

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Зав. кафедрой \_\_ПОВТ и АС

\_\_\_\_\_\_\_\_\_\_\_\_ Долгов В.В.

подпись ФИО

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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту по дисциплине Базы данных

на тему Проектирование и реализация базы данных склада запчастей

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

подпись Ф.И.О.

Направление/специальность, профиль/специализация:

02.03.03 Математическое обеспечение и администрирование информационных систем

коднаправления наименование профиля (специализации)

Обозначение курсового проекта (работы) КР.350000.000 Группа ВМО31

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

Проект (работа) защищен (а)

дата оценка подпись

г. Ростов-на-Дону

2024 г.



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Зав. кафедрой ПОВТ и АС

\_\_\_\_\_\_\_\_\_\_\_\_ Долгов В.В.

(подпись) (ФИО.)

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

**ЗАДАНИЕ**

к курсовой работе по дисциплине Базы данных

Студент Волкова Э.Ю. КР.350000.000 Группа ВМО31

Тема Проектирование и реализация базы данных склада запчастей выв

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

Исходные данные для курсового проекта (работы)

Задание на выполнение курсовой работы

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

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

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

Содержание пояснительной записки

ВВЕДЕНИЕ:

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

Наименование и содержание разделов:

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

ЗАКЛЮЧЕНИЕ:

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

Руководитель проекта (работы) Новиков С.П.

подпись, дата ФИО

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

подпись, дата ФИО

**Содержание**

[Введение 5](#_Toc193250564)

[1 Анализ предметной области и проектирования структуры базы данных 7](#_Toc193250565)

[1.1 Формирование уровня представлений базы данных склада запчастей 7](#_Toc193250566)

[1.1.1 Описание представлений пользователей и выделение основных сущностей 7](#_Toc193250567)

[1.1.2 Описание объектов предметной области, их атрибутов и доменов 10](#_Toc193250568)

[1.1.3 Создание схемы в виде ER-диаграммы 13](#_Toc193250569)

[1.2.2 Нормализация отношений 20](#_Toc193250570)

[1.2.2.1 Нормализация до первой нормальной формы (1НФ) 20](#_Toc193250571)

[1.2.2.2 Выбор первичных ключей и нормализация до второй нормальной формы (2НФ) 21](#_Toc193250572)

[1.2.2.3 Нормализация до третьей нормальной формы (3НФ) 23](#_Toc193250573)

[1.2.3 Логическое (даталогическое) проектирование базы данных 24](#_Toc193250574)

[2 Физическая реализация модели БД склада запчастей 26](#_Toc193250575)

[2.1 Обоснование выбора системы управления базами данных (СУБД) 26](#_Toc193250576)

[2.2 Создание таблиц данных в среде целевой СУБД 27](#_Toc193250577)

[3 Разработка клиентского интерфейса для работы с базой данных 34](#_Toc193250578)

[3.1 Выбор среды для разработки приложения 34](#_Toc193250579)

[3.2 Описание основных модулей и функций интерфейса 35](#_Toc193250580)

[3.3 Пример работы графического интерфейса 36](#_Toc193250581)

[Заключение 42](#_Toc193250582)

[Приложение А ER-диаграмма 44](#_Toc193250583)

[Приложение Б Даталогическая схема 45](#_Toc193250584)

[Приложение В Исходный код программного средства 46](#_Toc193250585)

[Приложение Г Исходный код базы данных склада запчастей 72](#_Toc193250586)

# **Введение**

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

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

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

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

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

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

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

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

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

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

## 1.1 Формирование уровня представлений базы данных склада запчастей

### 1.1.1 Описание представлений пользователей и выделение основных сущностей

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

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

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

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

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

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

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

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

На основе предметной области можно выделить следующих

пользователей: кладовщик, менеджер склада, владелец склада.

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

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

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

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

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

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

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

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

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

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

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

Ограничения:

warehouse\_id – первичный ключ (уникальные значения).

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

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

Ограничения:

room\_id – первичный ключ (уникальные значения);

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

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

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

Ограничения:

rack\_id – первичный ключ (уникальные значения);

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

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

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

Ограничения:

shelf\_id – первичный ключ (уникальные значения);

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

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

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

Ограничения:

detail\_id – первичный ключ (уникальные значения);

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

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

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

Ограничения:

counteragent\_id – первичный ключ (уникальные значения).

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

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

Ограничения:

counteragent\_id – первичный ключ (уникальные значения);

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

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

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

Ограничения:

invoice\_id – первичный ключ (уникальные значения);

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

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

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

Ограничения:

employee\_id – первичный ключ (уникальные значения).

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

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

Ограничения:

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

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

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

### 1.1.3 Создание схемы в виде ER-диаграммы

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

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

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

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

Рассмотрим основные понятия, используемые для построения ER-диаграммы.

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

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

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

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

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

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

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

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

Рассмотрим атрибуты каждой сущности, необходимые конечным пользователям для работы.

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

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

«Склад»: номер, адрес.

«Помещение»: номер.

«Стеллаж»: номер.

«Полка»: номер.

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

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

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

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

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

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

**1.1.4 Описание связей между объектами**

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

**1.2 Построение концептуального уровня базы данных**

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

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

{warehouse\_number} -> {address}

Ограничение: по одному адресу не могут стоять несколько складов.

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

Таблица warehouse имеет в качестве первичного ключа warehouse\_id.

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

{warehouseID} -> {room\_number}

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}

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}

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.

**details** = {detail\_id, shelfID, weight, type\_detail}

{type\_detail} -> {weight}

Каждая деталь идентифицируется по уникальному detail\_id, первичному ключу, а также по атрибуту shelfID, который является внешним ключом, ссылающийся на таблицу shelf. Атрибут weight функционально зависит от атрибута type\_detail и обозначает вид детали.

**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}

{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.

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

{employee\_id} -> {employee\_role, last\_name, first\_name, patronymic}

Каждый сотрудник идентифицируется по уникальному employee\_id, первичному ключу. Атрибуты employee\_role, last\_name, first\_name и patronymic функционально зависят от него и обозначают информацию о сотруднике: его роль, фамилия, имя и отчество.

**invoice\_employee** = {invoiceID, responsible, granted\_access, when\_granted}

{responsible} -> {invoiceID}

Каждая накладная идентифицируется по уникальному invoiceID, внешнему ключу, который ссылается на таблицу invoice. Атрибут invoiceID функционально зависит от responsible и обозначает ссылку на запись в таблицу employee, которая обозначает сотрудника ответственного за эту накладную.

### 1.2.2 Нормализация отношений

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

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

Рассмотрим базовые отношения базы данных «Склад запчастей».

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

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

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

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

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

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

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

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

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

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

Все базовые отношения находятся в 1НФ.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Все атрибуты зависят от room\_id.

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

Все атрибуты зависят от rack\_id.

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

Все атрибуты зависят от shelf\_id.

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

Все атрибуты зависят от detail\_id.

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

Все атрибуты зависят от counteragent\_id.

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

Все атрибуты зависят от invoice\_id.

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

Все атрибуты зависят от employee\_id.

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

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

Все атрибуты зависят от составного ключа.

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

Все атрибуты зависят от составного ключа.

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

### 1.2.3 Логическое (даталогическое) проектирование базы данных

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

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

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

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

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

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

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

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

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

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

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

## 2.2 Создание таблиц данных в среде целевой СУБД

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

В среде администрирования и разработки PostgreSQL с помощью SQL-запросов была создана база данных «Warehouse\_DB», что показано на рисунке 1.

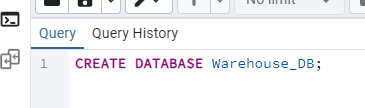


Рисунок 1 – Создание БД «Warehouse\_DB»

А также все необходимые нам таблицы: «warehouse», «room», «rack» и т.д. Создание таблиц представлено на рисунке 2 на примере таблицы «invoice».

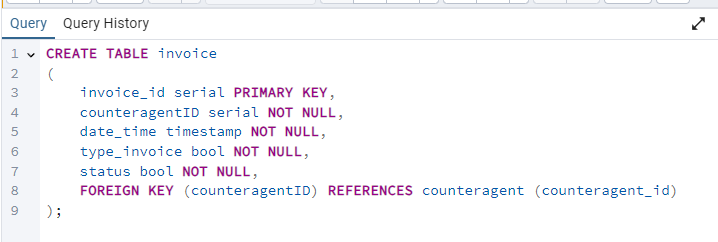


Рисунок 2 – Создание таблицы «invoice»

В данной базе данных находится таблица для реализации темпоральности: «log\_table». Данная таблица состоит из следующих полей:

* log\_id – уникальный идентификатор записей в таблице «log\_table»;
* table\_name – название таблицы, в которой были выполнены изменения;
* action\_type – тип изменения, который был выполнен в таблице (INSERT, UPDATE, DELETE);
* record\_id – уникальный идентификатор записи в исходной таблице, к которой относится изменение;
* action\_time – время, в которое было выполнены изменения;
* old\_values – старые значения измененной записи в таблице;
* new\_values – новые значения измененной записи в таблице.

Создание данной таблицы показано на рисунке 3.

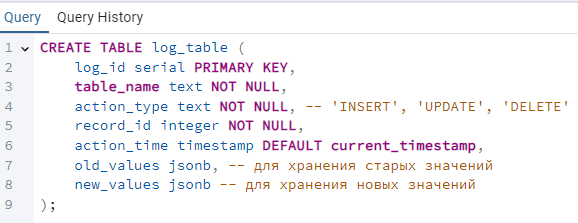


Рисунок 3 – Таблица «log\_table»

Для данной таблицы также были созданы триггеры и функции, позволяющие фиксировать изменения записей в базе данных.

Для таблицы warehouse была создана функция log\_warehouse\_changes(), которая записывает информацию об операциях INSERT, UPDATE и DELETE, выполненных в этой таблице. К данной функции был написан триггер warehouse\_changes, что срабатывает после операций INSERT, UPDATE или DELETE в соответствующей таблице.   
 Таким же образом были созданы функции и триггеры для оставшихся таблиц, а именно **«**room», **«**rack», **«**shelf», **«**invoice», **«**invoice\_employee», **«**invoice\_detail», **«**detail», **«**counteragents» и **«**employee». Данные функции показаны на примере функции log\_warehouse\_changes() и триггера warehouse\_changes, что изображены на рисунках 4 и 5.



Рисунок 4 – Функция log\_warehouse\_changes()

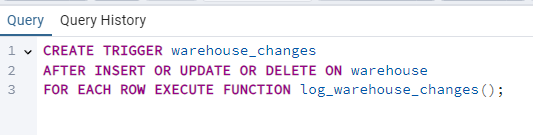


Рисунок 5 – Триггер warehouse\_changes

Для соблюдения актуальности данных в базе данных склада запчастей также разработаны и внедрены триггеры и функции, необходимые для выполнения этой задачи.

Для каскадного удаления связанных данных склада создана функция delete\_related\_data(), которая удаляет все связанные данные при удалении склада: комнаты, принадлежащие складу; стеллажи в этих комнатах, полки на этих стеллажах, детали на этих полках. Также создан триггер trg\_delete\_related\_data, что срабатывает перед удалением записи в таблице warehouse и выполняет функцию delete\_related\_data(). Данные функция и триггер показаны на рисунке 6.

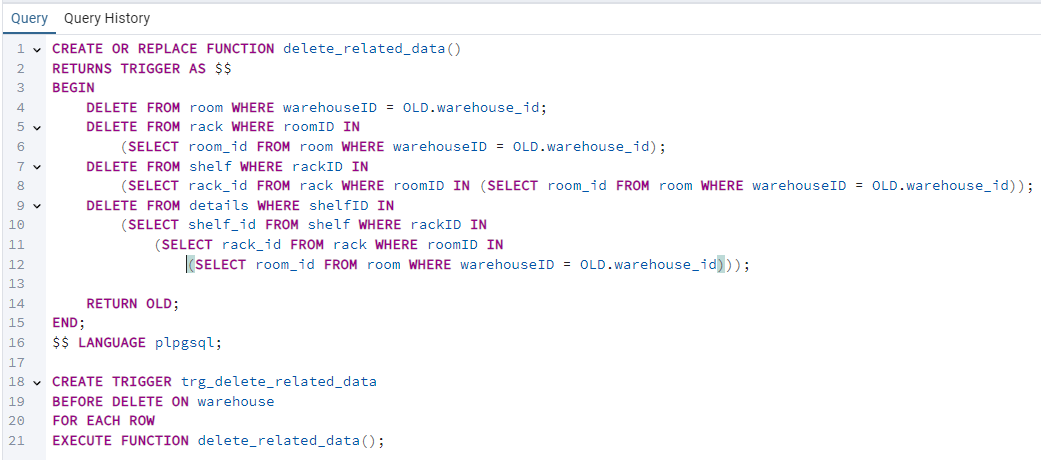


Рисунок 6 – Функция delete\_related\_data() и триггер trg\_delete\_related\_data

В среде администрирования и разработки PostgreSQL с помощью SQL-запросов было создано представление «invoice\_details\_view», что показано на рисунке 7:



Рисунок 7 – Создание представления «invoice\_details\_view»

Данное представление позволяет в полном объёме получить информацию по накладной: какому контрагенту принадлежит, когда создана, какой тип и статус у накладной, тип детали, количество этой детали, а также фамилия, имя и отчество ответственного за эту накладную кладовщика. Для поддержания актуальности информации в таблицах **«**invoice», **«**invoice\_detail», **«**details», **«**invoice\_employee», **«**employee» и **«**counteragent» были созданы следующие функции:

* insert\_invoice\_details\_view() – эта функция вставляет данные в связанные таблицы при добавлении через представление. Создаёт новую запись в «invoice», добавляет детали в «invoice\_detail» и добавляет информацию о сотрудниках в «invoice\_employee». Данной функции соответствует триггер trg\_insert\_invoice\_details\_view, срабатывающий при вставке в представление «invoice\_details\_view».
* delete\_invoice\_details\_view() – эта функция удаляет связанные данные при удалении через представление. Удаляет записи из «invoice\_employee», «invoice\_detail» и «invoice». Данной функции соответствует триггер invoice\_details\_view\_delete, срабатывающий при удалении из представления «invoice\_details\_view».

Для удобной работы с деталями было создано представление «warehouse\_details\_view», что показано на рисунке 8.

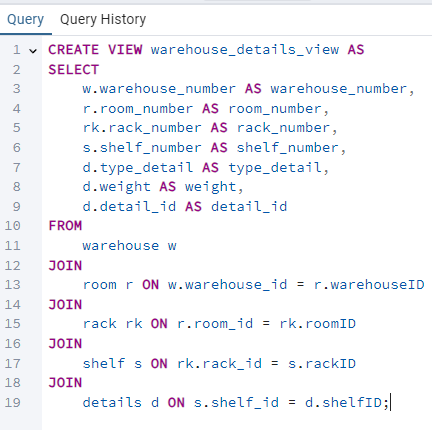


Рисунок 8 – Представление «warehouse\_details\_view»

Для представления «warehouse\_details\_view» созданы необходимые для поддержания актуальности информации в базе данных функции и триггеры:

* insert\_into\_warehouse\_details() – эта функция вставляет данные в иерархию склада: проверяет/добавляет комнату, стеллаж и полку. Добавляет деталь на склад. Данной функции соответствует триггер trigger\_insert\_warehouse\_details, что срабатывает при вставке в представление «warehouse\_details\_view».
* delete\_warehouse\_details() – эта функция удаляет данные из иерархии склада: находит идентификаторы всех связанных объектов, удаляет деталь и пустые полки, стеллажи, комнаты и склады. Данной функции соответствует триггер delete\_warehouse\_details\_trigger, что срабатывает при удалении из представления «warehouse\_details\_view».
* update\_warehouse\_details\_view() – эта функция обновляет данные в иерархии склада: обновляет деталь, номер полки, номер стеллажа, номер комнаты и номер склада. Данной функции соответствует триггер instead\_of\_update\_warehouse\_details\_view, что срабатывает при обновлении представления «warehouse\_details\_view».

В данной базе данных присутствует несколько функций и триггеров для преобразования внутренних данных базы данных в понятный для пользователя вид, без изменения этих данных в записях таблиц:

* convert\_type\_invoice() – эта функция преобразует текстовые значения «отгрузка»/«выгрузка» в булевы значения FALSE/TRUE для поля type\_invoice, позволяя пользователю вводить тип накладной в понятном для себя виде. Данной функции соответствует триггер set\_type\_invoice, что срабатывает перед INSERT/UPDATE в таблице invoice.
* convert\_status() – эта функция преобразует текстовые значения «в процессе»/«завершено» в булевы значения FALSE/TRUE для поля status, позволяя пользователю вводить тип накладной в понятном для себя виде. Данной функции соответствует триггер set\_status, что срабатывает перед INSERT/UPDATE в таблице invoice.

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

При помощи СУБД в 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 Г.

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

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

## Выбор среды для разработки приложения

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

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

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

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

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

## Описание основных модулей и функций интерфейса

Для реализации графического интерфейса и взаимодействия с базой данных PostgreSQL было разработано несколько ключевых модулей и функций.

В модуль окна авторизации создаётся и инициализируется главное окно приложения. Данный модуль использует библиотеку Tkinter для создания пользовательского интерфейса. Основной класс взаимодействия — это метод start\_work, который отвечает за авторизацию пользователя через базу данных и перенаправление его в главное окно приложения после успешной авторизации.

Также в данном модуле реализованы элементы интерфейса для ввода логина и пароля пользователя, их проверки и вызова соответствующих методов базы данных через модуль DB\_methods.

Модуль главного окна приложения содержит функции для управления основным окном приложения и меню. Основные методы:

clear\_window — очищает текущее окно от всех виджетов.

create\_main\_window — создаёт основное окно приложения, куда добавляются элементы меню для поиска деталей, накладных, сотрудников и контрагентов. Присутствует также логика выхода и подтверждения действий.

Модуль методов базы данных отвечает за работу с базой данных PostgreSQL через библиотеку psycopg2. Основные функции включают:

create\_connection — устанавливает соединение с базой данных на основе логина и пароля.

select, insert, update, delete — CRUD-методы для взаимодействия с таблицами базы данных.

log\_action — записывает действия пользователя в файл логов.

Исходный код приложения находится в Приложении В.

## 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 9 и 10.

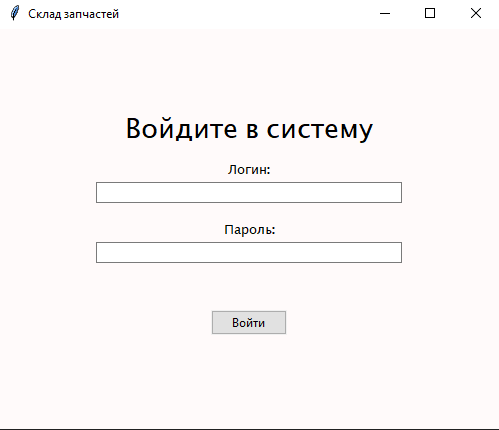


Рисунок 9 – Окно авторизации

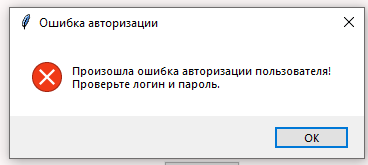


Рисунок 10 – Ошибка авторизации пользователя

Если пользователь ввёл верный логин и пароль, то на экране появиться главное окно приложения и кнопка «Резервное копирование». В верхней части будет выпадающее меню «Найти» и кнопка «Выйти», что показано на рисунке 11.

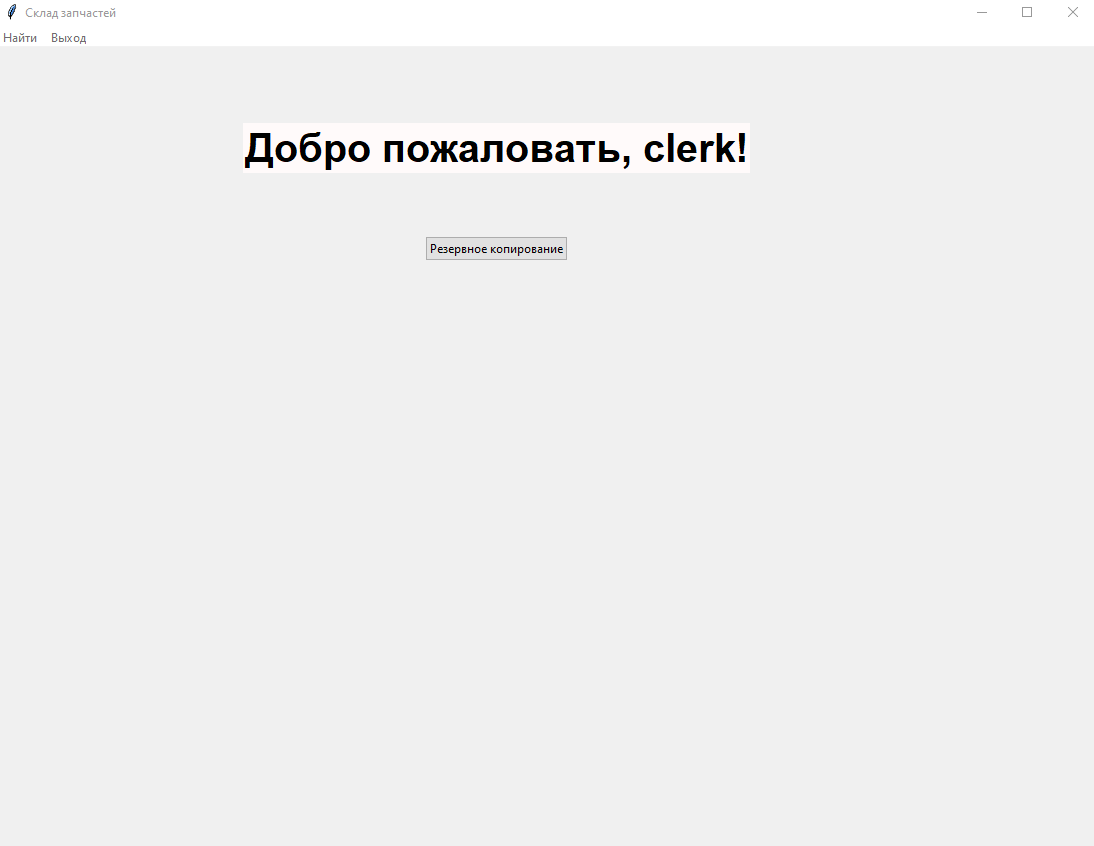


Рисунок 11 – Главное окно

После нажатия на кнопку «Резервное копирование» программа запустит копирование базы данных, а на экране пользователя появится уведомление об успехе копирования, что показано на рисунках 12 и 13.

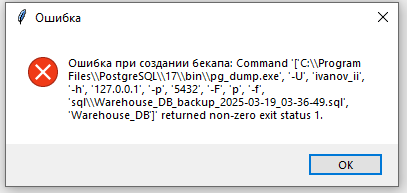


Рисунок 12 – Ошибка копирования

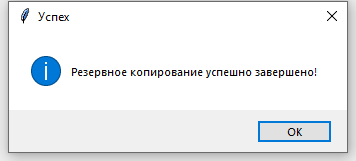


Рисунок 13 – Успешное завершение копирования

Нажав на выпадающее меню «Найти», пользователь может перейти по таблицам базы данных: «Детали», «Накладные», «Сотрудники», «Контрагенты», что показано на рисунке 14.

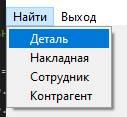
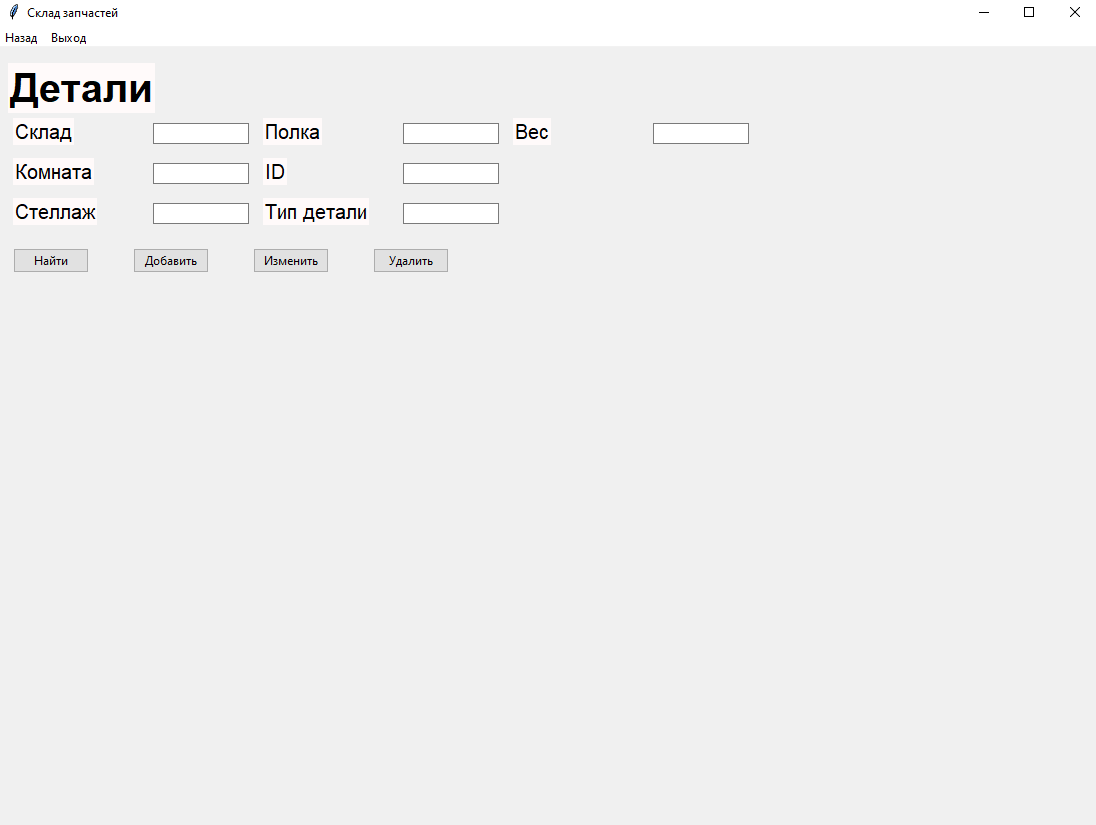


Рисунок 14 – Дерево меню «Найти»

После нажатия на «Детали», «Накладные», «Сотрудники» или «Контрагенты» программа откроет раздел, в котором будут созданы поля, необходимые для редактирования информации в базе данных, что показано на рисунках 15, 16, 17 и 18.

  
Рисунок 15 – Раздел «Детали»

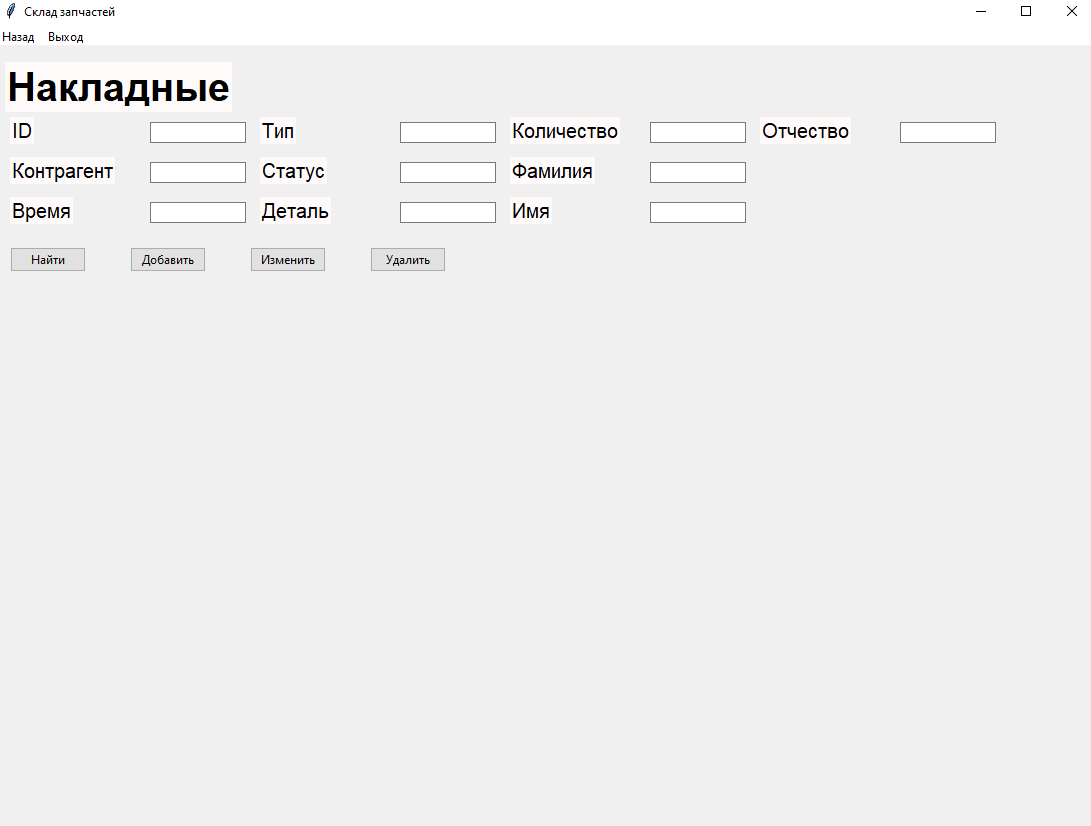


Рисунок 16 – Раздел «Накладные»

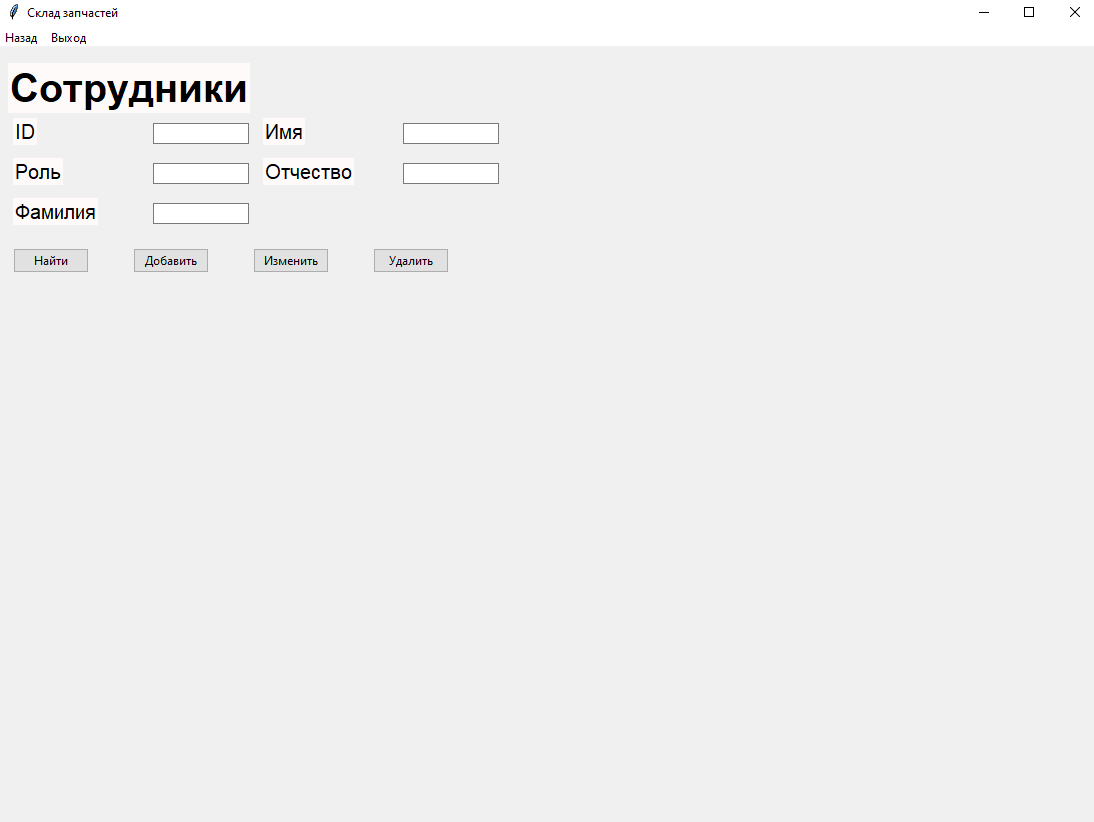


Рисунок 17 – Раздел «Сотрудники»

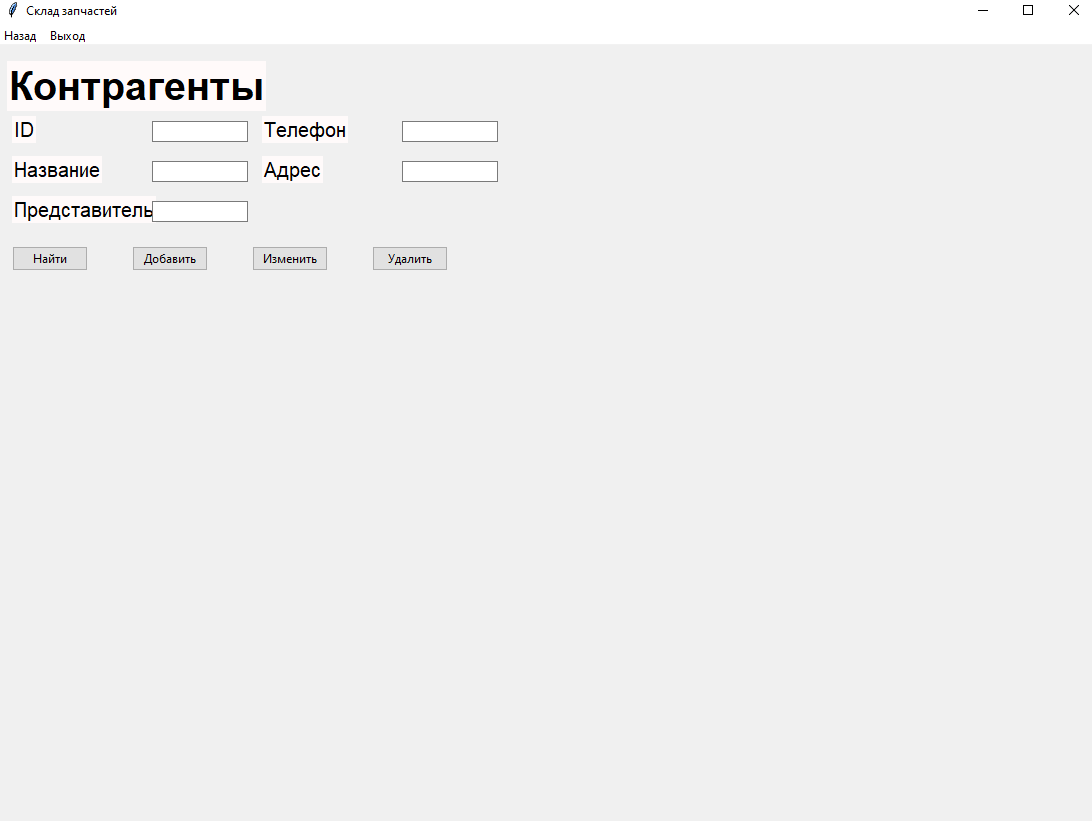


Рисунок 18 – Раздел «Контрагенты»

Если у пользователя нет необходимых привилегий для просмотра раздела, то на экран пользователя будет выведена ошибка, что показано на рисунке 19.

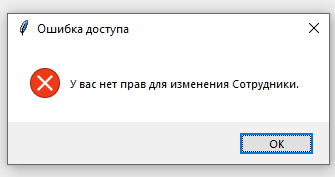


Рисунок 19 – Ошибка доступа к разделу

Если у пользователя нет необходимых привилегий для просмотра, изменения, добавления или удаления информации из базы данных, то будет выведена ошибка, что показано на рисунке 20.

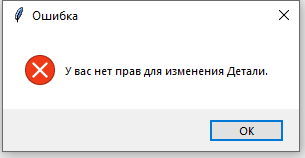


Рисунок 20 – Ошибка наличия привилегий

В верхней части программы вместо меню «Найти» будет кнопка «Назад», нажав на которую пользователь вернутся в главное окно программы, где он сможет вновь выбрать необходимый раздел в меню «Найти».

После нажатия на «Выйти» или на крестик в верхней части окна на экран пользователя, будет выведено окно подтверждения, что показано на рисунке 21. Нажав на кнопку «ОК», программа завершит работу.

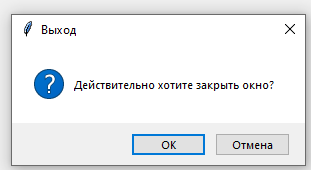


Рисунок 21 – Подтверждение выхода

Исходный код программного средства представлен в Приложении В.

# **Заключение**

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

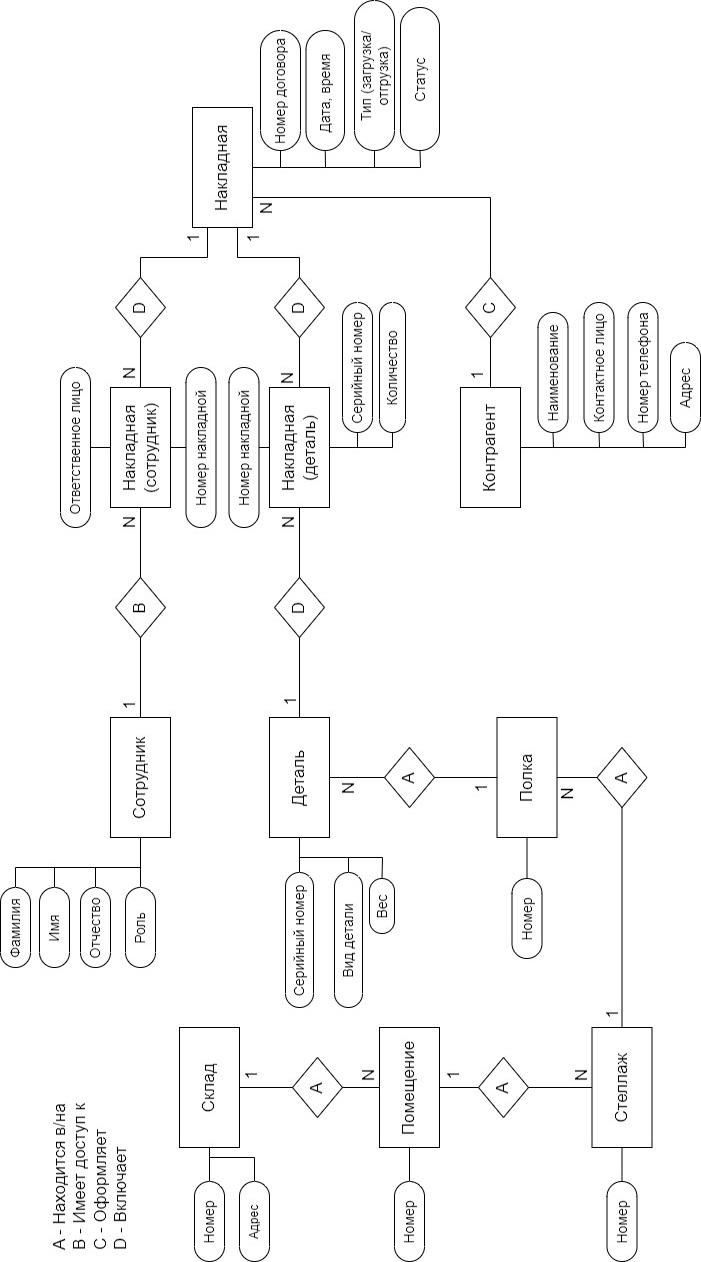
Для решения поставленной задачи была изучена предметная область, определены конечные пользователи, описаны объекты и их атрибуты. Также была разработана концептуальная модель в виде ER-диаграммы. Составлена даталогическая схема отношений. В ходе выполнения данной работы была изучена выбранная СУБД PostgreSQL,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

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

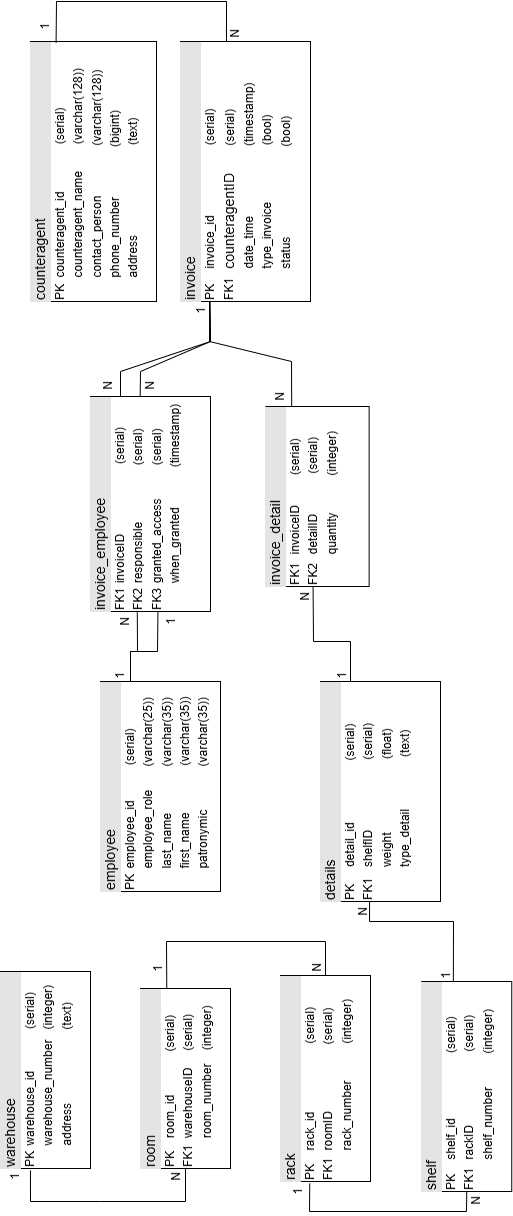
**Перечень используемых информационных источников**

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

# **Приложение А ER-диаграмма**



# **Приложение Б Даталогическая схема**

****

# **Приложение В Исходный код программного средства**

***Листинг В.1 – Модуль окна авторизации***

from tkinter import \*

from tkinter import ttk, messagebox

import DB\_methods as db

import window as win

def start\_work():

login, password = entry\_name.get(), entry\_password.get()

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

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

if active\_user != None:

window.destroy()

main\_win = Tk()

win.create\_main\_window(main\_win, login, active\_user, password)

main\_win.mainloop()

else:

messagebox.showerror('Ошибка авторизации', 'Произошла ошибка авторизации пользователя! Проверьте логин и пароль.')

return 0

window = Tk()

window.geometry('%dx%d+%d+%d' % (500, 400, (window.winfo\_screenwidth()/2) - (500/2), (window.winfo\_screenheight()/2) - (400/2)))

window.title("Склад запчастей")

window.configure(background="#FFFAFA")

middle\_window\_x = 500 / 2

middle\_window\_y = 400 / 3

title\_start = ttk.Label(master=window, text="Войдите в систему", font=("algerian", 20), background="#FFFAFA")

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

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

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

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

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

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

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

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

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

btn\_in = ttk.Button(text="Войти", command=start\_work)

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

window.mainloop()

***Листинг В.2 – Модуль главного окна приложения***

from tkinter import \*

from tkinter import ttk, messagebox, Menu

import DB\_methods as db

from datetime import datetime

import subprocess, os

def quit\_programm():

if messagebox.askokcancel('Выход', 'Действительно хотите закрыть окно?'):

db.close\_connection()

quit()

def clear\_window(window):

for widget in window.winfo\_children():

widget.destroy()

def convert\_to\_standard\_format(date\_str):

date\_formats = [

"%Y-%m-%d %H:%M:%S", # ГГГГ-ММ-ДД ЧЧ:ММ:СС

"%d.%m.%Y %H:%M:%S", # ДД.ММ.ГГГГ ЧЧ:ММ:СС

"%d/%m/%Y %H:%M:%S", # ДД/ММ/ГГГГ ЧЧ:ММ:СС

"%Y-%m-%d", # ГГГГ-ММ-ДД

"%d.%m.%Y", # ДД.ММ.ГГГГ

"%d/%m/%Y", # ДД/ММ/ГГГГ

"%H:%M:%S", # ЧЧ:ММ:СС

"%d-%m-%Y", # ДД-ММ-ГГГГ

]

if len(date\_str.split()) == 2 and len(date\_str.split()[1].split(":")) == 2:

date\_str += ":00"

for fmt in date\_formats:

try:

date\_obj = datetime.strptime(date\_str, fmt)

return date\_obj.strftime("%Y-%m-%d %H:%M:%S")

except ValueError:

continue

return False

def create\_backup():

os.environ['PGPASSWORD'] = password

pg\_dump\_path = r"C:\Program Files\PostgreSQL\17\bin\pg\_dump.exe"

username = login

host = "127.0.0.1"

port = "5432"

database = "Warehouse\_DB"

backup\_file = os.path.join('sql\\backup\\', f"{database}\_backup\_{datetime.now().strftime('%Y-%m-%d\_%H-%M-%S')}.sql")

command = [

pg\_dump\_path,

"-U", username,

"-h", host,

"-p", port,

"-F", "p",

"-f", backup\_file,

database

]

try:

result = subprocess.run(command, check=True, text=True, capture\_output=True)

messagebox.showinfo("Успех", "Резервное копирование успешно завершено!")

except subprocess.CalledProcessError as e:

messagebox.showerror("Ошибка", f"Ошибка при создании бекапа: {e}")

def create\_main\_window(win, log, user, pas):

global active\_user, window, login, password

window = win

active\_user = user

login = log

password = pas

clear\_window(window)

window.title("Склад запчастей")

window.geometry('%dx%d+%d+%d' % (1100, 800, (window.winfo\_screenwidth() / 2) - (1000 / 2), (window.winfo\_screenheight() / 2) - (800 / 2)))

window.option\_add("\*tearOff", FALSE)

find\_menu = Menu()

find\_menu.add\_command(label="Деталь", command=create\_select\_detail)

find\_menu.add\_command(label='Накладная', command=create\_select\_invoice)

find\_menu.add\_command(label="Сотрудник", command=create\_select\_employee)

find\_menu.add\_command(label="Контрагент", command=create\_select\_counteragent)

menu = Menu()

menu.add\_cascade(label="Найти", menu=find\_menu)

menu.add\_command(label="Выход", command=quit\_programm)

window.config(menu=menu)

label\_title = ttk.Label(window, text=f'Добро пожаловать, {login}!', font=("arial", 30, "bold"), background="#FFFAFA")

label\_title.place(x=1000 // 2, y=(800 // 2)-300, anchor='center')

# Кнопка для резервного копирования

backup\_button = ttk.Button(window, text="Резервное копирование", command=create\_backup)

backup\_button.place(x=1000 // 2, y=(800 // 2)-200, anchor='center') # Разместить кнопку немного ниже заголовка

# Кнопка для создания пользователей

'''backup\_button = ttk.Button(window, text="Добавить пользователя", command=create\_user)

backup\_button.place(x=1000 // 2, y=(800 // 2)-50, anchor='center')'''

window.protocol("WM\_DELETE\_WINDOW", quit\_programm)

def menu\_window(window, table, columns):

label\_title = ttk.Label(window, text=f'{table}', font=("arial", 30, "bold"), background="#FFFAFA")

label\_title.place(x=10, y=15)

x\_label, y\_label, n = 15, 70, 0

ENTRYS\_LIST = []

ENTRYS\_LIST\_titles = []

window.option\_add("\*tearOff", FALSE)

menu = Menu()

menu.add\_cascade(label="Назад", command=lambda: create\_main\_window(window, login, active\_user, password))

menu.add\_command(label="Выход", command=quit\_programm)

window.config(menu=menu)

for num in range(len(columns)):

new\_title = ttk.Label(window, text=columns[num], font=("arial", 15), background='#FFFAFA')

new\_title.place(x=x\_label, y=y\_label)

new\_input = ttk.Entry(window, width=15, background='#FFFAFA')

new\_input.place(x=x\_label + 140, y=y\_label + 5)

ENTRYS\_LIST.append(new\_input)

ENTRYS\_LIST\_titles.append(columns[num])

y\_label += 40

n += 1

if n == 3:

n = 0

x\_label += 250

y\_label = 70

btn\_find = ttk.Button(window, text='Найти', command=lambda: find\_item(ENTRYS\_LIST, label\_title))

btn\_insert = ttk.Button(window, text='Добавить', command=lambda: insert\_item(ENTRYS\_LIST, label\_title))

btn\_update = ttk.Button(window, text='Изменить', command=lambda: where\_update(ENTRYS\_LIST\_titles, ENTRYS\_LIST, label\_title))

btn\_delete = ttk.Button(window, text='Удалить', command=lambda: delete\_item(ENTRYS\_LIST, label\_title))

btn\_find.place(x=15, y=200)

btn\_insert.place(x=135, y=200)

btn\_update.place(x=255, y=200)

btn\_delete.place(x=375, y=200)

window.protocol("WM\_DELETE\_WINDOW", quit\_programm)

#методы бд

def find\_item(ENTRYS\_LIST, label\_title):

entries = [entry.get() for entry in ENTRYS\_LIST]

if label\_title.cget('text') == 'Детали':

res\_sql = db.select('warehouse\_details\_view', '\*', [f'warehouse\_number = {entries[0]}',

f'room\_number = {entries[1]}',

f'rack\_number = {entries[2]}',

f'shelf\_number = {entries[3]}',

f'detail\_id = {entries[4]}',

f'type\_detail = {"'"+entries[5]+"'"}',

f'weight = {entries[6]}'])

if type(res\_sql) == str:

messagebox.showerror('Нет результата', res\_sql)

elif len(res\_sql) == 0:

messagebox.showerror('Результат запроса', 'По вашему запросу ничего не найдено')

else:

new\_window(('Склад', 'Комната', 'Стеллаж','Полка', 'Тип', 'Вес', 'ID'), res\_sql)

elif label\_title.cget('text') == 'Накладные':

date = convert\_to\_standard\_format(entries[2])

if (date == False) and (entries[2] != ""):

messagebox.showerror('Неверный формат', "Неверный формат даты!")

elif entries[2] == "":

res\_sql = db.select('invoice\_details\_view', '\*', [f'invoice\_id = {entries[0]}',

f'counteragent\_name = {"'"+entries[1].replace("'", '"')+"'"}',

f'date\_time::date = {entries[2]}',

f'type\_invoice\_text = {"'"+entries[3].capitalize()+"'"}',

f'status\_text = {"'"+entries[4].capitalize()+"'"}',

f'type\_detail = {"'"+entries[5]+"'"}',

f'quantity = {entries[6]}',

f'responsible\_last\_name = {"'"+entries[7]+"'"}',

f'responsible\_first\_name = {"'"+entries[8]+"'"}',

f'responsible\_patronymic = {"'"+entries[9]+"'"}'])

else:

res\_sql = db.select('invoice\_details\_view', '\*', [f'invoice\_id = {entries[0]}',

f'counteragent\_name = {"'"+entries[1].replace("'", '"')+"'"}',

f'date\_time::date = {"'"+entries[2]+"'"}',

f'type\_invoice\_text = {"'"+entries[3].capitalize()+"'"}',

f'status\_text = {"'"+entries[4].capitalize()+"'"}',

f'type\_detail = {"'"+entries[5]+"'"}',

f'quantity = {entries[6]}',

f'responsible\_last\_name = {"'"+entries[7]+"'"}',

f'responsible\_first\_name = {"'"+entries[8]+"'"}',

f'responsible\_patronymic = {"'"+entries[9]+"'"}'])

if type(res\_sql) == str:

messagebox.showerror('Нет результата', res\_sql)

elif len(res\_sql) == 0:

messagebox.showerror('Результат запроса', 'По вашему запросу ничего не найдено')

else:

new\_window(('ID', 'Контрагент', 'Время', 'Тип', 'Статус', "Деталь", "Количество", "Фамилия", "Имя", "Отчество"), res\_sql)

elif label\_title.cget('text') == 'Сотрудники':

res\_sql = db.select('employee', '\*', [f'employee\_id = {entries[0]}',

f'employee\_role = {"'"+entries[1]+"'"}',

f'last\_name = {"'"+entries[2]+"'"}',

f'first\_name = {"'"+entries[3]+"'"}',

f'patronymic = {"'"+entries[4]+"'"}'])

if type(res\_sql) == str:

messagebox.showerror('Нет результата', res\_sql)

elif len(res\_sql) == 0:

messagebox.showerror('Результат запроса', 'По вашему запросу ничего не найдено')

else:

new\_window(('ID', 'Должность', 'Фамилия', 'Имя', 'Отчество'), res\_sql)

elif label\_title.cget('text') == 'Контрагенты':

res\_sql = db.select('counteragent', '\*', [f'counteragent\_id = {entries[0]}',

f'counteragent\_name = {"'"+entries[1]+"'"}',

f'contact\_person = {"'"+entries[2]+"'"}',

f'phone\_number = {entries[3]}',

f'address = {"'"+entries[4]+"'"}'])

if type(res\_sql) == str:

messagebox.showerror('Нет результата', res\_sql)

elif len(res\_sql) == 0:

messagebox.showerror('Результат запроса', 'По вашему запросу ничего не найдено')

else:

new\_window(('ID', 'Контрагент', 'Представиль', 'Телефон', 'Адрес'), res\_sql)

def insert\_item(ENTRYS\_LIST, label\_title):

entries = [entry.get() for entry in ENTRYS\_LIST]

if label\_title.cget('text') == 'Детали':

res\_sql = db.insert('warehouse\_details\_view', [f'warehouse\_number = {entries[0]}',

f'room\_number = {entries[1]}',

f'rack\_number = {entries[2]}',

f'shelf\_number = {entries[3]}',

f'detail\_id = {entries[4]}',

f'type\_detail = {entries[5]}',

f'weight = {entries[6]}'])

elif label\_title.cget('text') == 'Накладные':

date = convert\_to\_standard\_format(entries[2])

if (date == False) and (entries[2] != ""):

messagebox.showerror('Неверный формат', "Неверный формат даты!")

elif entries[2] == "":

res\_sql = db.insert('invoice\_details\_view', [f'invoice\_id = {entries[0]}',

f'counteragent\_name = {entries[1].replace("'", '"')}',

f'date\_time = {entries[2]}',

f'type\_invoice\_text = {entries[3]}',

f'status\_text = {entries[4]}',

f'type\_detail = {entries[5]}',

f'quantity = {entries[6]}',

f'responsible\_last\_name = {entries[7]}',

f'responsible\_first\_name = {entries[8]}',

f'responsible\_patronymic = {entries[9]}'])

else:

res\_sql = db.insert('invoice\_details\_view', [f'invoice\_id = {entries[0]}',

f'counteragent\_name = {entries[1].replace("'", '"')}',

f'date\_time = {entries[2]}',

f'type\_invoice\_text = {entries[3]}',

f'status\_text = {entries[4]}',

f'type\_detail = {entries[5]}',

f'quantity = {entries[6]}',

f'responsible\_last\_name = {entries[7]}',

f'responsible\_first\_name = {entries[8]}',

f'responsible\_patronymic = {entries[9]}'])

elif label\_title.cget('text') == 'Сотрудники':

res\_sql = db.insert('employee', [f'employee\_id = {entries[0]}',

f'employee\_role = {entries[1]}',

f'last\_name = {entries[2]}',

f'first\_name = {entries[3]}',

f'patronymic = {entries[4]}'])

elif label\_title.cget('text') == 'Контрагенты':

res\_sql = db.insert('counteragent', [f'counteragent\_id = {entries[0]}',

f'counteragent\_name = {entries[1]}',

f'contact\_person = {entries[2]}',

f'phone\_number = {entries[3]}',

f'address = {entries[4]}'])

if res\_sql != True:

messagebox.showerror('Ошибка', res\_sql)

else:

messagebox.showinfo('Результат', 'Добавление прошло успешно!')

def delete\_item(ENTRYS\_LIST, label\_title):

entries = [entry.get() for entry in ENTRYS\_LIST]

if label\_title.cget('text') == 'Детали':

res\_sql = db.delete('warehouse\_details\_view', [f'warehouse\_number = {entries[0]}',

f'room\_number = {entries[1]}',

f'rack\_number = {entries[2]}',

f'shelf\_number = {entries[3]}',

f'detail\_id = {entries[4]}',

f'type\_detail = {"'"+entries[5]+"'"}',

f'weight = {entries[6]}'])

elif label\_title.cget('text') == 'Накладные':

date = convert\_to\_standard\_format(entries[2])

if (date == False) and (entries[2] != ""):

messagebox.showerror('Неверный формат', "Неверный формат даты!")

elif entries[2] == "":

res\_sql = db.delete('invoice\_details\_view', [f'invoice\_id = {entries[0]}',

f'counteragent\_name = {"'"+entries[1].replace("'", '"')+"'"}',

f'date\_time::date = {entries[2]}',

f'type\_invoice\_text = {entries[3]}',

f'status\_text = {entries[4]}',

f'type\_detail = {"'"+entries[5]+"'"}',

f'quantity = {entries[6]}',

f'responsible\_last\_name = {"'"+entries[7]+"'"}',

f'responsible\_first\_name = {"'"+entries[8]+"'"}',

f'responsible\_patronymic = {"'"+entries[9]+"'"}'])

else:

res\_sql = db.delete('invoice\_details\_view', [f'invoice\_id = {entries[0]}',

f'counteragent\_name = {"'"+entries[1].replace("'", '"')+"'"}',

f'date\_time::date = {"'"+entries[2]+"'"}',

f'type\_invoice\_text = {entries[3]}',

f'status\_text = {entries[4]}',

f'type\_detail = {"'"+entries[5]+"'"}',

f'quantity = {entries[6]}',

f'responsible\_last\_name = {"'"+entries[7]+"'"}',

f'responsible\_first\_name = {"'"+entries[8]+"'"}',

f'responsible\_patronymic = {"'"+entries[9]+"'"}'])

elif label\_title.cget('text') == 'Сотрудники':

res\_sql = db.delete('employee', [f'employee\_id = {entries[0]}',

f'employee\_role = {"'"+entries[1]+"'"}',

f'last\_name = {"'"+entries[2]+"'"}',

f'first\_name = {"'"+entries[3]+"'"}',

f'patronymic = {"'"+entries[4]+"'"}'])

elif label\_title.cget('text') == 'Контрагенты':

res\_sql = db.delete('counteragent', [f'counteragent\_id = {entries[0]}',

f'counteragent\_name = {"'"+entries[1].replace("'", '"')+"'"}',

f'contact\_person = {"'"+entries[2]+"'"}',

f'phone\_number = {entries[3]}',

f'address = {"'"+entries[4]+"'"}'])

if res\_sql != True:

messagebox.showerror('Ошибка', res\_sql)

else:

messagebox.showinfo('Результат', 'Удаление прошло успешно!')

def where\_update(colmns, ENTRYS\_LIST, label\_title):

global toplev

k = 0

for i in [entry.get() for entry in ENTRYS\_LIST]:

if i == "":

k += 1

if len([entry.get() for entry in ENTRYS\_LIST]) == k:

messagebox.showerror('Ошибка', 'Введите хотя бы один параметр!')

return 0

toplev = Toplevel()

toplev.geometry('1000x400')

toplev.title('Новые данные')

ENTRYS\_LIST\_wheres = []

x\_label, y\_label, n = 15, 70, 0

for num in range(len(colmns)):

new\_title = ttk.Label(master=toplev, text=colmns[num], font=("arial", 15), background='#FFFAFA')

new\_title.place(x=x\_label, y=y\_label)

new\_input = ttk.Entry(master=toplev, width=15, background='#FFFAFA')

new\_input.place(x=x\_label + 120, y=y\_label + 5)

ENTRYS\_LIST\_wheres.append(new\_input)

y\_label += 40

n += 1

if n == 3:

x\_label += 250

y\_label, n = 70, 0

btn\_update = ttk.Button(master=toplev, text='Изменить', command=lambda: update\_item(ENTRYS\_LIST\_wheres, ENTRYS\_LIST, label\_title))

btn\_update.place(x=15, y=250)

def update\_item(ENTRYS\_LIST, ENTRYS\_LIST\_wheres, label\_title):

entrys1 = [entry.get() for entry in ENTRYS\_LIST]

entrys2 = [entry.get() for entry in ENTRYS\_LIST\_wheres]

if label\_title.cget('text') == 'Детали':

res\_sql = db.update('warehouse\_details\_view', [f'warehouse\_number = {entrys1[0]}',

f'room\_number = {entrys1[1]}',

f'rack\_number = {entrys1[2]}',

f'shelf\_number = {entrys1[3]}',

f'detail\_id = {entrys1[4]}',

f'type\_detail = {"'"+entrys1[5]+"'"}',

f'weight = {entrys1[6]}'],

[f'warehouse\_number = {entrys2[0]}',

f'room\_number = {entrys2[1]}',

f'rack\_number = {entrys2[2]}',

f'shelf\_number = {entrys2[3]}',

f'detail\_id = {entrys2[4]}',

f'type\_detail = {"'"+entrys2[5]+"'"}',

f'weight = {entrys2[6]}'])

elif label\_title.cget('text') == 'Накладные':

date = convert\_to\_standard\_format(entrys1[2])

date2 = convert\_to\_standard\_format(entrys2[2])

if (date == False) and ((entrys1[2] != "") or (entrys2[2] != "")):

messagebox.showerror('Неверный формат', "Неверный формат даты!")

elif (entrys1[2] == "") and (entrys2[2] == ""):

if login != 'clerk':

res\_sql = db.update('invoice\_details\_view', [f'invoice\_id = {entrys1[0]}',

f'counteragent\_name = {"'"+entrys1[1].replace("'", '"')+"'"}',

f'date\_time = {entrys1[2]}',

f'type\_invoice\_text = {"'"+entrys1[3]+"'"}',

f'status\_text = {"'"+entrys1[4]+"'"}',

f'type\_detail = {"'"+entrys1[5]+"'"}',

f'quantity = {entrys1[6]}',

f'responsible\_last\_name = {"'"+entrys1[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys1[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys1[9]+"'"}'],

[f'invoice\_id = {entrys2[0]}',

f'counteragent\_name = {"'"+entrys2[1].replace("'", '"')+"'"}',

f'date\_time = {entrys2[2]}',

f'type\_invoice\_text = {"'"+entrys2[3]+"'"}',

f'status\_text = {"'"+entrys2[4]+"'"}',

f'type\_detail = {"'"+entrys2[5]+"'"}',

f'quantity = {entrys2[6]}',

f'responsible\_last\_name = {"'"+entrys2[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys2[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys2[9]+"'"}'])

else:

res\_sql = db.update('invoice\_details\_view', [f'invoice\_id = ',

f'counteragent\_name = ',

f'date\_time = ',

f'type\_invoice\_text = ',

f'status\_text = {"'"+entrys1[4]+"'"}',

f'type\_detail = ',

f'quantity = ',

f'responsible\_last\_name = ',

f'responsible\_first\_name = ',

f'responsible\_patronymic = '],

[f'invoice\_id = {entrys2[0]}',

f'counteragent\_name = {"'"+entrys2[1].replace("'", '"')+"'"}',

f'date\_time = {entrys2[2]}',

f'type\_invoice\_text = {"'"+entrys2[3]+"'"}',

f'status\_text = {"'"+entrys2[4]+"'"}',

f'type\_detail = {"'"+entrys2[5]+"'"}',

f'quantity = {entrys2[6]}',

f'responsible\_last\_name = {"'"+entrys2[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys2[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys2[9]+"'"}'])

else:

if (entrys1[2] != "") and ((entrys2[2] == "")):

res\_sql = db.update('invoice\_details\_view', [f'invoice\_id = {entrys1[0]}',

f'counteragent\_name = {"'"+entrys1[1].replace("'", '"')+"'"}',

f'date\_time = {"'"+date+"'"}',

f'type\_invoice\_text = {"'"+entrys1[3]+"'"}',

f'status\_text = {"'"+entrys1[4]+"'"}',

f'type\_detail = {"'"+entrys1[5]+"'"}',

f'quantity = {entrys1[6]}',

f'responsible\_last\_name = {"'"+entrys1[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys1[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys1[9]+"'"}'],

[f'invoice\_id = {entrys2[0]}',

f'counteragent\_name = {"'"+entrys2[1].replace("'", '"')+"'"}',

f'date\_time = {entrys2[2]}',

f'type\_invoice\_text = {"'"+entrys2[3]+"'"}',

f'status\_text = {"'"+entrys2[4]+"'"}',

f'type\_detail = {"'"+entrys2[5]+"'"}',

f'quantity = {entrys2[6]}',

f'responsible\_last\_name = {"'"+entrys2[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys2[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys2[9]+"'"}'])

elif (entrys1[2] == "") and ((entrys2[2] != "")):

if login != 'clerk':

res\_sql = db.update('invoice\_details\_view', [f'invoice\_id = {entrys1[0]}',

f'counteragent\_name = {"'"+entrys1[1].replace("'", '"')+"'"}',

f'date\_time = {entrys1[2]}',

f'type\_invoice\_text = {"'"+entrys1[3]+"'"}',

f'status\_text = {"'"+entrys1[4]+"'"}',

f'type\_detail = {"'"+entrys1[5]+"'"}',

f'quantity = {entrys1[6]}',

f'responsible\_last\_name = {"'"+entrys1[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys1[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys1[9]+"'"}'],

[f'invoice\_id = {entrys2[0]}',

f'counteragent\_name = {"'"+entrys2[1].replace("'", '"')+"'"}',

f'date\_time = {"'"+date2+"'"}',

f'type\_invoice\_text = {"'"+entrys2[3]+"'"}',

f'status\_text = {"'"+entrys2[4]+"'"}',

f'type\_detail = {"'"+entrys2[5]+"'"}',

f'quantity = {entrys2[6]}',

f'responsible\_last\_name = {"'"+entrys2[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys2[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys2[9]+"'"}'])

else:

res\_sql = db.update('invoice\_details\_view', [f'invoice\_id = ',

f'counteragent\_name = ',

f'date\_time = ',

f'type\_invoice\_text = ',

f'status\_text = {"'"+entrys1[4]+"'"}',

f'type\_detail = ',

f'quantity = ',

f'responsible\_last\_name = ',

f'responsible\_first\_name = ',

f'responsible\_patronymic = '],

[f'invoice\_id = {entrys2[0]}',

f'counteragent\_name = {"'"+entrys2[1].replace("'", '"')+"'"}',

f'date\_time = {"'"+date2+"'"}',

f'type\_invoice\_text = {"'"+entrys2[3]+"'"}',

f'status\_text = {"'"+entrys2[4]+"'"}',

f'type\_detail = {"'"+entrys2[5]+"'"}',

f'quantity = {entrys2[6]}',

f'responsible\_last\_name = {"'"+entrys2[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys2[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys2[9]+"'"}'])

elif (entrys1[2] != "") and ((entrys2[2] != "")):

if login != 'clerk':

res\_sql = db.update('invoice\_details\_view', [f'invoice\_id = {entrys1[0]}',

f'counteragent\_name = {"'"+entrys1[1].replace("'", '"')+"'"}',

f'date\_time = {"'"+date+"'"}',

f'type\_invoice\_text = {"'"+entrys1[3]+"'"}',

f'status\_text = {"'"+entrys1[4]+"'"}',

f'type\_detail = {"'"+entrys1[5]+"'"}',

f'quantity = {entrys1[6]}',

f'responsible\_last\_name = {"'"+entrys1[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys1[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys1[9]+"'"}'],

[f'invoice\_id = {entrys2[0]}',

f'counteragent\_name = {"'"+entrys2[1].replace("'", '"')+"'"}',

f'date\_time = {"'"+date2+"'"}',

f'type\_invoice\_text = {"'"+entrys2[3]+"'"}',

f'status\_text = {"'"+entrys2[4]+"'"}',

f'type\_detail = {"'"+entrys2[5]+"'"}',

f'quantity = {entrys2[6]}',

f'responsible\_last\_name = {"'"+entrys2[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys2[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys2[9]+"'"}'])

else:

res\_sql = db.update('invoice\_details\_view', [f'invoice\_id = ',

f'counteragent\_name = ',

f'date\_time = ',

f'type\_invoice\_text = ',

f'status\_text = {"'"+entrys1[4]+"'"}',

f'type\_detail = ',

f'quantity = ',

f'responsible\_last\_name = ',

f'responsible\_first\_name = ',

f'responsible\_patronymic = '],

[f'invoice\_id = {entrys2[0]}',

f'counteragent\_name = {"'"+entrys2[1].replace("'", '"')+"'"}',

f'date\_time = {"'"+date2+"'"}',

f'type\_invoice\_text = {"'"+entrys2[3]+"'"}',

f'status\_text = {"'"+entrys2[4]+"'"}',

f'type\_detail = {"'"+entrys2[5]+"'"}',

f'quantity = {entrys2[6]}',

f'responsible\_last\_name = {"'"+entrys2[7]+"'"}',

f'responsible\_first\_name = {"'"+entrys2[8]+"'"}',

f'responsible\_patronymic = {"'"+entrys2[9]+"'"}'])

elif label\_title.cget('text') == 'Сотрудники':

res\_sql = db.update('employee', [f'employee\_id = {entrys1[0]}',

f'employee\_role = {"'"+entrys1[1]+"'"}',

f'last\_name = {"'"+entrys1[2]+"'"}',

f'first\_name = {"'"+entrys1[3]+"'"}',

f'patronymic = {"'"+entrys1[4]+"'"}'],

[f'employee\_id = {entrys2[0]}',

f'employee\_role = {"'"+entrys2[1]+"'"}',

f'last\_name = {"'"+entrys2[2]+"'"}',

f'first\_name = {"'"+entrys2[3]+"'"}',

f'patronymic = {"'"+entrys2[4]+"'"}'])

elif label\_title.cget('text') == 'Контрагенты':

res\_sql = db.update('counteragent', [f'counteragent\_id = {entrys1[0]}',

f'counteragent\_name = {"'"+entrys1[1].replace("'", '"')+"'"}',

f'contact\_person = {"'"+entrys1[2]+"'"}',

f'phone\_number = {entrys1[3]}',

f'address = {"'"+entrys1[4]+"'"}'],

[f'counteragent\_id = {entrys2[0]}',

f'counteragent\_name = {"'"+entrys2[1].replace("'", '"')+"'"}',

f'contact\_person = {"'"+((entrys2[2]).replace("'", '"'))+"'"}',

f'phone\_number = {entrys2[3]}',

f'address = {"'"+entrys2[4]+"'"}'])

if res\_sql != True:

messagebox.showerror('Ошибка', res\_sql)

else:

toplev.destroy()

messagebox.showinfo('Результат', 'Изменение прошло успешно!')

def new\_window(columns, result):

topLev = Toplevel()

topLev.geometry('1200x400')

new\_label = ttk.Label(topLev, text='Результат поиска:', font=("arial", 15), background='#FFFAFA')

new\_label.place(x=5, y=15)

tree = ttk.Treeview(topLev, columns=columns, show='headings')

tree.place(x=15, y=50)

for i in range(len(columns)):

tree.heading(columns[i], text=columns[i], anchor=W)

for row in result:

tree.insert("", END, values=row)

#окна интерфейса

def create\_select\_detail():

clear\_window(window)

columns = ('Склад', 'Комната', 'Стеллаж', 'Полка', 'ID', 'Тип детали', 'Вес')

res\_select = db.select('details')

if type(res\_select) == str:

messagebox.showerror('Ошибка доступа', res\_select)

create\_main\_window(window, login, active\_user, password)

else:

clear\_window(window)

menu\_window(window, 'Детали', columns)

def create\_select\_invoice():

clear\_window(window)

columns = ('ID', 'Контрагент', 'Время', 'Тип', 'Статус', "Деталь", "Количество", "Фамилия", "Имя", "Отчество")

res\_select = db.select('invoice')

if type(res\_select) == str:

messagebox.showerror('Ошибка доступа', res\_select)

create\_main\_window(window, login, active\_user, password)

else:

clear\_window(window)

menu\_window(window, 'Накладные', columns)

def create\_select\_employee():

clear\_window(window)

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

res\_select = db.select('employee')

if type(res\_select) == str:

messagebox.showerror('Ошибка доступа', res\_select)

create\_main\_window(window, login, active\_user, password)

else:

clear\_window(window)

menu\_window(window, 'Сотрудники', columns)

def create\_select\_counteragent():

clear\_window(window)

columns = ('ID', 'Название', 'Представитель', 'Телефон', 'Адрес')

res\_select = db.select('counteragent')

if type(res\_select) == str:

messagebox.showerror('Ошибка выборки', res\_select)

create\_main\_window(window, login, active\_user, password)

else:

clear\_window(window)

menu\_window(window, 'Контрагенты', columns)

***Листинг В.3 – Модуль методов базы данных***

import psycopg2 as ps2

from datetime import datetime

host = "127.0.0.1"

db\_name = "Warehouse\_DB"

connection = None

labels = {'details': 'Детали',

'warehouse\_details\_view': 'Детали',

'invoice': 'Накладные',

'invoice\_details\_view': 'Накладные',

'employee': 'Сотрудники',

'counteragent': 'Контрагенты'}

def create\_connection(log, password):

global connection, login

login = log

try:

connection = ps2.connect(

host=host,

user=login,

password=password,

database=db\_name

)

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

except Exception as ex:

print(f"[INFO] Error while working with PostgreSQL: {ex}")

return connection

return connection

def close\_connection():

global connection

if connection:

connection.close()

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

def no\_privilege(table):

return f'У вас нет прав для изменения {labels[table]}.'

def transaction\_error():

return 'Произошла ошибка! Перезапустите приложение.'

def no\_data():

return 'Информации по введёным данным не существует!'

def input\_error():

return 'Произошла ошибка! Обязательные поля не заполнены.'

def error(e):

return e

def log\_action(login, action, table, details):

with open('user\_actions\_log.txt', 'a', encoding='utf-8') as file:

timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

log\_entry = f"{login}: {action} table {table}, {details}, time {timestamp};\n"

file.write(log\_entry)

def create\_user(log, pas, role):

create\_user\_sql = f"CREATE USER {log} WITH PASSWORD '{pas}';"

grant\_role = f"GRANT {role} TO {log};"

if connection:

with connection.cursor() as cursor:

try:

cursor.execute(create\_user\_sql)

cursor.execute(grant\_role)

return True

except ps2.errors.InsufficientPrivilege:

connection.rollback()

return no\_privilege("Сотрудники")

except ps2.errors.InFailedSqlTransaction:

connection.rollback()

return transaction\_error()

def select(table, columns='\*', where=None):

sql = f"SELECT {columns} FROM {table}"

new\_where = []

if where is not None:

for i in where:

if i.split("=")[1] != " " and i.split("=")[1] != " ''":

new\_where.append(i)

if len(new\_where) == 0:

return "Введите значения для поиска!"

if len(new\_where) != 1:

conditions = ' AND '.join(new\_where)

else:

conditions = new\_where[0]

sql += f" WHERE {conditions};"

print(sql)

if connection:

with connection.cursor() as cursor:

try:

cursor.execute(sql)

rows = cursor.fetchall()

return rows

except ps2.errors.InsufficientPrivilege as e:

print(e)

connection.rollback()

return no\_privilege(table)

except ps2.errors.InFailedSqlTransaction:

connection.rollback()

return transaction\_error()

except ps2.errors.UndefinedColumn as e:

print(e)

connection.rollback()

return no\_data()

except ps2.errors.ObjectNotInPrerequisiteState as e:

print(e)

connection.rollback()

return no\_privilege(table)

except Exception as e:

print(e)

connection.rollback()

return error(e)

def insert(table, columns\_values):

if not columns\_values:

return 'Добавление было приостановлено. Нет информации для добавления.'

columns\_list = []

for col in columns\_values:

if col.split('=')[1] != " ":

columns\_list.append(col.split('=')[0])

columns = ', '.join(columns\_list)

valuse\_list = []

for col in columns\_values:

try:

if int(col.split('=')[1]):

valuse\_list.append(str(col.split('=')[1]))

except:

try:

if str(col.split('=')[1]):

if col.split('=')[1] != " ":

valuse\_list.append("'"+str(col.split(' = ')[1])+"'")

except:

return f'Неверно введены данные для поля {col.split('=')[0]}'

if table != 'invoice':

sql = f"INSERT INTO {table} ({columns}) VALUES ({", ".join(valuse\_list)});"

else:

sql = f"INSERT INTO {table} ({columns}) VALUES ({', '.join(valuse\_list[:-2])});"

sql = f"INSERT INTO invoice\_detail (detailID, quantity) VALUES ((SELECT detail\_id FROM details WHERE type\_detail = {valuse\_list[4]} LIMIT 1), {valuse\_list[5]});"

print(sql)

if connection:

with connection.cursor() as cursor:

try:

cursor.execute("BEGIN;")

cursor.execute(sql)

connection.commit()

details = f"new item with values {', '.join([f'{col} = {val}' for col, val in zip(columns\_list, valuse\_list)])}"

log\_action(login, "insert", table, details)

return True

except ps2.errors.InsufficientPrivilege as e:

print(e)

connection.rollback()

return no\_privilege(table)

except ps2.errors.InFailedSqlTransaction as e:

print(e)

connection.rollback()

return transaction\_error()

except ps2.errors.NotNullViolation as e:

print(e)

connection.rollback()

return input\_error()

except ps2.errors.UndefinedColumn as e:

print(e)

connection.rollback()

return no\_data()

except Exception as e:

print(e)

connection.rollback()

return error(e)

def update(table, columns='', where=False):

if not where:

return 'Обновление приостановлено. Нет информации об изменяемой записи.'

if not columns:

return 'Обновление приостановлено. Нет информации для изменения записи.'

k = 0

for i in where:

if i.split('=')[1] == "":

k += 1

if len(where) == k:

return 'Введите хотя бы одно условие для обновления!'

if len(columns) == 0:

return 'Разрешено обновлять только статус накладной!'

new\_columns = []

new\_where = []

# Обработка columns и where без изменения двойных кавычек

for i in columns:

if (i.split("=")[1] != " ") and (i.split(" = ")[1] != "''"):

new\_columns.append(i)

for i in where:

if (i.split("=")[1] != " ") and (i.split(" = ")[1] != "''"):

new\_where.append(i)

# Формирование SQL-запроса без замены кавычек

sql = f'UPDATE {table} SET '

sql += ', '.join(i for i in new\_columns) # Не меняем кавычки

sql += f' WHERE {' AND '.join(i for i in new\_where)};'

print(sql)

if connection:

with connection.cursor() as cursor:

try:

cursor.execute("BEGIN;")

cursor.execute(sql)

connection.commit()

details = f"item {', '.join(new\_where)} new value {', '.join(new\_columns)}"

log\_action(login, "update", table, details)

return True

except ps2.errors.InsufficientPrivilege as e:

print(e)

connection.rollback()

return no\_privilege(table)

except ps2.errors.InFailedSqlTransaction as e:

print(e)

connection.rollback()

return transaction\_error()

except ps2.errors.UndefinedColumn as e:

print(e)

connection.rollback()

return no\_data()

except Exception as e:

print(e)

connection.rollback()

return error(e)

def delete(table, where=None):

if not where:

return 'Удаление приостановлено. Нет информации для удаления.'

sql = f"DELETE FROM {table}"

if where != 'all':

new\_where = []

for i in where:

if (i.split("=")[1] != " ") and (i.split(" = ")[1] != "''"):

new\_where.append(i)

conditions = ' AND '.join(new\_where)

sql += f" WHERE {conditions}"

sql += ";"

if connection:

with connection.cursor() as cursor:

try:

cursor.execute("BEGIN;")

cursor.execute(sql)

connection.commit()

details = f"deleted where {', '.join(new\_where)}"

log\_action(login, "delete", table, details)

return True

except ps2.errors.InsufficientPrivilege as e:

print(e)

connection.rollback()

return no\_privilege(table)

except ps2.errors.ForeignKeyViolation as e:

print(e)

return no\_privilege(table)

except ps2.errors.InFailedSqlTransaction as e:

print(e)

connection.rollback()

return transaction\_error()

except ps2.errors.UndefinedColumn as e:

print(e)

connection.rollback()

return no\_data()

except Exception as e:

print(e)

connection.rollback()

return error(e)

**Приложение Г Исходный код базы данных склада запчастей**

***Листинг 1 – Исходный код базы данных склада запчастей***

SET statement\_timeout = 0;

SET lock\_timeout = 0;

SET idle\_in\_transaction\_session\_timeout = 0;

SET transaction\_timeout = 0;

SET client\_encoding = 'UTF8';

SET standard\_conforming\_strings = on;

SELECT pg\_catalog.set\_config('search\_path', '', false);

SET check\_function\_bodies = false;

SET xmloption = content;

SET client\_min\_messages = warning;

SET row\_security = off;

CREATE FUNCTION public.convert\_text\_to\_boolean(text\_value text, field\_type text DEFAULT 'status'::text) RETURNS boolean

LANGUAGE plpgsql IMMUTABLE

AS $$

BEGIN

text\_value := LOWER(TRIM(text\_value));

IF field\_type = 'type' THEN

RETURN text\_value IN ('выгрузка', 'выгрузить', 'отправка', 'true', '1', 'да', 'yes', 'y');

ELSE

RETURN text\_value IN ('завершено', 'готово', 'выполнено', 'done', 'true', '1', 'да', 'yes', 'y');

END IF;

END;

$$;

ALTER FUNCTION public.convert\_text\_to\_boolean(text\_value text, field\_type text) OWNER TO postgres;

CREATE FUNCTION public.delete\_invoice\_details\_view() RETURNS trigger

LANGUAGE plpgsql SECURITY DEFINER

AS $$

BEGIN

DELETE FROM invoice\_employee WHERE invoiceID = OLD.invoice\_id;

DELETE FROM invoice\_detail WHERE invoiceID = OLD.invoice\_id;

DELETE FROM invoice WHERE invoice\_id = OLD.invoice\_id;

RETURN OLD;

END;

$$;

ALTER FUNCTION public.delete\_invoice\_details\_view() OWNER TO postgres;

CREATE FUNCTION public.delete\_related\_data() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

DELETE 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 shelf WHERE rackID IN (SELECT rack\_id FROM rack WHERE roomID IN (SELECT room\_id FROM room WHERE warehouseID = OLD.warehouse\_id));

DELETE 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)));

RETURN OLD;

END;

$$;

ALTER FUNCTION public.delete\_related\_data() OWNER TO postgres;

CREATE FUNCTION public.delete\_warehouse\_details() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE

warehouse\_id\_value integer;

room\_id\_value integer;

rack\_id\_value integer;

shelf\_id\_value integer;

BEGIN

SELECT w.warehouse\_id INTO warehouse\_id\_value

FROM warehouse w

WHERE w.warehouse\_number = OLD.warehouse\_number;

SELECT r.room\_id INTO room\_id\_value

FROM room r

WHERE r.warehouseID = warehouse\_id\_value AND r.room\_number = OLD.room\_number;

SELECT ra.rack\_id INTO rack\_id\_value

FROM rack ra

WHERE ra.roomID = room\_id\_value AND ra.rack\_number = OLD.rack\_number;

SELECT s.shelf\_id INTO shelf\_id\_value

FROM shelf s

WHERE s.rackID = rack\_id\_value AND s.shelf\_number = OLD.shelf\_number;

DELETE FROM details WHERE detail\_id = OLD.detail\_id;

DELETE FROM shelf

WHERE shelf.shelf\_id = shelf\_id\_value

AND NOT EXISTS (SELECT 1 FROM details WHERE details.shelfID = shelf.shelf\_id);

DELETE FROM rack

WHERE rack.rack\_id = rack\_id\_value

AND NOT EXISTS (SELECT 1 FROM shelf WHERE shelf.rackID = rack.rack\_id);

DELETE FROM room

WHERE room.room\_id = room\_id\_value

AND NOT EXISTS (SELECT 1 FROM rack WHERE rack.roomID = room.room\_id);

DELETE FROM warehouse

WHERE warehouse.warehouse\_id = warehouse\_id\_value

AND NOT EXISTS (SELECT 1 FROM room WHERE room.warehouseID = warehouse.warehouse\_id);

RETURN OLD;

END;

$$;

ALTER FUNCTION public.delete\_warehouse\_details() OWNER TO postgres;

CREATE FUNCTION public.get\_employee\_id(p\_last\_name character varying, p\_first\_name character varying, p\_patronymic character varying) RETURNS integer

LANGUAGE plpgsql SECURITY DEFINER

AS $$

DECLARE

v\_id integer;

BEGIN

SELECT employee\_id INTO v\_id

FROM employee

WHERE last\_name = p\_last\_name

AND first\_name = p\_first\_name

AND patronymic = p\_patronymic;

IF NOT FOUND THEN

RAISE EXCEPTION 'Сотрудник % % % не найден',

p\_last\_name, p\_first\_name, p\_patronymic;

END IF;

RETURN v\_id;

END;

$$;

ALTER FUNCTION public.get\_employee\_id(p\_last\_name character varying, p\_first\_name character varying, p\_patronymic character varying) OWNER TO postgres;

CREATE FUNCTION public.insert\_into\_warehouse\_details() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE

warehouse\_id integer;

room\_id integer;

rack\_id integer;

shelf\_id integer;

BEGIN

IF NOT EXISTS (SELECT 1 FROM warehouse WHERE warehouse\_number = NEW.warehouse\_number) THEN

INSERT INTO warehouse (warehouse\_number, address)

VALUES (NEW.warehouse\_number, 'default address')

RETURNING warehouse.warehouse\_id INTO warehouse\_id;

ELSE

SELECT warehouse.warehouse\_id INTO warehouse\_id FROM warehouse WHERE warehouse\_number = NEW.warehouse\_number;

END IF;

IF NOT EXISTS (SELECT 1 FROM room WHERE room\_number = NEW.room\_number AND warehouseID = warehouse\_id) THEN

INSERT INTO room (room\_number, warehouseID)

VALUES (NEW.room\_number, warehouse\_id)

RETURNING room.room\_id INTO room\_id;

ELSE

SELECT room.room\_id INTO room\_id FROM room WHERE room\_number = NEW.room\_number AND warehouseID = warehouse\_id;

END IF;

IF NOT EXISTS (SELECT 1 FROM rack WHERE rack\_number = NEW.rack\_number AND roomID = room\_id) THEN

INSERT INTO rack (rack\_number, roomID)

VALUES (NEW.rack\_number, room\_id)

RETURNING rack.rack\_id INTO rack\_id;

ELSE

SELECT rack.rack\_id INTO rack\_id FROM rack WHERE rack\_number = NEW.rack\_number AND roomID = room\_id;

END IF;

IF NOT EXISTS (SELECT 1 FROM shelf WHERE shelf\_number = NEW.shelf\_number AND rackID = rack\_id) THEN

INSERT INTO shelf (shelf\_number, rackID)

VALUES (NEW.shelf\_number, rack\_id)

RETURNING shelf.shelf\_id INTO shelf\_id;

ELSE

SELECT shelf.shelf\_id INTO shelf\_id FROM shelf WHERE shelf\_number = NEW.shelf\_number AND rackID = rack\_id;

END IF;

INSERT INTO details (shelfID, weight, type\_detail)

VALUES (shelf\_id, NEW.weight, NEW.type\_detail);

RETURN NEW;

END;

$$;

ALTER FUNCTION public.insert\_into\_warehouse\_details() OWNER TO postgres;

CREATE FUNCTION public.insert\_invoice\_details\_view() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE

v\_invoice\_id INTEGER;

v\_counteragent\_id INTEGER;

v\_detail\_id INTEGER;

v\_employee\_id INTEGER;

v\_type\_invoice BOOLEAN;

v\_status BOOLEAN;

BEGIN

SELECT counteragent\_id INTO v\_counteragent\_id

FROM counteragent

WHERE counteragent\_name = NEW.counteragent\_name;

IF NOT FOUND THEN

RAISE EXCEPTION 'Контрагент с именем % не найден', NEW.counteragent\_name;

END IF;

IF NEW.type\_invoice\_text IS NOT NULL THEN

v\_type\_invoice := convert\_text\_to\_boolean(NEW.type\_invoice\_text, 'type');

ELSE

v\_type\_invoice := COALESCE(NEW.type\_invoice\_bool, FALSE);

END IF;

IF NEW.status\_text IS NOT NULL THEN

v\_status := convert\_text\_to\_boolean(NEW.status\_text, 'status');

ELSE

v\_status := COALESCE(NEW.status\_bool, FALSE);

END IF;

IF NEW.invoice\_id IS NOT NULL THEN

PERFORM 1 FROM invoice WHERE invoice\_id = NEW.invoice\_id;

IF FOUND THEN

UPDATE invoice SET

counteragentID = v\_counteragent\_id,

date\_time = NEW.date\_time,

type\_invoice = v\_type\_invoice,

status = v\_status

WHERE invoice\_id = NEW.invoice\_id;

v\_invoice\_id := NEW.invoice\_id;

ELSE

INSERT INTO invoice (invoice\_id, counteragentID, date\_time, type\_invoice, status)

VALUES (NEW.invoice\_id, v\_counteragent\_id, NEW.date\_time, v\_type\_invoice, v\_status)

RETURNING invoice\_id INTO v\_invoice\_id;

END IF;

ELSE

INSERT INTO invoice (counteragentID, date\_time, type\_invoice, status)

VALUES (v\_counteragent\_id, NEW.date\_time, v\_type\_invoice, v\_status)

RETURNING invoice\_id INTO v\_invoice\_id;

END IF;

SELECT detail\_id INTO v\_detail\_id

FROM details

WHERE type\_detail = NEW.type\_detail;

IF NOT FOUND THEN

RAISE EXCEPTION 'Деталь типа % не найдена', NEW.type\_detail;

END IF;

INSERT INTO invoice\_detail (invoiceID, detailID, quantity)

VALUES (v\_invoice\_id, v\_detail\_id, NEW.quantity)

ON CONFLICT (invoiceID, detailID)

DO UPDATE SET quantity = invoice\_detail.quantity + NEW.quantity;

v\_employee\_id := get\_employee\_id(

NEW.responsible\_last\_name,

NEW.responsible\_first\_name,

NEW.responsible\_patronymic

);

INSERT INTO invoice\_employee (invoiceID, responsible, granted\_access, when\_granted)

VALUES (v\_invoice\_id, v\_employee\_id, v\_employee\_id, NOW())

ON CONFLICT (invoiceID, responsible) DO NOTHING;

NEW.invoice\_id := v\_invoice\_id;

RETURN NEW;

END;

$$;

ALTER FUNCTION public.insert\_invoice\_details\_view() OWNER TO postgres;

CREATE FUNCTION public.log\_counteragent\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('counteragent', 'INSERT', NEW.counteragent\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('counteragent', 'UPDATE', OLD.counteragent\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('counteragent', 'DELETE', OLD.counteragent\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_counteragent\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_details\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('details', 'INSERT', NEW.detail\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('details', 'UPDATE', OLD.detail\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('details', 'DELETE', OLD.detail\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_details\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_employee\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('employee', 'INSERT', NEW.employee\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('employee', 'UPDATE', OLD.employee\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('employee', 'DELETE', OLD.employee\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_employee\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_invoice\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('invoice', 'INSERT', NEW.invoice\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('invoice', 'UPDATE', OLD.invoice\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('invoice', 'DELETE', OLD.invoice\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_invoice\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_invoice\_detail\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('invoice\_detail', 'INSERT', NEW.invoiceID, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('invoice\_detail', 'UPDATE', OLD.invoiceID, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('invoice\_detail', 'DELETE', OLD.invoiceID, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_invoice\_detail\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_invoice\_employee\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('invoice\_employee', 'INSERT', NEW.invoiceID, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('invoice\_employee', 'UPDATE', OLD.invoiceID, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('invoice\_employee', 'DELETE', OLD.invoiceID, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_invoice\_employee\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_rack\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('rack', 'INSERT', NEW.rack\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('rack', 'UPDATE', OLD.rack\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('rack', 'DELETE', OLD.rack\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_rack\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_room\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('room', 'INSERT', NEW.room\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('room', 'UPDATE', OLD.room\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('room', 'DELETE', OLD.room\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_room\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_shelf\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('shelf', 'INSERT', NEW.shelf\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('shelf', 'UPDATE', OLD.shelf\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('shelf', 'DELETE', OLD.shelf\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_shelf\_changes() OWNER TO postgres;

CREATE FUNCTION public.log\_warehouse\_changes() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

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

VALUES ('warehouse', 'INSERT', NEW.warehouse\_id, to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

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

VALUES ('warehouse', 'UPDATE', OLD.warehouse\_id, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

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

VALUES ('warehouse', 'DELETE', OLD.warehouse\_id, to\_jsonb(OLD));

RETURN OLD;

END IF;

END;

$$;

ALTER FUNCTION public.log\_warehouse\_changes() OWNER TO postgres;

CREATE FUNCTION public.update\_invoice\_details\_view() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE

v\_updated BOOLEAN := FALSE;

BEGIN

IF NEW.type\_invoice\_text IS DISTINCT FROM OLD.type\_invoice\_text THEN

UPDATE invoice SET

type\_invoice = convert\_text\_to\_boolean(NEW.type\_invoice\_text, 'type')

WHERE invoice\_id = NEW.invoice\_id;

v\_updated := TRUE;

END IF;

IF NEW.status\_text IS DISTINCT FROM OLD.status\_text THEN

UPDATE invoice SET

status = convert\_text\_to\_boolean(NEW.status\_text, 'status')

WHERE invoice\_id = NEW.invoice\_id;

v\_updated := TRUE;

END IF;

IF NOT v\_updated AND (

NEW.invoice\_id IS DISTINCT FROM OLD.invoice\_id OR

NEW.counteragent\_name IS DISTINCT FROM OLD.counteragent\_name OR

NEW.date\_time IS DISTINCT FROM OLD.date\_time OR

NEW.type\_detail IS DISTINCT FROM OLD.type\_detail OR

NEW.quantity IS DISTINCT FROM OLD.quantity OR

NEW.responsible\_last\_name IS DISTINCT FROM OLD.responsible\_last\_name OR

NEW.responsible\_first\_name IS DISTINCT FROM OLD.responsible\_first\_name OR

NEW.responsible\_patronymic IS DISTINCT FROM OLD.responsible\_patronymic

) THEN

RAISE EXCEPTION 'Разрешено обновлять только поля type\_invoice\_text и status\_text';

END IF;

RETURN NEW;

END;

$$;

ALTER FUNCTION public.update\_invoice\_details\_view() OWNER TO postgres;

CREATE FUNCTION public.update\_invoice\_status() RETURNS trigger

LANGUAGE plpgsql SECURITY DEFINER

AS $$

BEGIN

IF TG\_OP = 'UPDATE' AND (

OLD.invoice\_id IS DISTINCT FROM NEW.invoice\_id OR

OLD.counteragent\_name IS DISTINCT FROM NEW.counteragent\_name OR

OLD.date\_time IS DISTINCT FROM NEW.date\_time OR

OLD.type\_invoice IS DISTINCT FROM NEW.type\_invoice OR

OLD.type\_detail IS DISTINCT FROM NEW.type\_detail OR

OLD.quantity IS DISTINCT FROM NEW.quantity OR

OLD.responsible\_last\_name IS DISTINCT FROM NEW.responsible\_last\_name OR

OLD.responsible\_first\_name IS DISTINCT FROM NEW.responsible\_first\_name OR

OLD.responsible\_patronymic IS DISTINCT FROM NEW.responsible\_patronymic OR

OLD.responsible\_id IS DISTINCT FROM NEW.responsible\_id

) THEN

RAISE EXCEPTION 'Разрешено обновлять только поле status. Попытка изменить другие поля запрещена.';

END IF;

IF OLD.status IS NOT DISTINCT FROM NEW.status THEN

RETURN NEW;

END IF;

UPDATE invoice SET status = NEW.status

WHERE invoice\_id = NEW.invoice\_id;

RETURN NEW;

END;

$$;

ALTER FUNCTION public.update\_invoice\_status() OWNER TO postgres;

CREATE FUNCTION public.update\_warehouse\_details\_view() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE details

SET type\_detail = NEW.type\_detail, weight = NEW.weight

WHERE detail\_id = OLD.detail\_id;

UPDATE shelf

SET shelf\_number = NEW.shelf\_number

WHERE shelf\_id = (SELECT shelfID FROM details WHERE detail\_id = OLD.detail\_id);

UPDATE rack

SET rack\_number = NEW.rack\_number

WHERE rack\_id = (SELECT rackID FROM shelf WHERE shelf\_id = (SELECT shelfID FROM details WHERE detail\_id = OLD.detail\_id));

UPDATE room

SET room\_number = NEW.room\_number

WHERE room\_id = (SELECT roomID FROM rack WHERE rack\_id = (SELECT rackID FROM shelf WHERE shelf\_id = (SELECT shelfID FROM details WHERE detail\_id = OLD.detail\_id)));

UPDATE warehouse

SET warehouse\_number = NEW.warehouse\_number

WHERE warehouse\_id = (SELECT warehouseID FROM room WHERE room\_id = (SELECT roomID FROM rack WHERE rack\_id = (SELECT rackID FROM shelf WHERE shelf\_id = (SELECT shelfID FROM details WHERE detail\_id = OLD.detail\_id)));

RETURN NEW;

END;

$$;

ALTER FUNCTION public.update\_warehouse\_details\_view() OWNER TO postgres;

SET default\_tablespace = '';

SET default\_table\_access\_method = heap;

CREATE TABLE public.counteragent (

counteragent\_id integer NOT NULL,

counteragent\_name character varying(128) NOT NULL,

contact\_person character varying(128) NOT NULL,

phone\_number bigint NOT NULL,

address text NOT NULL

);

ALTER TABLE public.counteragent OWNER TO postgres;

CREATE SEQUENCE public.counteragent\_counteragent\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.counteragent\_counteragent\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.counteragent\_counteragent\_id\_seq OWNED BY public.counteragent.counteragent\_id;

CREATE TABLE public.details (

detail\_id integer NOT NULL,

shelfid integer NOT NULL,

weight double precision NOT NULL,

type\_detail text NOT NULL

);

ALTER TABLE public.details OWNER TO postgres;

CREATE SEQUENCE public.details\_detail\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.details\_detail\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.details\_detail\_id\_seq OWNED BY public.details.detail\_id;

CREATE SEQUENCE public.details\_shelfid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.details\_shelfid\_seq OWNER TO postgres;

ALTER SEQUENCE public.details\_shelfid\_seq OWNED BY public.details.shelfid;

CREATE TABLE public.employee (

employee\_id integer NOT NULL,

employee\_role character varying(25) NOT NULL,

last\_name character varying(35) NOT NULL,

first\_name character varying(35) NOT NULL,

patronymic character varying(35) NOT NULL

);

ALTER TABLE public.employee OWNER TO postgres;

CREATE SEQUENCE public.employee\_employee\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.employee\_employee\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.employee\_employee\_id\_seq OWNED BY public.employee.employee\_id;

CREATE TABLE public.invoice (

invoice\_id integer NOT NULL,

counteragentid integer NOT NULL,

date\_time timestamp without time zone NOT NULL,

type\_invoice boolean NOT NULL,

status boolean NOT NULL

);

ALTER TABLE public.invoice OWNER TO postgres;

CREATE SEQUENCE public.invoice\_counteragentid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_counteragentid\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_counteragentid\_seq OWNED BY public.invoice.counteragentid;

CREATE TABLE public.invoice\_detail (

invoiceid integer NOT NULL,

detailid integer NOT NULL,

quantity integer NOT NULL

);

ALTER TABLE public.invoice\_detail OWNER TO postgres;

CREATE SEQUENCE public.invoice\_detail\_detailid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_detail\_detailid\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_detail\_detailid\_seq OWNED BY public.invoice\_detail.detailid;

CREATE SEQUENCE public.invoice\_detail\_invoiceid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_detail\_invoiceid\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_detail\_invoiceid\_seq OWNED BY public.invoice\_detail.invoiceid;

CREATE TABLE public.invoice\_employee (

invoiceid integer NOT NULL,

responsible integer NOT NULL,

granted\_access integer NOT NULL,

when\_granted timestamp without time zone NOT NULL

);

ALTER TABLE public.invoice\_employee OWNER TO postgres;

CREATE VIEW public.invoice\_details\_view AS

SELECT inv.invoice\_id,

ca.counteragent\_name,

inv.date\_time,

CASE

WHEN inv.type\_invoice THEN 'Выгрузка'::text

ELSE 'Отгрузка'::text

END AS type\_invoice\_text,

CASE

WHEN inv.status THEN 'Завершено'::text

ELSE 'В процессе'::text

END AS status\_text,

det.type\_detail,

invd.quantity,

emp.last\_name AS responsible\_last\_name,

emp.first\_name AS responsible\_first\_name,

emp.patronymic AS responsible\_patronymic,

emp.employee\_id AS responsible\_id,

inv.status AS status\_bool,

inv.type\_invoice AS type\_invoice\_bool

FROM (((((public.invoice inv

JOIN public.invoice\_detail invd ON ((inv.invoice\_id = invd.invoiceid)))

JOIN public.details det ON ((invd.detailid = det.detail\_id)))

JOIN public.invoice\_employee inv\_emp ON ((inv.invoice\_id = inv\_emp.invoiceid)))

JOIN public.employee emp ON ((inv\_emp.responsible = emp.employee\_id)))

JOIN public.counteragent ca ON ((inv.counteragentid = ca.counteragent\_id)));

ALTER VIEW public.invoice\_details\_view OWNER TO postgres;

CREATE SEQUENCE public.invoice\_employee\_granted\_access\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_employee\_granted\_access\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_employee\_granted\_access\_seq OWNED BY public.invoice\_employee.granted\_access;

CREATE SEQUENCE public.invoice\_employee\_invoiceid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_employee\_invoiceid\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_employee\_invoiceid\_seq OWNED BY public.invoice\_employee.invoiceid;

CREATE SEQUENCE public.invoice\_employee\_responsible\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_employee\_responsible\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_employee\_responsible\_seq OWNED BY public.invoice\_employee.responsible;

CREATE SEQUENCE public.invoice\_invoice\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.invoice\_invoice\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.invoice\_invoice\_id\_seq OWNED BY public.invoice.invoice\_id;

CREATE TABLE public.log\_table (

log\_id integer NOT NULL,

table\_name text NOT NULL,

action\_type text NOT NULL,

record\_id integer NOT NULL,

action\_time timestamp without time zone DEFAULT CURRENT\_TIMESTAMP,

old\_values jsonb,

new\_values jsonb

);

ALTER TABLE public.log\_table OWNER TO postgres;

CREATE SEQUENCE public.log\_table\_log\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.log\_table\_log\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.log\_table\_log\_id\_seq OWNED BY public.log\_table.log\_id;

CREATE TABLE public.rack (

rack\_id integer NOT NULL,

roomid integer NOT NULL,

rack\_number integer NOT NULL

);

ALTER TABLE public.rack OWNER TO postgres;

CREATE SEQUENCE public.rack\_rack\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.rack\_rack\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.rack\_rack\_id\_seq OWNED BY public.rack.rack\_id;

CREATE SEQUENCE public.rack\_roomid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.rack\_roomid\_seq OWNER TO postgres;

ALTER SEQUENCE public.rack\_roomid\_seq OWNED BY public.rack.roomid;

CREATE TABLE public.room (

room\_id integer NOT NULL,

warehouseid integer NOT NULL,

room\_number integer NOT NULL

);

ALTER TABLE public.room OWNER TO postgres;

CREATE SEQUENCE public.room\_room\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.room\_room\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.room\_room\_id\_seq OWNED BY public.room.room\_id;

CREATE SEQUENCE public.room\_warehouseid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.room\_warehouseid\_seq OWNER TO postgres;

ALTER SEQUENCE public.room\_warehouseid\_seq OWNED BY public.room.warehouseid;

CREATE TABLE public.shelf (

shelf\_id integer NOT NULL,

rackid integer NOT NULL,

shelf\_number integer NOT NULL

);

ALTER TABLE public.shelf OWNER TO postgres;

CREATE SEQUENCE public.shelf\_rackid\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.shelf\_rackid\_seq OWNER TO postgres;

ALTER SEQUENCE public.shelf\_rackid\_seq OWNED BY public.shelf.rackid;

CREATE SEQUENCE public.shelf\_shelf\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.shelf\_shelf\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.shelf\_shelf\_id\_seq OWNED BY public.shelf.shelf\_id;

CREATE TABLE public.warehouse (

warehouse\_id integer NOT NULL,

warehouse\_number integer NOT NULL,

address text NOT NULL

);

ALTER TABLE public.warehouse OWNER TO postgres;

CREATE VIEW public.warehouse\_details\_view AS

SELECT w.warehouse\_number,

r.room\_number,

rk.rack\_number,

s.shelf\_number,

d.type\_detail,

d.weight,

d.detail\_id

FROM ((((public.warehouse w

JOIN public.room r ON ((w.warehouse\_id = r.warehouseid)))

JOIN public.rack rk ON ((r.room\_id = rk.roomid)))

JOIN public.shelf s ON ((rk.rack\_id = s.rackid)))

JOIN public.details d ON ((s.shelf\_id = d.shelfid)));

ALTER VIEW public.warehouse\_details\_view OWNER TO postgres;

CREATE SEQUENCE public.warehouse\_warehouse\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER SEQUENCE public.warehouse\_warehouse\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.warehouse\_warehouse\_id\_seq OWNED BY public.warehouse.warehouse\_id;

ALTER TABLE ONLY public.counteragent ALTER COLUMN counteragent\_id SET DEFAULT nextval('public.counteragent\_counteragent\_id\_seq'::regclass);

ALTER TABLE ONLY public.details ALTER COLUMN detail\_id SET DEFAULT nextval('public.details\_detail\_id\_seq'::regclass);

ALTER TABLE ONLY public.details ALTER COLUMN shelfid SET DEFAULT nextval('public.details\_shelfid\_seq'::regclass);

ALTER TABLE ONLY public.employee ALTER COLUMN employee\_id SET DEFAULT nextval('public.employee\_employee\_id\_seq'::regclass);

ALTER TABLE ONLY public.invoice ALTER COLUMN invoice\_id SET DEFAULT nextval('public.invoice\_invoice\_id\_seq'::regclass);

ALTER TABLE ONLY public.invoice ALTER COLUMN counteragentid SET DEFAULT nextval('public.invoice\_counteragentid\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_detail ALTER COLUMN invoiceid SET DEFAULT nextval('public.invoice\_detail\_invoiceid\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_detail ALTER COLUMN detailid SET DEFAULT nextval('public.invoice\_detail\_detailid\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_employee ALTER COLUMN invoiceid SET DEFAULT nextval('public.invoice\_employee\_invoiceid\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_employee ALTER COLUMN responsible SET DEFAULT nextval('public.invoice\_employee\_responsible\_seq'::regclass);

ALTER TABLE ONLY public.invoice\_employee ALTER COLUMN granted\_access SET DEFAULT nextval('public.invoice\_employee\_granted\_access\_seq'::regclass);

ALTER TABLE ONLY public.log\_table ALTER COLUMN log\_id SET DEFAULT nextval('public.log\_table\_log\_id\_seq'::regclass);

ALTER TABLE ONLY public.rack ALTER COLUMN rack\_id SET DEFAULT nextval('public.rack\_rack\_id\_seq'::regclass);

ALTER TABLE ONLY public.rack ALTER COLUMN roomid SET DEFAULT nextval('public.rack\_roomid\_seq'::regclass);

ALTER TABLE ONLY public.room ALTER COLUMN room\_id SET DEFAULT nextval('public.room\_room\_id\_seq'::regclass);

ALTER TABLE ONLY public.room ALTER COLUMN warehouseid SET DEFAULT nextval('public.room\_warehouseid\_seq'::regclass);

ALTER TABLE ONLY public.shelf ALTER COLUMN shelf\_id SET DEFAULT nextval('public.shelf\_shelf\_id\_seq'::regclass);

ALTER TABLE ONLY public.shelf ALTER COLUMN rackid SET DEFAULT nextval('public.shelf\_rackid\_seq'::regclass);

ALTER TABLE ONLY public.warehouse ALTER COLUMN warehouse\_id SET DEFAULT nextval('public.warehouse\_warehouse\_id\_seq'::regclass);

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 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\_warehouse\_details\_trigger INSTEAD OF DELETE ON public.warehouse\_details\_view FOR EACH ROW EXECUTE FUNCTION public.delete\_warehouse\_details();

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 instead\_of\_update\_warehouse\_details\_view INSTEAD OF UPDATE ON public.warehouse\_details\_view FOR EACH ROW EXECUTE FUNCTION public.update\_warehouse\_details\_view();

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 trigger\_delete\_warehouse\_details INSTEAD OF DELETE ON public.warehouse\_details\_view FOR EACH ROW EXECUTE FUNCTION public.delete\_warehouse\_details();

CREATE TRIGGER trigger\_insert\_warehouse\_details INSTEAD OF INSERT ON public.warehouse\_details\_view FOR EACH ROW EXECUTE FUNCTION public.insert\_into\_warehouse\_details();

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 ON TABLE public.counteragent TO warehouse\_manager;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.counteragent TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.counteragent\_counteragent\_id\_seq 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 ON TABLE public.details TO warehouse\_owner;

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,UPDATE ON TABLE public.invoice TO warehouse\_clerk;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.invoice TO warehouse\_manager;

GRANT SELECT ON TABLE public.invoice TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_counteragentid\_seq 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\_owner;

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,INSERT,DELETE,UPDATE ON TABLE public.invoice\_employee TO warehouse\_manager;

GRANT SELECT ON TABLE public.invoice\_employee TO warehouse\_owner;

GRANT SELECT ON TABLE public.invoice\_employee TO warehouse\_clerk;

GRANT SELECT ON TABLE public.invoice\_details\_view TO warehouse\_owner;

GRANT SELECT,UPDATE ON TABLE public.invoice\_details\_view TO warehouse\_clerk;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.invoice\_details\_view TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_employee\_granted\_access\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_employee\_invoiceid\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_employee\_responsible\_seq TO warehouse\_owner;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_invoice\_id\_seq TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.invoice\_invoice\_id\_seq TO warehouse\_owner;

GRANT SELECT,INSERT ON TABLE public.log\_table TO warehouse\_owner;

GRANT SELECT,INSERT ON TABLE public.log\_table TO warehouse\_clerk;

GRANT SELECT,INSERT ON TABLE public.log\_table TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.log\_table\_log\_id\_seq TO warehouse\_clerk;

GRANT SELECT,USAGE ON SEQUENCE public.log\_table\_log\_id\_seq TO warehouse\_manager;

GRANT SELECT,USAGE ON SEQUENCE public.log\_table\_log\_id\_seq TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE public.rack TO warehouse\_clerk;

GRANT SELECT ON TABLE public.rack TO warehouse\_owner;

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\_clerk;

GRANT SELECT ON TABLE public.room TO warehouse\_owner;

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\_clerk;

GRANT SELECT ON TABLE public.shelf TO warehouse\_owner;

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\_clerk;

GRANT SELECT ON TABLE public.warehouse TO warehouse\_owner;

GRANT SELECT ON TABLE public.warehouse\_details\_view TO warehouse\_owner;

GRANT SELECT,INSERT,DELETE,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;