



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
код направления 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

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 «Вильямс», 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, предъявляемым конечными пользователями.

Руководитель проекта (работы)

подпись, дата

Новиков С.П.

ФИО

Задание принял(а) к исполнению

подпись, дата

Волкова Э.Ю.

ФИО

Содержание

Введение.....	5
1 Анализ предметной области и проектирования структуры базы данных.....	7
1.1 Формирование уровня представлений базы данных склада запчастей.....	7
1.1.1 Описание представлений пользователей и выделение основных сущностей.....	7
1.1.2 Описание объектов предметной области, их атрибутов и доменов	10
1.1.3 Создание схемы в виде ER-диаграммы	13
1.2.2 Нормализация отношений.....	20
1.2.2.1 Нормализация до первой нормальной формы (1НФ).....	20
1.2.2.2 Выбор первичных ключей и нормализация до второй нормальной формы (2НФ)	21
1.2.2.3 Нормализация до третьей нормальной формы (3НФ).....	22
1.2.3 Логическое (дatalogическое) проектирование базы данных	24
2 Физическая реализация модели БД склада запчастей.....	25
2.1 Обоснование выбора системы управления базами данных (СУБД)	25
2.2 Создание таблиц данных в среде целевой СУБД	26
3 Разработка клиентского интерфейса для работы с базой данных	33
3.1 Выбор среды для разработки приложения	33
3.2 Описание основных модулей и функций интерфейса	34
3.3 Пример работы графического интерфейса.....	36
Заключение	43
Приложение А ER-диаграмма.....	45
Приложение Б Datalogическая схема	46
Приложение В UML-диаграмма классов программного средства	47
Приложение Г Исходный код программного средства.....	48
Приложение Д Исходный код базы данных склада запчастей.....	101
Приложение Е Отзыв руководителя	119

					КР.350000.00						
Изм.	Лист	№ докум.	Подпись	Дата							
Разработал		Волкова Э.Ю.			Проектирование и реализация базы данных склада запчастей			Лит.	Лист	Листов	
Проверил		Новиков С.П.								4	119
								ДГТУ кафедра «ПОВТиАС»			
Утвердил		Долгов В.В.									

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.

					<i>KP.350000.000</i>	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

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

					КР.350000.000	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

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

					КР.350000.000	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

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».

					КР.350000.000	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

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);

shelfID – 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 (дatalogическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных дatalogическая 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-

					КР.350000.000	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

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

					КР.350000.000	Лист
						15
Изм.	Лист	№ докум.	Подпись	Дата		

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.

					КР.350000.000	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

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.

					<i>КР.350000.000</i>	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

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)

					КР.350000.000	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

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 (дatalogическое) 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

					<i>KP.350000.000</i>	Лист
						25
Изм.	Лист	№ докум.	Подпись	Дата		

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.

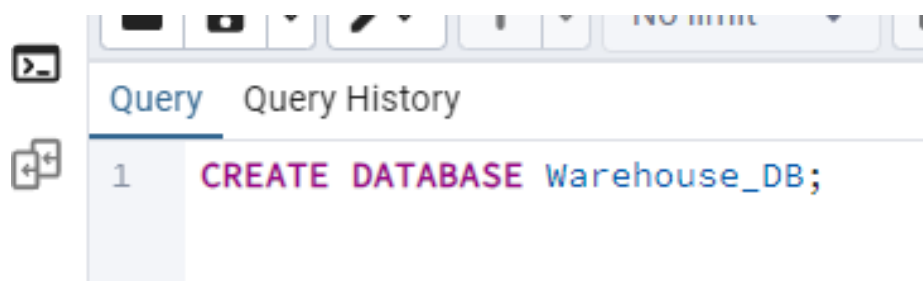


Рисунок 1 – Создание БД «Warehouse_DB»

```

Query Query History
1 CREATE TABLE invoice
2 (
3     invoice_id serial PRIMARY KEY,
4     counteragentID serial NOT NULL,
5     date_time timestamp NOT NULL,
6     type_invoice bool NOT NULL,
7     status bool NOT NULL,
8     FOREIGN KEY (counteragentID) REFERENCES counteragent (counteragent_id)
9 );

```

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.

```

CREATE TABLE IF NOT EXISTS public.log_table
(
    log_id integer NOT NULL DEFAULT nextval('log_table_log_id_seq'::regclass),
    table_name character varying(50) COLLATE pg_catalog."default" NOT NULL,
    action_type character varying(20) COLLATE pg_catalog."default" NOT NULL,
    record_id integer,
    action_time timestamp without time zone DEFAULT CURRENT_TIMESTAMP,
    old_values jsonb,
    new_values jsonb,
    CONSTRAINT log_table_pkey PRIMARY KEY (log_id)
)

```

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.

```

CREATE OR REPLACE FUNCTION public.log_warehouse_changes()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
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;
$BODY$;

```

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.

```

CREATE OR REPLACE FUNCTION public.delete_related_data()
RETURNS trigger
LANGUAGE 'plpgsql'
COST 100
VOLATILE NOT LEAKPROOF SECURITY DEFINER
AS $BODY$
BEGIN
-- Для склада удаляем связанные комнаты, стеллажи, полки и детали
IF TG_TABLE_NAME = 'warehouse' THEN
-- Удаляем детали через отдельный запрос с явными правами
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;

-- Для комнаты удаляем связанные стеллажи, полки и детали
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;

-- Для стеллажа удаляем связанные полки и детали
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;

-- Для полки удаляем связанные детали
ELSIF TG_TABLE_NAME = 'shelf' THEN
PERFORM * FROM details WHERE shelfID = OLD.shelf_id LIMIT 1;
END IF;
RETURN OLD;
END;
$BODY$;

```

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.

					<i>KP.350000.000</i>	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

```

CREATE OR REPLACE VIEW public.invoice_details_view
AS
SELECT inv.invoice_id,
       ca.counteragent_name,
       inv.date_time,
       CASE
         WHEN inv.type_invoice THEN 'Выпуска'::text
         ELSE 'Отпуска'::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 invoice inv
JOIN invoice_detail invd ON inv.invoice_id = invd.invoiceid
JOIN details det ON invd.detailid = det.detail_id
JOIN invoice_employee inv_emp ON inv.invoice_id = inv_emp.invoiceid
JOIN employee emp ON inv_emp.responsible = emp.employee_id
JOIN counteragent ca ON inv.counteragentid = ca.counteragent_id;

```

Рисунок 6 – Представление «invoice_details_view»

```

CREATE OR REPLACE 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 warehouse w
JOIN room r ON w.warehouse_id = r.warehouseid
JOIN rack rk ON r.room_id = rk.roomid
JOIN shelf s ON rk.rack_id = s.rackid
JOIN details d ON s.shelf_id = d.shelfid;

```

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:

					<i>KP.350000.000</i>	Лист
						30
Изм.	Лист	№ докум.	Подпись	Дата		

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.

```
CREATE OR REPLACE FUNCTION public.convert_text_to_boolean(
    text_value text,
    field_type text DEFAULT 'status'::text)
    RETURNS boolean
    LANGUAGE 'plpgsql'
    COST 100
    IMMUTABLE PARALLEL UNSAFE
AS $BODY$
BEGIN
    text_value := LOWER(TRIM(text_value));

    -- Для типа накладной
    IF field_type = 'type' THEN
        RETURN text_value IN ('выгрузка', 'выгрузить', 'отправка', 'true', '1', 'да', 'yes', 'y');
    -- Для статуса
    ELSE
        RETURN text_value IN ('завершено', 'готово', 'выполнено', 'done', 'true', '1', 'да', 'yes', 'y');
    END IF;
END;
$BODY$;
```

Рисунок 8 – Функция convert_text_to_boolean()

```
CREATE OR REPLACE FUNCTION public.get_employee_id(
    p_last_name character varying,
    p_first_name character varying,
    p_patronymic character varying)
    RETURNS integer
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE SECURITY DEFINER PARALLEL UNSAFE
AS $BODY$
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;
$BODY$;
```

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.

					<i>KP.350000.000</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		32

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.

3.1 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

					КР.350000.000	Лист
						33
Изм.	Лист	№ докум.	Подпись	Дата		

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.

3.2 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.).

					<i>KP.350000.000</i>	Лист
						35
Изм.	Лист	№ докум.	Подпись	Дата		

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 В

Исходный код приложения представлен в Приложении Г.

3.3 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.

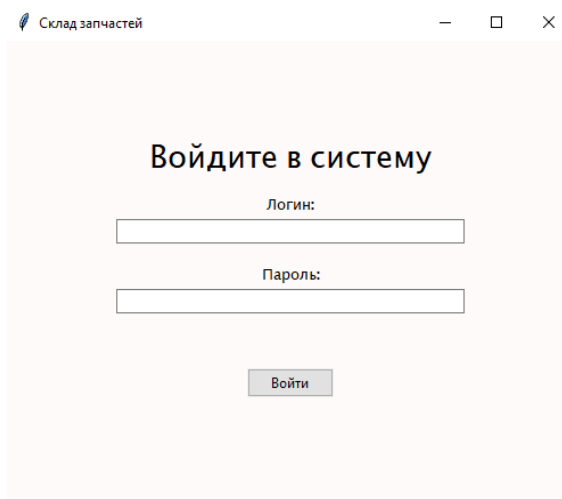


Рисунок 10 – Окно авторизации

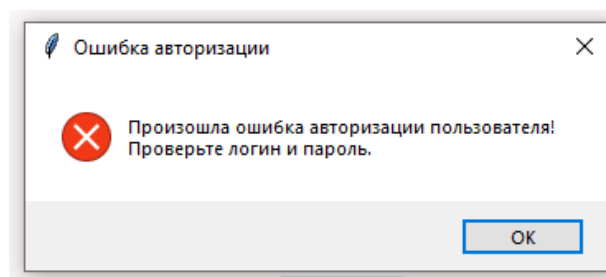


Рисунок 11 – Ошибка авторизации пользователя

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

ID	Склад	Комната	Стеллаж	Полка	Деталь	Вес
1	101	12	122	1221	Двигатель	12.0
2	101	11	111	1112	Тормозные колодки	5.0
3	101	11	111	1113	Подвеска	20.7
4	101	11	111	1114	Фары	7.3
5	101	11	111	1115	Шины	15.2
6	101	11	113	1131	Двигатель	12.2
7	101	11	114	1143	Тормозные колодки	5.0
8	101	11	112	1122	Подвеска	20.7
9	101	11	114	1144	Фары	7.3
10	101	11	115	1151	Шины	15.2
11	101	11	115	1151	Двигатель	12.5
12	101	11	115	1152	Тормозные колодки	5.0
13	101	11	115	1153	Подвеска	20.7
14	101	11	115	1154	Фары	7.3
15	101	11	115	1155	Шины	15.2
16	101	11	112	1123	Коробка передач	12.5
17	101	11	112	1124	Карданный вал	5.0
18	101	11	112	1125	Радиатор	20.7
19	101	11	113	1131	Генератор	7.3
20	101	11	113	1132	Стартер	15.2
21	101	11	113	1133	Поршень	12.5
26	101	12	124	1245	Выхлопная система	12.5
39	101	11	111	1111	Полуось	1.0

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».

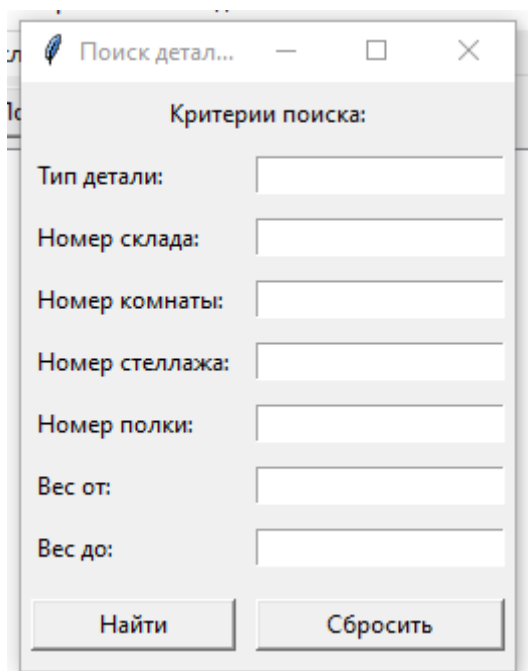


Рисунок 13 – Поиск детали в таблице «Склад»

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.

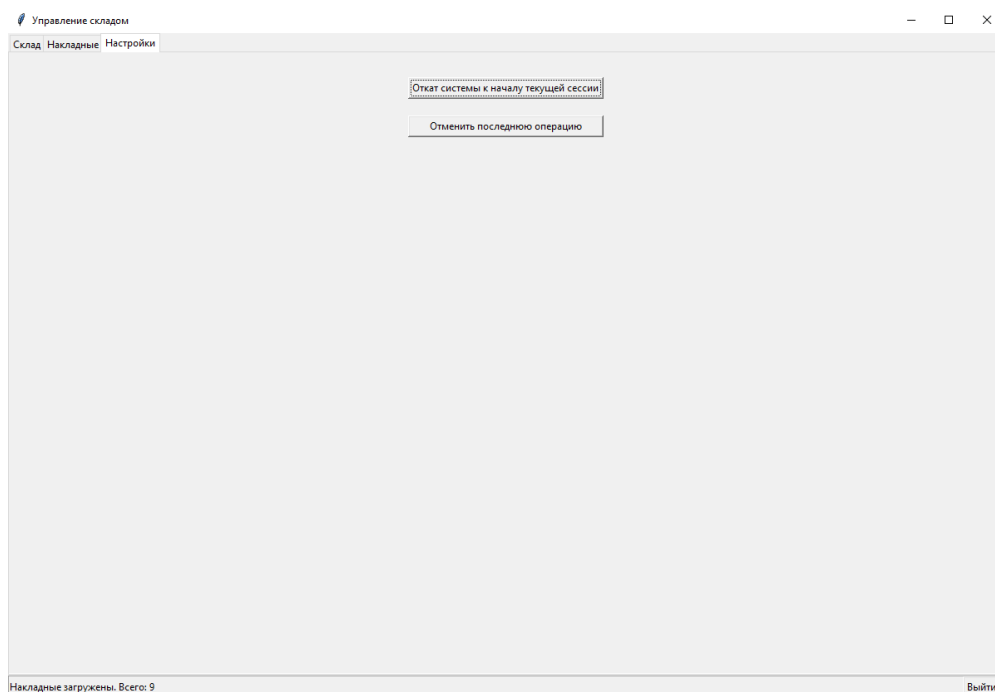


Рисунок 14 – Кнопки отката системы

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

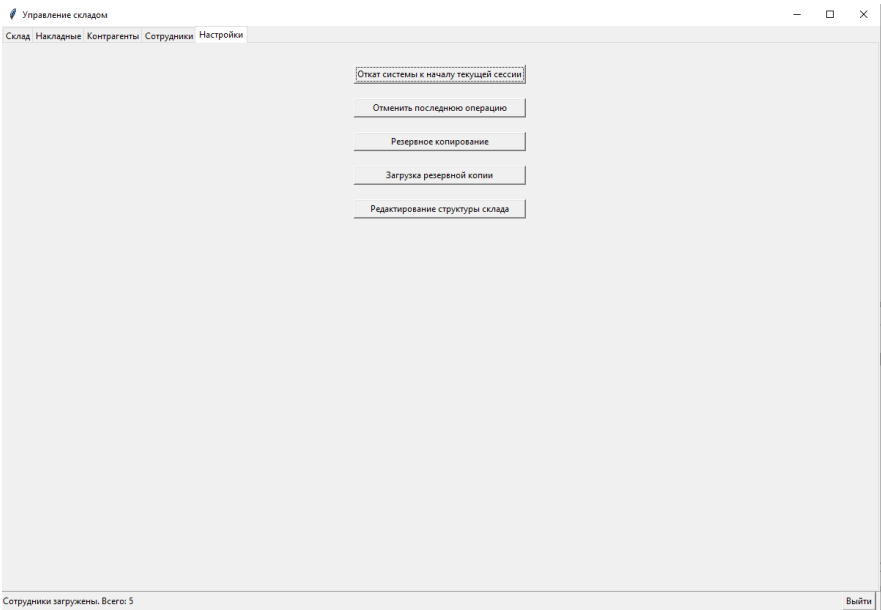


Рисунок 15 – Вкладка «Настройки» для пользователя owner

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.

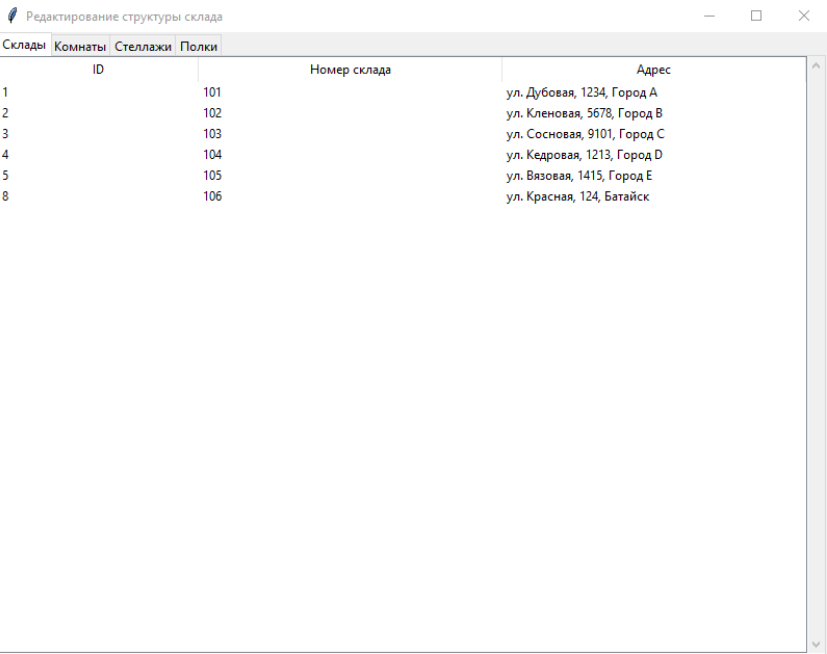


Рисунок 16 – Окно редактирования структуры склада

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.

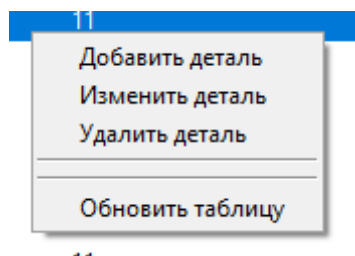


Рисунок 17 – Список доступных действий

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

 A screenshot of a form titled 'Добавить детал...' (Add detail...). The form has a title bar with a feather icon, a minus button, a maximize button, and a close button. It contains several input fields: 'Номер склада:' (Warehouse number) with a dropdown menu showing '101'; 'Номер комнаты:' (Room number) with a dropdown menu showing '11'; 'Номер стеллажа:' (Shelf number) with a dropdown menu showing '111'; 'Номер полки:' (Shelf number) with a dropdown menu showing '1111'; 'Тип детали:' (Detail type) with an empty text field; and 'Вес (кг):' (Weight in kg) with a text field showing '0.0'. At the bottom right of the form is a 'Сохранить' (Save) button.

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.

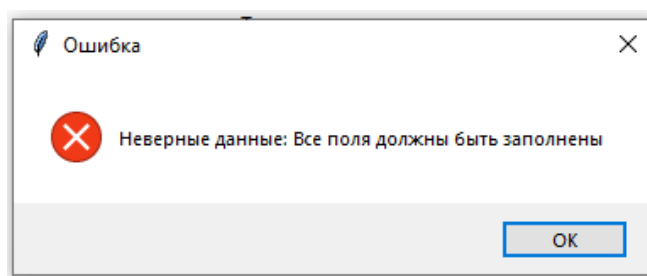


Рисунок 19 – Ошибка заполнения полей

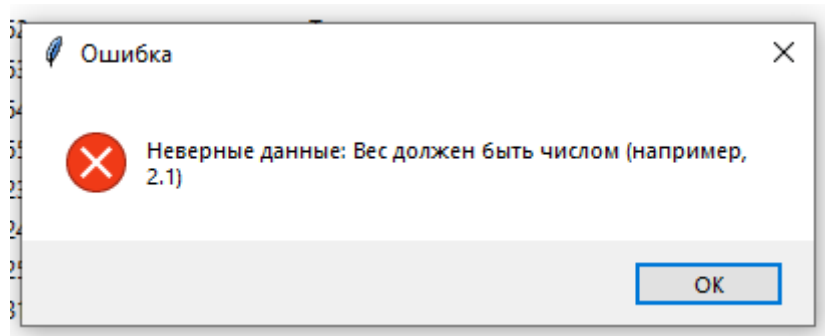


Рисунок 20 – Неверно введенные данные

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

Если пользователь нажмет на кнопку «Изменить», то на экране появится 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 появится уведомление, что изображено на рисунке 22.

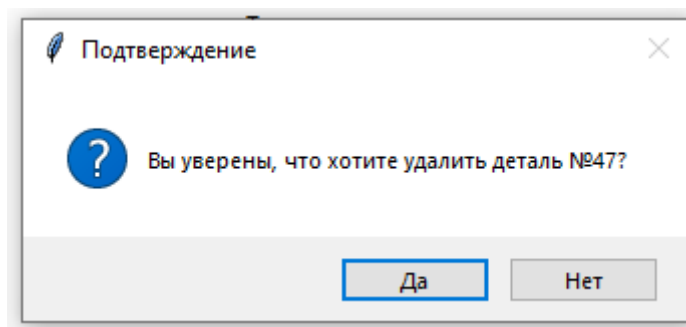


Рисунок 22 – Подтверждение удаления записи

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

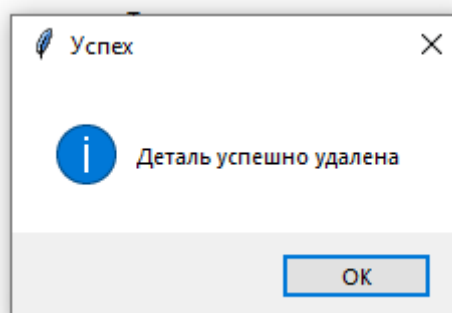


Рисунок 23 – Уведомление об успешном удалении записи

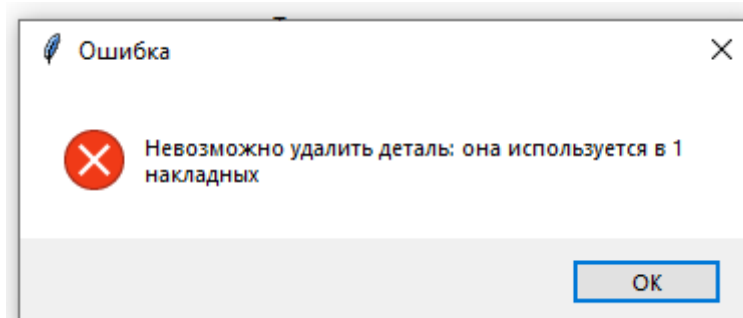


Рисунок 24 – Уведомление об ошибке удаления

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.

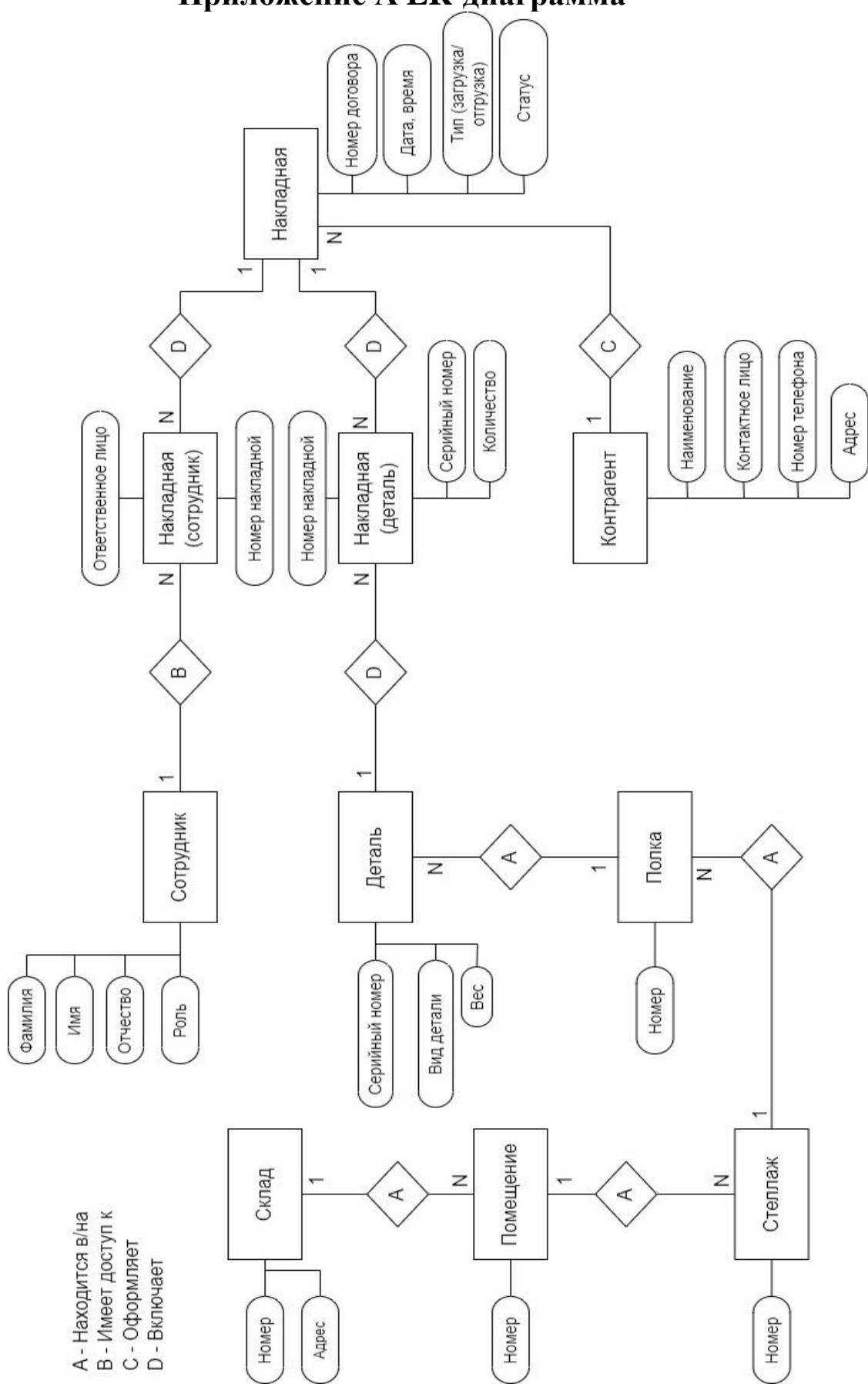
					<i>КР.350000.000</i>	Лист
						43
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень используемых информационных источников

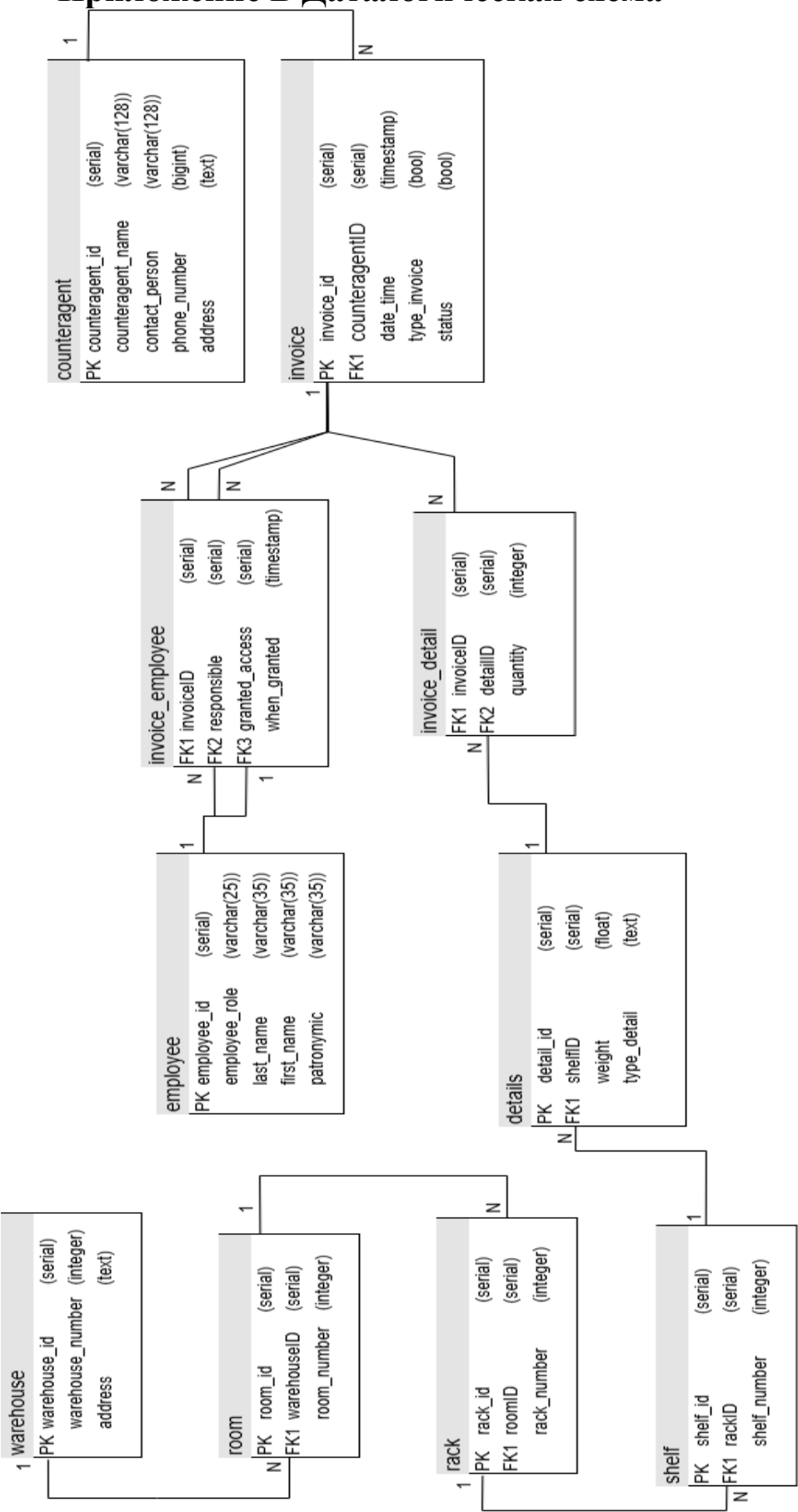
1. Дейт, Кристофер – Введение в системы баз данных, 8-е издание / Пер. с англ. – М.: Издательский дом «Вильямс», 2005 – 1328 с.: ил. – Парал. тит. англ. – ISBN 5-8459-0788-8
2. Паттон Джефф. Пользовательские истории. Искусство гибкой разработки ПО. – СПб.: Питер, 2019. – 288 с.
3. Карпова Т.С. Базы данных: 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

					КР.350000.000	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

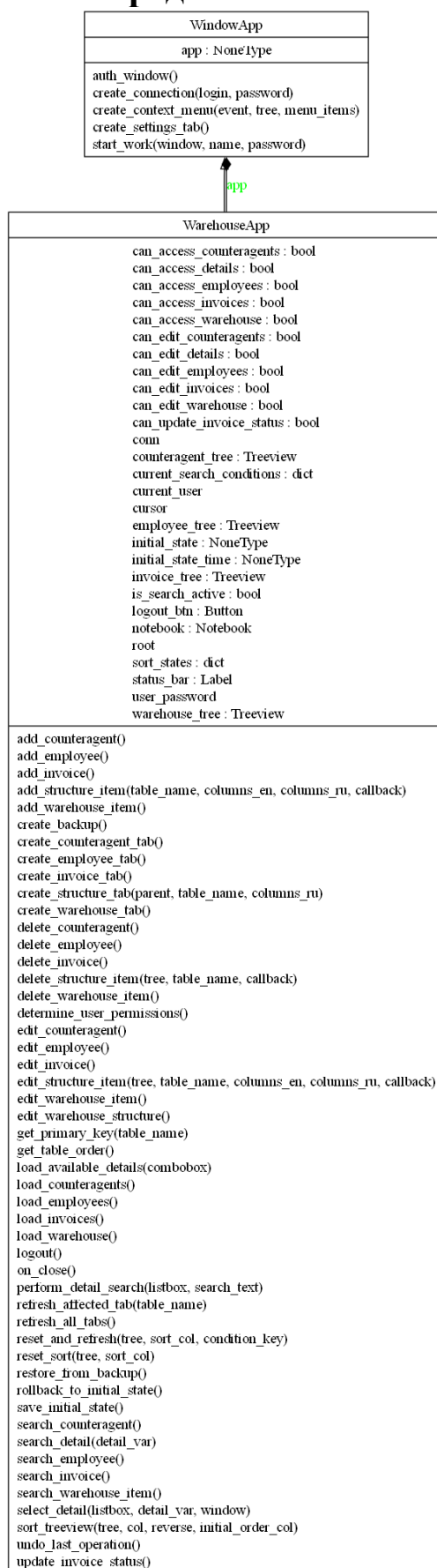
Приложение А ER-диаграмма



Приложение Б Даталогическая схема



Приложение В UML-диаграмма классов программного средства



Приложение Г Исходный код программного средства

Листинг 1 – Исходный код программного средства

```
import psycpg2
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.wininfo_screenwidth()/2) -
            (500/2),
            (window.wininfo_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 =
            psycpg2.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:
```

					КР.350000.000	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		


```

        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.upda
te({'invoice': None}),
self.app.load_invoices()])
    elif tree ==
self.warehouse_tree:

menu.add_command(label="Обновить
таблицу", command=lambda:
[self.current_search_conditions.upda
te({'warehouse': None}),
self.app.load_warehouse()])
    elif tree ==
self.counteragent_tree:

menu.add_command(label="Обновить
таблицу", command=lambda:
[self.current_search_conditions.upda
te({'counteragent': None}),
self.app.load_counteragents()])
    elif tree ==
self.employee_tree:

menu.add_command(label="Обновить
таблицу", command=lambda:
[self.current_search_conditions.upda

```

```

te({'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_sta
te)
    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_structur
e)

        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_li
stbox, 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_li
stbox, "")

            search_entry.bind("<Return>", lambda
e:
self.perform_detail_search(detail_li
stbox, 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("Предупрежден
ие", "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.updat
e({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("Предупрежден
ие", "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}
        WHEN
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 откатена 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
]

```

					КР.350000.000	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

```

        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(t
able_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.upda
te({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_WIND
OW", 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("Предупрежден
ие", "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("Предупрежден
ие", "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("Ответствен
ный", width=150)
```

```
    scroll = ttk.Scrollbar(tab,
command=self.invoice_tree.yview)
    scroll.pack(side=RIGHT,
fill=Y)
```

```
self.invoice_tree.configure(yscrollc
ommand=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("Предупрежде
ие", "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
e_tree, "ID", 'warehouse'))
            refresh_btn.pack(side=LEFT,
padx=2)

            columns = ("ID", "Склад",
"Комната", "Стеллаж", "Полка",
"Деталь", "Бес")
            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("Бес",
width=80)

            scroll = ttk.Scrollbar(tab,
command=self.warehouse_tree.yview)
            scroll.pack(side=RIGHT,
fill=Y)

self.warehouse_tree.configure(yscroll
lcommand=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.countera
gent_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("Назв
ание", text="Название")

self.counteragent_tree.heading("Конт
акт", text="Контактное лицо")

self.counteragent_tree.heading("Теле
фон", text="Телефон")

self.counteragent_tree.heading("Адре
с", text="Адрес")

self.counteragent_tree.column("ID",
width=50)

self.counteragent_tree.column("Назва
ние", width=200)

```

```

self.counteragent_tree.column("Конта
кт", width=150)

```

```

self.counteragent_tree.column("Телеф
он", 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.pack(fill=BOT
H, expand=True)

```

```

self.counteragent_tree.configure(ysc
rollcommand=scroll.set)

```

```

self.counteragent_tree.pack(fill=BOT
H, 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(yscroll
command=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.invoi
ce_tree.get_children())

            if
self.current_search_conditions['invo
ice']:

                conditions =
self.current_search_conditions['invo
ice']['conditions']

                params =
self.current_search_conditions['invo
ice']['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['invo
ice']:

self.status_bar.config(text=f"Найден
о накладных: {count}")

```

```

                                else:

self.status_bar.config(text=f"Наклад
ные 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.g
et()}%")

```

```

        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['invo
ice'] = {
            '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['invo
ice'] = 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.war
ehouse_tree.get_children())

        if
self.current_search_conditions['ware
house']:
            conditions =
self.current_search_conditions['ware
house']['conditions']
            params =
self.current_search_conditions['ware
house']['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,
                    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,
                    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['ware
house']:

self.status_bar.config(text=f"Найден
о 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['ware
house'] = {
            '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['ware
house'] = 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['coun
teragent']:
            conditions =
self.current_search_conditions['coun
teragent']['conditions']
            params =
self.current_search_conditions['coun
teragent']['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_child
ren())

```

```

        if
self.current_search_conditions['coun
teragent']:

```

```

self.status_bar.config(text=f"Найден
о контрагентов: {count}")
else:

```

```

self.status_bar.config(text=f"Контра
генты 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::tex
t 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['coun
teragent'] = {
            '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['coun
teragent'] = 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.empl
oyee_tree.get_children())

                if
self.current_search_conditions['empl
oyee']:

                    conditions =
self.current_search_conditions['empl
oyee']['conditions']
                    params =
self.current_search_conditions['empl
oyee']['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['empl
oyee']:

self.status_bar.config(text=f"Найден
о сотрудников: {count}")
                else:

self.status_bar.config(text=f"Сотруд
ники 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['empl
oyee'] = {
                    '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['empl
oyee'] = 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
self.can_edit_invoices:
                    not

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("Предупрежден
ие", "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_va
r.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("Предупрежден
ие", "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:

```

					КР.350000.000	Лист
Изм.	Лист	№ докум.	Подпись	Дата		84

```

            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("Предупрежден
ие", "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

```

					<div style="text-align: right; font-size: 1.2em; font-weight: bold;"> <i>KP.350000.000</i> </div>	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

```

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['invo
ice']:

messagebox.showinfo("Информация",
"Таблица пуста. Сброс условий
поиска.")

self.current_search_conditions['invo
ice'] = 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("""
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("""
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("<<ComboboxS
                                                elected>>", update_rooms)
                                                room_combobox.bind("<<ComboboxSelect
                                                ed>>", update_Racks)
                                                rack_combobox.bind("<<ComboboxSelect
                                                ed>>", 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("""

```



```

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("<<ComboboxS
elected>>", update_rooms)

room_combobox.bind("<<ComboboxSelect
ed>>", update_racks)

rack_combobox.bind("<<ComboboxSelect
ed>>", update_shelves)

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 = (

```



```

        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['ware
house']:

messagebox.showinfo("Информация",
"Таблица пуста. Сброс условий
поиска.")

self.current_search_conditions['ware
house'] = 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
self.can_edit_counteragents:
                not
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("Предупрежден
ие", "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("Предупрежден
ие", "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_child
ren()) == 0 and
self.current_search_conditions['coun
teragent']:

messagebox.showinfo("Информация",
"Таблица пуста. Сброс условий
поиска.")

self.current_search_conditions['coun
teragent'] = 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("Предупрежден
ие", "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 $$
BEGIN
IF TG_TABLE_NAME = 'warehouse'
THEN
PERFORM * FROM details WHERE
shelfID IN (
SELECT shelf_id FROM
shelf WHERE rackID IN (
SELECT rack_id FROM
rack WHERE roomID IN (
SELECT room_id
FROM room WHERE warehouseID =
OLD.warehouse_id
)
) LIMIT 1;
DELETE FROM shelf WHERE
rackID IN (
SELECT rack_id FROM rack
WHERE roomID IN (
SELECT room_id FROM
room WHERE warehouseID =
OLD.warehouse_id
)
);
DELETE FROM rack WHERE
roomID IN (
SELECT room_id FROM room
WHERE warehouseID = OLD.warehouse_id
);
DELETE FROM room WHERE
warehouseID = OLD.warehouse_id;
ELSIF TG_TABLE_NAME = 'room'
THEN
PERFORM * FROM details WHERE
shelfID IN (
SELECT shelf_id FROM
shelf WHERE rackID IN (
DELETE FROM invoice_detail WHERE
invoiceID = OLD.invoice_id;
DELETE FROM invoice WHERE
invoice_id = OLD.invoice_id;
RETURN OLD;
END;
$$;

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 (
DELETE FROM invoice_employee
WHERE invoiceID = OLD.invoice_id;
```

```

        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_inv
oice_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_t
ext, '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_inv
            oice_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_t
            ext, '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_acce
ss_seq
AS integer
START WITH 1
```

```

INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1;

ALTER                                SEQUENCE
public.invoice_employee_granted_acce
ss_seq OWNER TO postgres;
ALTER                                SEQUENCE
public.invoice_employee_granted_acce
ss_seq                               OWNED      BY
public.invoice_employee.granted_acce
ss;

CREATE                                SEQUENCE
public.invoice_employee_invoiceid_se
q
AS integer
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1;

ALTER                                SEQUENCE
public.invoice_employee_invoiceid_se
q OWNER TO postgres;
ALTER                                SEQUENCE
public.invoice_employee_invoiceid_se
q                               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_counter
agent_id_seq'::regclass);
ALTER TABLE ONLY public.details
ALTER COLUMN detail_id SET DEFAULT
nextval('public.details_detail_id_se
q'::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_s
eq'::regclass);
ALTER TABLE ONLY public.invoice
ALTER COLUMN counteragentid SET
DEFAULT
nextval('public.invoice_counteragent
id_seq'::regclass);
ALTER TABLE ONLY
public.invoice_detail ALTER COLUMN
invoiceid SET DEFAULT
nextval('public.invoice_detail_invoi
ceid_seq'::regclass);
ALTER TABLE ONLY
public.invoice_detail ALTER COLUMN
detailid SET DEFAULT
nextval('public.invoice_detail_detai
lid_seq'::regclass);
ALTER TABLE ONLY
public.invoice_employee ALTER COLUMN
invoiceid SET DEFAULT
nextval('public.invoice_employee_inv
oiceid_seq'::regclass);
```

```
ALTER TABLE ONLY
public.invoice_employee ALTER COLUMN
responsible SET DEFAULT
nextval('public.invoice_employee_res
ponsible_seq'::regclass);
ALTER TABLE ONLY
public.invoice_employee ALTER COLUMN
granted_access SET DEFAULT
nextval('public.invoice_employee_gra
nted_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'::r
egclass);
ALTER TABLE ONLY public.rack ALTER
COLUMN roomid SET DEFAULT
nextval('public.rack_roomid_seq'::re
gclass);
ALTER TABLE ONLY public.room ALTER
COLUMN room_id SET DEFAULT
nextval('public.room_room_id_seq'::r
egclass);
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'::r
egclass);
ALTER TABLE ONLY public.warehouse
ALTER COLUMN warehouse_id SET
DEFAULT
nextval('public.warehouse_warehouse_
id_seq'::regclass);
```

```
SELECT
pg_catalog.setval('public.counterage
nt_counteragent_id_seq', 6, true);
SELECT
pg_catalog.setval('public.details_de
tail_id_seq', 47, true);
SELECT
pg_catalog.setval('public.details_sh
elfid_seq', 1, false);
SELECT
pg_catalog.setval('public.employee_e
mployee_id_seq', 6, true);
SELECT
pg_catalog.setval('public.invoice_co
unteragentid_seq', 1, false);
SELECT
pg_catalog.setval('public.invoice_de
tail_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_room_id_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_rack_id_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_acce
ss_seq TO warehouse_owner;
GRANT SELECT,USAGE ON SEQUENCE
public.invoice_employee_invoiceid_se
q 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 Эмилиии 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

администрирование информационных систем

(код и наименование направления специальности подготовки)

Программное обеспечение вычислительной техник

и автоматизированных систем

(наименования профиля/специализация)

ст. преподаватель
(должность)

_____/ Новиков С. П. /
(подпись) (Ф.И.О)
«___» _____ 20 г.

					КР.350000.000	Лист
						119
Изм.	Лист	№ докум.	Подпись	Дата		