass);

ALTER TABLE ONLY public.log\_table ALTER COLUMN log\_id SET DEFAULT nextval('public.log\_table\_log\_id\_seq'::regclass);

ALTER TABLE ONLY public.rack ALTER COLUMN rack\_id SET DEFAULT nextval('public.rack\_rack\_id\_seq'::regclass);

ALTER TABLE ONLY public.rack ALTER COLUMN roomid SET DEFAULT nextval('public.rack\_roomid\_seq'::regclass);

ALTER TABLE ONLY public.room ALTER COLUMN room\_id SET DEFAULT nextval('public.room\_room\_id\_seq'::regclass);

ALTER TABLE ONLY public.room ALTER COLUMN warehouseid SET DEFAULT nextval('public.room\_warehouseid\_seq'::regclass);

ALTER TABLE ONLY public.shelf ALTER COLUMN shelf\_id SET DEFAULT nextval('public.shelf\_shelf\_id\_seq'::regclass);

ALTER TABLE ONLY public.shelf ALTER COLUMN rackid SET DEFAULT nextval('public.shelf\_rackid\_seq'::regclass);

ALTER TABLE ONLY public.warehouse ALTER COLUMN warehouse\_id SET DEFAULT nextval('public.warehouse\_warehouse\_id\_seq'::regclass);

SELECT pg\_catalog.setval('public.counteragent\_counteragent\_id\_seq', 6, true);

SELECT pg\_catalog.setval('public.details\_detail\_id\_seq', 47, true);

SELECT pg\_catalog.setval('public.details\_shelfid\_seq', 1, false);

SELECT pg\_catalog.setval('public.employee\_employee\_id\_seq', 6, true);

SELECT pg\_catalog.setval('public.invoice\_counteragentid\_seq', 1, false);

SELECT pg\_catalog.setval('public.invoice\_detail\_detailid\_seq', 1, false);

SELECT pg\_catalog.setval('public.invoice\_detail\_invoiceid\_seq', 1, false);

SELECT pg\_catalog.setval('public.invoice\_employee\_granted\_access\_seq', 1, false);

SELECT pg\_catalog.setval('public.invoice\_employee\_invoiceid\_seq', 1, false);

SELECT pg\_catalog.setval('public.invoice\_employee\_responsible\_seq', 1, false);

SELECT pg\_catalog.setval('public.invoice\_invoice\_id\_seq', 13, true);

SELECT pg\_catalog.setval('public.log\_table\_log\_id\_seq', 1954, true);

SELECT pg\_catalog.setval('public.rack\_rack\_id\_seq', 126, true);

SELECT pg\_catalog.setval('public.rack\_roomid\_seq', 1, false);

SELECT pg\_catalog.setval('public.room\_room\_id\_seq', 26, true);

SELECT pg\_catalog.setval('public.room\_warehouseid\_seq', 1, false);

SELECT pg\_catalog.setval('public.shelf\_rackid\_seq', 1, false);

SELECT pg\_catalog.setval('public.shelf\_shelf\_id\_seq', 625, true);

SELECT pg\_catalog.setval('public.warehouse\_warehouse\_id\_seq', 9, true);

ALTER TABLE ONLY public.counteragent

ADD CONSTRAINT counteragent\_pkey PRIMARY KEY (counteragent\_id);

ALTER TABLE ONLY public.details

ADD CONSTRAINT details\_pkey PRIMARY KEY (detail\_id);

ALTER TABLE ONLY public.employee

ADD CONSTRAINT employee\_pkey PRIMARY KEY (employee\_id);

ALTER TABLE ONLY public.invoice\_detail

ADD CONSTRAINT invoice\_detail\_unique UNIQUE (invoiceid, detailid);

ALTER TABLE ONLY public.invoice\_employee

ADD CONSTRAINT invoice\_employee\_unique UNIQUE (invoiceid, responsible);

ALTER TABLE ONLY public.invoice

ADD CONSTRAINT invoice\_pkey PRIMARY KEY (invoice\_id);

ALTER TABLE ONLY public.log\_table

ADD CONSTRAINT log\_table\_pkey PRIMARY KEY (log\_id);

ALTER TABLE ONLY public.rack

ADD CONSTRAINT rack\_pkey PRIMARY KEY (rack\_id);

ALTER TABLE ONLY public.room

ADD CONSTRAINT room\_pkey PRIMARY KEY (room\_id);

ALTER TABLE ONLY public.shelf

ADD CONSTRAINT shelf\_pkey PRIMARY KEY (shelf\_id);

ALTER TABLE ONLY public.warehouse

ADD CONSTRAINT unique\_address UNIQUE (address);

ALTER TABLE ONLY public.warehouse

ADD CONSTRAINT warehouse\_pkey PRIMARY KEY (warehouse\_id);

CREATE TRIGGER counteragent\_changes AFTER INSERT OR DELETE OR UPDATE ON public.counteragent FOR EACH ROW EXECUTE FUNCTION public.log\_counteragent\_changes();

CREATE TRIGGER delete\_related\_data BEFORE DELETE ON public.rack FOR EACH ROW EXECUTE FUNCTION public.delete\_related\_data();

CREATE TRIGGER delete\_related\_data BEFORE DELETE ON public.room FOR EACH ROW EXECUTE FUNCTION public.delete\_related\_data();

CREATE TRIGGER delete\_related\_data BEFORE DELETE ON public.shelf FOR EACH ROW EXECUTE FUNCTION public.delete\_related\_data();

CREATE TRIGGER delete\_related\_data BEFORE DELETE ON public.warehouse FOR EACH ROW EXECUTE FUNCTION public.delete\_related\_data();

CREATE TRIGGER details\_changes AFTER INSERT OR DELETE OR UPDATE ON public.details FOR EACH ROW EXECUTE FUNCTION public.log\_details\_changes();

CREATE TRIGGER employee\_changes AFTER INSERT OR DELETE OR UPDATE ON public.employee FOR EACH ROW EXECUTE FUNCTION public.log\_employee\_changes();

CREATE TRIGGER invoice\_changes AFTER INSERT OR DELETE OR UPDATE ON public.invoice FOR EACH ROW EXECUTE FUNCTION public.log\_invoice\_changes();

CREATE TRIGGER invoice\_detail\_changes AFTER INSERT OR DELETE OR UPDATE ON public.invoice\_detail FOR EACH ROW EXECUTE FUNCTION public.log\_invoice\_detail\_changes();

CREATE TRIGGER invoice\_details\_view\_delete INSTEAD OF DELETE ON public.invoice\_details\_view FOR EACH ROW EXECUTE FUNCTION public.delete\_invoice\_details\_view();

CREATE TRIGGER invoice\_details\_view\_insert INSTEAD OF INSERT ON public.invoice\_details\_view FOR EACH ROW EXECUTE FUNCTION public.insert\_invoice\_details\_view();

CREATE TRIGGER invoice\_details\_view\_update INSTEAD OF UPDATE ON public.invoice\_details\_view FOR EACH ROW EXECUTE FUNCTION public.update\_invoice\_details\_view();

CREATE TRIGGER invoice\_employee\_changes AFTER INSERT OR DELETE OR UPDATE ON public.invoice\_employee FOR EACH ROW EXECUTE FUNCTION public.log\_invoice\_employee\_changes();

CREATE TRIGGER rack\_changes AFTER INSERT OR DELETE OR UPDATE ON public.rack FOR EACH ROW EXECUTE FUNCTION public.log\_rack\_changes();

CREATE TRIGGER room\_changes AFTER INSERT OR DELETE OR UPDATE ON public.room FOR EACH ROW EXECUTE FUNCTION public.log\_room\_changes();

CREATE TRIGGER shelf\_changes AFTER INSERT OR DELETE OR UPDATE ON public.shelf FOR EACH ROW EXECUTE FUNCTION public.log\_shelf\_changes();

CREATE TRIGGER trg\_delete\_invoice\_details AFTER DELETE ON public.invoice FOR EACH ROW EXECUTE FUNCTION public.delete\_invoice\_details\_view();

CREATE TRIGGER trg\_delete\_related\_data BEFORE DELETE ON public.warehouse FOR EACH ROW EXECUTE FUNCTION public.delete\_related\_data();

CREATE TRIGGER warehouse\_changes AFTER INSERT OR DELETE OR UPDATE ON public.warehouse FOR EACH ROW EXECUTE FUNCTION public.log\_warehouse\_changes();

ALTER TABLE ONLY public.details

ADD CONSTRAINT details\_shelfid\_fkey FOREIGN KEY (shelfid) REFERENCES public.shelf(shelf\_id);

ALTER TABLE ONLY public.invoice

ADD CONSTRAINT invoice\_counteragentid\_fkey FOREIGN KEY (counteragentid) REFERENCES public.counteragent(counteragent\_id);

ALTER TABLE ONLY public.invoice\_detail

ADD CONSTRAINT invoice\_detail\_detailid\_fkey FOREIGN KEY (detailid) REFERENCES public.details(detail\_id);

ALTER TABLE ONLY public.invoice\_detail

ADD CONSTRAINT invoice\_detail\_invoiceid\_fkey FOREIGN KEY (invoiceid) REFERENCES public.invoice(invoice\_id);

ALTER TABLE ONLY public.invoice\_employee

ADD CONSTRAINT invoice\_employee\_granted\_access\_fkey FOREIGN KEY (granted\_access) REFERENCES public.employee(employee\_id);

ALTER TABLE ONLY public.invoice\_employee

ADD CONSTRAINT invoice\_employee\_invoiceid\_fkey FOREIGN KEY (invoiceid) REFERENCES public.invoice(invoice\_id);

ALTER TABLE ONLY public.invoice\_employee

ADD CONSTRAINT invoice\_employee\_responsible\_fkey FOREIGN KEY (responsible) REFERENCES public.employee(employee\_id);

ALTER TABLE ONLY public.rack

ADD CONSTRAINT rack\_roomid\_fkey FOREIGN KEY (roomid) REFERENCES public.room(room\_id);

ALTER TABLE ONLY public.room

ADD CONSTRAINT room\_warehouseid\_fkey FOREIGN KEY (warehouseid) REFERENCES public.warehouse(warehouse\_id);

ALTER TABLE ONLY public.shelf

ADD CONSTRAINT shelf\_rackid\_fkey FOREIGN KEY (rackid) REFERENCES public.rack(rack\_id);

GRANT USAGE ON SCHEMA public TO warehouse\_owner;

GRANT ALL ON FUNCTION public.convert\_text\_to\_boolean(text\_value text, field\_type text) TO warehouse\_owner;

GRANT ALL ON FUNCTION public.delete\_invoice\_details\_view() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.delete\_related\_data() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.delete\_warehouse\_details() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.get\_employee\_id(p\_last\_name character varying, p\_first\_name character varying, p\_patronymic character varying) TO warehouse\_owner;

GRANT ALL ON FUNCTION public.get\_employee\_id(p\_last\_name character varying, p\_first\_name character varying, p\_patronymic character varying) TO warehouse\_manager;

GRANT ALL ON FUNCTION public.insert\_into\_warehouse\_details() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.insert\_invoice\_details\_view() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.insert\_invoice\_details\_view() TO warehouse\_manager;

GRANT ALL ON FUNCTION public.log\_counteragent\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.log\_details\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.log\_employee\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.log\_invoice\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.log\_invoice\_detail\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.log\_invoice\_employee\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.log\_rack\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.log\_room\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.log\_shelf\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.log\_warehouse\_changes() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.update\_invoice\_details\_view() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.update\_invoice\_status() TO warehouse\_owner;

GRANT ALL ON FUNCTION public.update\_warehouse\_details\_view() TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.counteragent TO warehouse\_owner;

GRANT SELECT ON TABLE public.counteragent TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.counteragent\_counteragent\_id\_seq TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.details TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.details TO warehouse\_clerk;

GRANT SELECT ON TABLE public.details TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.details\_detail\_id\_seq TO warehouse\_clerk;

GRANT SELECT,USAGE ON SEQUENCE public.details\_detail\_id\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.details\_shelfid\_seq TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.employee TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.employee\_employee\_id\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.employee\_employee\_id\_seq TO warehouse\_manager;

GRANT SELECT ON TABLE public.invoice TO warehouse\_owner;

GRANT SELECT,UPDATE ON TABLE public.invoice TO warehouse\_clerk;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.invoice TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_counteragentid\_seq TO warehouse\_owner;

GRANT SELECT ON TABLE public.invoice\_detail TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.invoice\_detail TO warehouse\_manager;

GRANT SELECT ON TABLE public.invoice\_detail TO warehouse\_clerk;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_detail\_detailid\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_detail\_invoiceid\_seq TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_detail\_invoiceid\_seq TO warehouse\_owner;

GRANT SELECT ON TABLE public.invoice\_employee TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.invoice\_employee TO warehouse\_manager;

GRANT SELECT ON TABLE public.invoice\_employee TO warehouse\_clerk;

GRANT SELECT ON TABLE public.invoice\_details\_view TO warehouse\_owner;

GRANT SELECT,UPDATE ON TABLE public.invoice\_details\_view TO warehouse\_clerk;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.invoice\_details\_view TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_employee\_granted\_access\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_employee\_invoiceid\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_employee\_responsible\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_invoice\_id\_seq TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_invoice\_id\_seq TO warehouse\_owner;

GRANT SELECT,INSERT ON TABLE public.log\_table TO warehouse\_owner;

GRANT SELECT,INSERT ON TABLE public.log\_table TO warehouse\_clerk;

GRANT SELECT,INSERT ON TABLE public.log\_table TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.log\_table\_log\_id\_seq TO warehouse\_clerk;

GRANT SELECT,USAGE ON SEQUENCE public.log\_table\_log\_id\_seq TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.log\_table\_log\_id\_seq TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.rack TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.rack TO warehouse\_clerk;

GRANT SELECT,USAGE ON SEQUENCE public.rack\_rack\_id\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.rack\_rack\_id\_seq TO warehouse\_clerk;

GRANT SELECT,USAGE ON SEQUENCE public.rack\_roomid\_seq TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.room TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.room TO warehouse\_clerk;

GRANT SELECT,USAGE ON SEQUENCE public.room\_room\_id\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.room\_room\_id\_seq TO warehouse\_clerk;

GRANT SELECT,USAGE ON SEQUENCE public.room\_warehouseid\_seq TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.shelf TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.shelf TO warehouse\_clerk;

GRANT SELECT,USAGE ON SEQUENCE public.shelf\_rackid\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.shelf\_shelf\_id\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.shelf\_shelf\_id\_seq TO warehouse\_clerk;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.warehouse TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.warehouse TO warehouse\_clerk;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.warehouse\_details\_view TO warehouse\_owner;

GRANT SELECT,INSERT,UPDATE ON TABLE public.warehouse\_details\_view TO warehouse\_clerk;

GRANT SELECT ON TABLE public.warehouse\_details\_view TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.warehouse\_warehouse\_id\_seq TO warehouse\_owner;

ALTER DEFAULT PRIVILEGES FOR ROLE postgres IN SCHEMA public GRANT ALL ON FUNCTIONS TO warehouse\_owner;

ALTER DEFAULT PRIVILEGES FOR ROLE postgres IN SCHEMA public GRANT SELECT ON TABLES TO warehouse\_owner;warehouse\_owner;

**Приложение Е Отзыв руководителя**



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

**ОТЗЫВ**

**руководителя на курсовую работу**

Волковой Эмилии Юрьевны (Ф.И.О. студента)

Проектирование и реализация базы данных склада запчастей

(наименование темы КР)

представленный к защите по направлению/специальности

02.03.03 Математическое обеспечение и и

администрирование информационных систем

(код и наименование направления специальности подготовки)

Программное обеспечение вычислительной техник и

и автоматизированных систем (наименования профиля/специализация)

ст. преподаватель / Новиков С. П. /

(должность) (подпись) (Ф.И.О)

« » 20 г.