*Федеральное государственное бюджетное образовательное учреждение* *высшего образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(национальный исследовательский университет)***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ \_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_Компьютерные Системы и сети (ИУ6)\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬАНЯ ЗАПИСКА**

**К курсовой работе**

**Название:** База данных “Продажа деталей”

**Дисциплина:** Базы данных

Студент гр. ИУ6-45Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**П. И. Шегай**\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_** М.А. Скворцова **\_\_**

(Подпись, дата) (И.О. Фамилия)

Москва, 2025

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_ИУ6\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Пролетарский

(И.О. Фамилия)

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы**

по дисциплине \_\_\_\_Базы данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_\_\_ИУ6-45Б\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шегай Павел Игоревич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Тема курсовой работы \_\_\_\_\_\_\_\_\_Продажа деталей\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направленность КР (учебная, исследовательская, практическая, производственная, др.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_учебная\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_\_\_\_\_\_ИУ6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения КР: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Техническое задание

Необходимо разработать базу данных «Продажа деталей», содержащую не менее 7 связных таблиц. Основная сущность должна содержать не менее 1 млн. записей, остальные не менее 100 записей. Разработать инфологическую и даталогическую модель базы данных. В базе данных должно быть разработано не менее 7 сложных/вложенных запросов. В одном из запросов реализовать возможность его формирования по условию преподавателя.

***Оформление курсовой работы:***

Расчетно-пояснительная записка (РПЗ) на не менее 25 листах формата А4.

Дата выдачи задания «07» февраля 2025 г.

**Руководитель курсовой работы**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_М.А. Скворцова\_\_\_

(Подпись, дата) (И.О. Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**П.И. Шегай**\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

**РЕФЕРАТ**

Расчетно-пояснительная записка 60 страниц, 15 рисунков, 8 таблиц.

Объектом разработки является база данных для учета продаж деталей строительного и промышленного назначения.

Цель работы – создание автоматизированной системы учета со следующими функциями:

* Управление складскими запасами;
* Формирование накладных;
* Анализ продаж.

Поставленная цель достигается за счет СУБД PostgreSQL версии 16.3, средства проектирования pgAdmin 4 версии 8.9, языка программирования Python версии 3.12.2 и среды разработки Microsoft Visual Studio Code версии 1.88.1. Разрабатываемая база данных содержит 7 таблиц, взаимодействующих друг с другом при помощи связей.

Ключевые слова – база данных, индексы, PostgreSQL, Python, SQL-запросы, таблицы, триггеры, записи, поля, Telegram.

**СОДЕРЖАНИЕ**

Введение5

1 Анализ предметной области6

2 Разработка бизнес-процессов7

3 Проектирование базы данных8

3.1 Выделение главной сущности8

3.2 Описание таблиц8

3.4 Схема базы данных12

4 Заполнение таблиц13

4.1 Создание базы данных13

4.2 Заполнение таблиц13

5 Разработка запросов14

6 Разработка интерфейса23

Заключение 28

Приложение А 29

Приложение Б 32

Приложение В42

Приложение Г 44

Приложение Д58

**ВВЕДЕНИЕ**

В данной работе создается является база данных для учета продаж деталей строительного и промышленного назначения. Она предназначена для управления складскими запасами, формирования накладных и анализа продаж. Интерфейс предоставляет пользователю возможность быстрого и удобного формирования соответствующих запросов в базу данных.

Актуальность разработки обусловлена ростом спроса на промышленные комплектующие и необходимостью перехода от ручных методов учета к автоматизированным решениям. Внедрение данной системы позволяет:

* Оптимизировать складской учет;
* Снизить количество ошибок при оформлении документов;
* Ускорить процесс обработки заказов;
* Улучшить контроль за платежами и задолженностями;
* Получать аналитические данные для принятия управленческих решений.

Интерфейсная часть реализована в виде Telegram-бота и отвечает следующим требованиям:

* Простота и понятность;
* Кроссплатформенная доступность;
* Быстрый доступ к ключевым операциям.

Особое внимание уделено автоматизации рутинных операций и минимизации человеческого фактора при работе с данными, что в совокупности повышает эффективность бизнес-процессов компании.

**1 Анализ предметной области**

Предметной областью разрабатываемого продукта является формирование накладных.

Разрабатываемая база данных "Детали для строительных и производственных компаний" предназначена для автоматизации учета и управления продажами промышленных комплектующих.

База данных выполняет следующие функции:

1) Хранение информации о поставщиках;

2) Хранение информации о деталях;

3) Хранение информации о сотрудниках;

4) Хранение информации о покупателях.

Данные требования к разрабатываемой базе данных были учтены в дальнейшем при создании модели, а также доработаны в соответствии с возникающими проблемами при проектировании.

После рассмотрения предметной области было решено создать 8 таблиц: “Деталь”, “Тип детали”, “Поставщик”, “Покупатель”, “Сотрудник”, “Накладная”, “Строка накладной”, “Платеж”.

****Схема связи таблиц представлена на рисунке 1.

Рисунок 1 - Схема связи

**2 Разработка бизнес-процессов**

Основные бизнес-процессы:

1) Управление складскими запасами

Описание: Контроль наличия деталей на складе, пополнение запасов при достижении минимального уровня.

Участники: Складской работник, менеджер по закупкам.

События:

* Достижение минимального уровня запасов;
* Формирование заявки на пополнение;
* Получение деталей от поставщика.

Результат: Обновление данных о количестве деталей на складе.

2) Формирование накладных

Описание: Оформление накладной на продажу деталей покупателю.

Участники: Менеджер по продажам, покупатель.

События:

* Создание накладной;
* Добавление строк накладной с указанием деталей и их количества;
* Проверка наличия деталей на складе.

Результат: Накладная с статусом "Оформлена".

3) Обработка платежей

Описание: Учет оплат по накладным, обновление статуса оплаты.

Участники: Бухгалтер, покупатель.

События:

* Поступление платежа;
* Проверка соответствия суммы платежа сумме накладной;
* Обновление статуса накладной.

Результат: Накладная с актуальным статусом оплаты.

4) Анализ продаж

Описание: Формирование отчетов по продажам, выявление тенденций.

Участники: Аналитик, руководство.

События:

* Запуск отчетов (ежемесячно или по запросу);
* Визуализация данных (графики, таблицы).

Результат: Отчеты для принятия управленческих решений.

**3 Проектирование базы данных**

**3.1 Выделение главной сущности**

В разрабатываемой базе данных можно выделить следующую иерархию:

* Центральная сущность: накладная. Выбор обусловлен предметной областью разрабатываемого продукта;
* Второстепенные сущности: деталь, тип детали, поставщик, покупатель, платеж, строка накладной, сотрудник.

**3.2 Описание таблиц**

Т.к. в базе данных необходимо формирование накладных, были созданы таблицы “Накладная”. Она содержит данные о дате и времени оформления накладной, общей сумме покупки, покупателе из таблицы “Покупатель”, сотруднике, который оформил накладную, из таблицы “Сотрудник” и статусе оплаты (оплачено, не оплачено, частично оплачено).

|  |  |
| --- | --- |
| Название столбца | Тип данных |
| invoice\_id | SERIAL |
| invoice\_date | TIMESTAMP |
| total\_amount | DECIMAL(12,2) |
| customer\_id | SERIAL |
| employee\_id | SERIAL |
| payment\_status | VARCHAR(50) |

Для хранения более подробной информации о накладных была создана таблица “Строка накладной”, содержащая информацию о номере накладной, к которой принадлежит строка, из таблицы “Накладная”, детали из таблицы “Деталь”, количестве, цене за единицу, которая совпадает с соответствующей ценой детали из таблицы “Деталь”, и общей сумме покупки, которая вычисляется как произведение предыдущих двух полей.

|  |  |
| --- | --- |
| Название столбца | Тип данных |
| invoiceline\_id | SERIAL |
| invoice\_id | SERIAL |
| part\_id | INT |
| quantity | INT |
| unit\_price | DECIMAL(10,2) |
| line\_total | DECIMAL(12,2) |

Таблица “Деталь” содержит информацию о материале, весе, цене, типе детали из таблицы “Тип детали”, поставщике из таблицы “Поставщик”, количестве на складе, минимальном необходимом количестве на складе и наличии в продаже. Наличие в продаже определяется в зависимости от того, превышает ли текущее количество деталей на складе минимальное необходимое количество.

|  |  |
| --- | --- |
| Название столбца | Тип данных |
| part\_id | SERIAL |
| material | VARCHAR(50) |
| weight | DECIMAL(10,2) |
| price | INT |
| parttype\_id | SERIAL |
| supplier\_id | SERIAL |
| min\_stock\_level | INT |
| is\_active | BOOLEAN |

Для хранения более подробной информации о типах деталей была создана таблица “Тип детали”. Она содержит информацию о названии типа и описание.

|  |  |
| --- | --- |
| Название столбца | Тип данных |
| parttype\_id | SERIAL |
| type\_name | VARCHAR(100) |
| description | TEXT |

Для хранения информации о поставщиках необходимо наличие таблицы “Поставщик”. Она содержит название компании поставщика, контактный телефон и почту.

|  |  |
| --- | --- |
| Название столбца | Тип данных |
| supplier\_id | SERIAL |
| supplier\_name | VARCHAR(100) |
| contact\_phone | VARCHAR(50) |
| email | VARCHAR(50) |

Аналогично таблице “Поставщик” была создана таблица “Покупатель”, содержащая информацию о названии компании покупателя, городе, контактном телефоне и почте.

|  |  |
| --- | --- |
| Название столбца | Тип данных |
| customer\_id | SERIAL |
| customer\_name | VARCHAR(100) |
| city | VARCHAR(50) |
| contact\_phone | VARCHAR(50) |
| email | VARCHAR(50) |

Также была создана таблица “Сотрудник” с информацией об имени, фамилии и отчестве сотрудника, занимаемой должности, дате найма и возрасте.

|  |  |
| --- | --- |
| Название столбца | Тип данных |
| employee\_id | SERIAL |
| first\_name | VARCHAR(50) |
| second\_name | VARCHAR(50) |
| last\_name | VARCHAR(50) |
| hire\_date | DATE |
| age | int |

Для хранения информации о платежах была создана таблица “Платеж” с информацией о накладной, к которой принадлежит платеж, дате совершения оплаты, сумме и методе оплаты (наличный расчет, безналичный расчет).

|  |  |
| --- | --- |
| Название столбца | Тип данных |
| payment\_id | SERIAL |
| invoice\_id | SERIAL |
| payment\_date | DATE |
| amount | DECIMAL(12,2) |
| payment\_method | VARCHAR(50) |

**3.3 Схема базы данных**

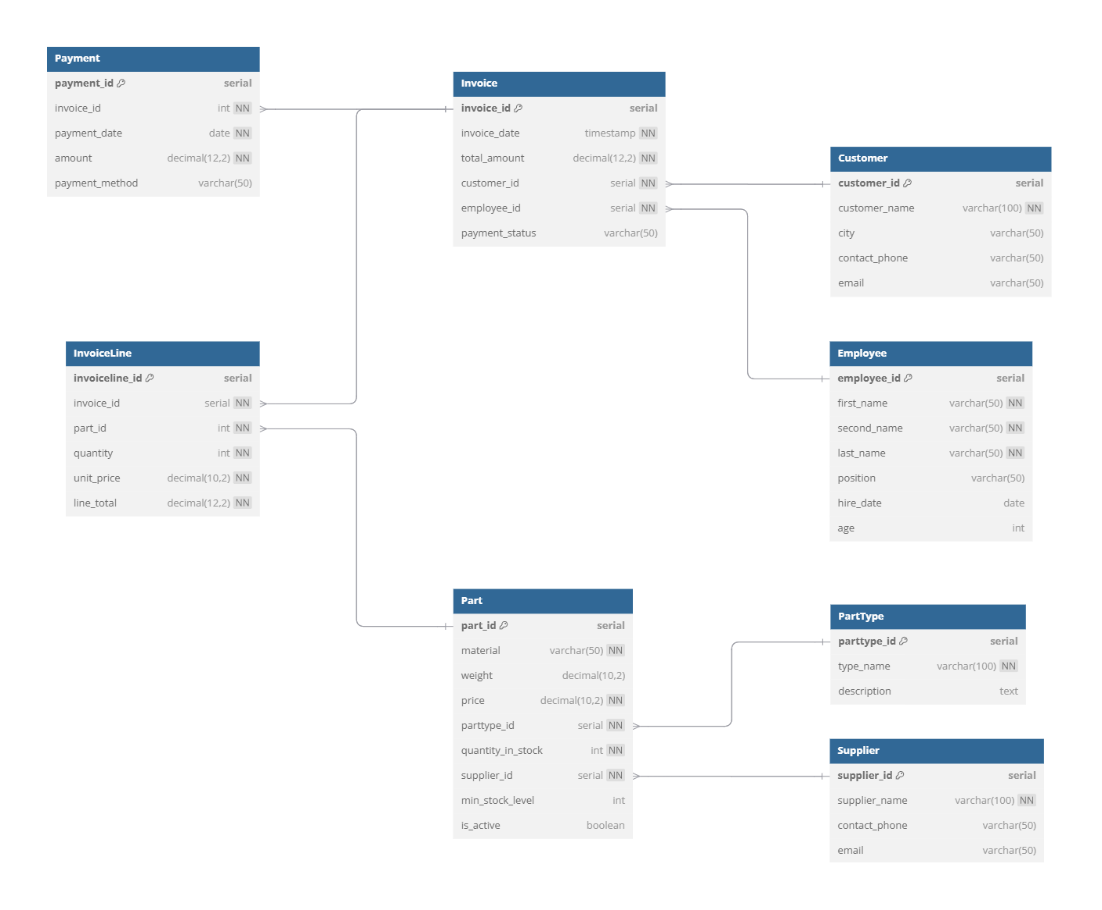
Модель разработанной базы данных “Продажа деталей” представлена на рисунке 2.

Рисунок 2 - Схема базы данных

На данном рисунке присутствует множество стрелок, соединяющих таблицы.

Стрелка, соединяющая таблицы “Строка накладной” и “Накладная”, является идентификацией первичного ключа “invoice\_id” таблицы “Накладная” и внешнего ключа “invoice\_id” таблицы “Строка накладной”.

Стрелка, соединяющая таблицы “Строка накладной” и “Деталь”, является идентификацией первичного ключа “part\_id” таблицы “Деталь” и внешнего ключа “part\_id” таблицы “Строка накладной”.

Стрелка, соединяющая таблицы “Платеж” и “Накладная”, является идентификацией первичного ключа “invoice\_id” таблицы “Накладная” и внешнего ключа “invoice\_id” таблицы “Платеж”.

Стрелка, соединяющая таблицы “Накладная” и “Покупатель”, является идентификацией первичного ключа “customer\_id” таблицы “Покупатель” и внешнего ключа “customer\_id” таблицы “Накладная”.

Стрелка, соединяющая таблицы “Накладная” и “Сотрудник”, является идентификацией первичного ключа “employee\_id” таблицы “Сотрудник” и внешнего ключа “employee\_id” таблицы “Накладная”.

Стрелка, соединяющая таблицы “Деталь” и “Тип детали”, является идентификацией первичного ключа “part type\_id” таблицы “Деталь” и внешнего ключа “paratype\_id” таблицы “Тип детали”.

Стрелка, соединяющая таблицы “Деталь” и “Поставщик”, является идентификацией первичного ключа “supplier\_id” таблицы “Деталь” и внешнего ключа “supplier\_id” таблицы “Поставщик”.

**4 Заполнение таблиц**

**4.1 Создание базы данных в pgAdmin4**

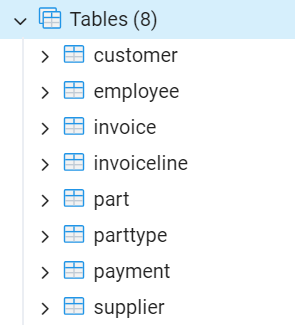
Таблицы были созданы при помощи sql-запроса, показанном в приложении А. В результате выполнения запроса были созданы 8 таблиц, показанных на рисунке 3:

Рисунок 3 - Созданные таблицы

**4.2 Заполнение таблиц**

Созданные ранее таблицы были заполнены при помощи функции “copy” после выполнения sql-запроса по созданию таблиц (смотреть приложение А). Файлы .csv были созданы при помощи скрипта на Python, с которым можно ознакомиться в приложении Б.

**5 Разработка запросов**

Согласно, заданию необходимо разработать не менее 7 запросов.

1) Проверка на необходимость пополнения деталей:

|  |
| --- |
| SELECT  part\_id,  material,  quantity\_in\_stock,  min\_stock\_level  FROM Part  WHERE quantity\_in\_stock < min\_stock\_level  ORDER BY min\_stock\_level - quantity\_in\_stock DESC |

 Данный запрос проверяет, какие детали на складе нуждаются в пополнении и сортирует их в порядке убывания количества нехватки. Данный запрос является необходимым для управления товарным ассортиментом. Результат запроса продемонстрирован на рисунке 4:

Рисунок 4 - Результат запроса №1

2) Просмотр самых активных покупателей за последний месяц:

|  |
| --- |
| SELECT customer\_id,  (SELECT customer\_name FROM Customer c WHERE c.customer\_id = Invoice.customer\_id) AS customer,  COUNT(\*) AS sold  FROM Invoice  WHERE invoice\_date >= current\_date - INTERVAL '1 month'  GROUP BY customer\_id  ORDER BY sold DESC |

 Данный запрос выводит самых активный покупателей. Активным считается покупатель, совершивший наибольшее количество покупок (на которого оформлено наибольшее количество накладных). Запрос содержит вложенный запрос для определения названия компании покупателя по ее ключу. Результат запроса продемонстрирован на рисунке 5:

Рисунок 5 - Результат запроса №2

3) Просмотр самых продаваемых деталей за все время:

|  |
| --- |
| SELECT  l.part\_id,  (SELECT p.material FROM Part p WHERE p.part\_id = l.part\_id),  (SELECT type\_name  FROM PartType pt  WHERE pt.parttype\_id = (  SELECT parttype\_id FROM Part p WHERE p.part\_id = l.part\_id  )),  SUM(l.line\_total) AS summ  FROM InvoiceLine l  GROUP BY l.part\_id  ORDER BY summ DESC |

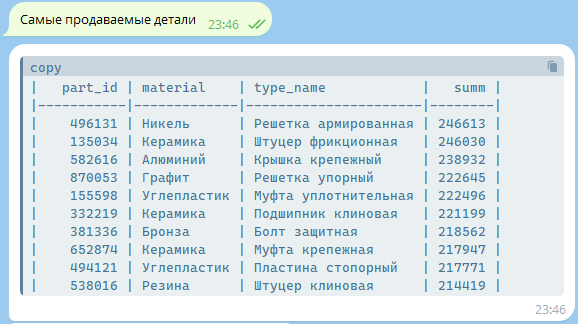
 Данный запрос выводит наиболее продаваемые детали. Продаваемой считается деталь, которую покупали наибольшее количество раз (на которую оформлено наибольшее количество накладных). Данный запрос содержит вложенные запросы для определения материала и типа детали по их ключам. Результат запроса продемонстрирован на рисунке 6:

Рисунок 6 - Результат запроса №3

4) Просмотр самых ценных сотрудников:

|  |
| --- |
| WITH EmployeeSales AS (  SELECT  e.second\_name || ' ' || e.first\_name || ' ' || e.last\_name AS full name,  (SELECT COUNT(\*) FROM Invoice WHERE employee\_id = e.employee\_id) AS invoices,  ROUND((SELECT SUM(total\_amount) FROM Invoice WHERE employee\_id = e.employee\_id), 2) AS  FROM Employee e  )  SELECT  full\_name,  invoices\_total,  ROUND(total / NULLIF(invoices, 0), 2) AS average,  total  FROM EmployeeSales  ORDER BY total DESC |

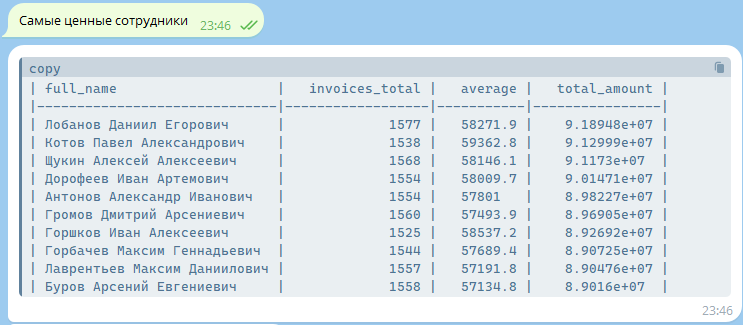
 Данный запрос выводит наиболее ценных сотрудников за все время. Ценным считается сотрудник, оформивший наибольшее количество накладных. Данный запрос содержит вложенные запросы для подсчета количества накладных, оформленных определенным сотрудником по его ключу, и для подсчета общей суммы накладных, оформленных тем же сотрудником. Также для улучшения читаемости запроса было использовано общее табличное выражение (WITH). Результат запроса продемонстрирован на рисунке 7:

Рисунок 7- Результат запроса №5

5) Просмотр самых крупных должников среди покупателей

|  |
| --- |
| SELECT  i.customer\_id,  (SELECT customer\_name || ', ' || city FROM Customer c WHERE c.customer\_id = i.customer\_id) AS customer\_name,  COUNT(\*) AS invoice\_count,  SUM(i.total\_amount) AS total\_billed,  SUM(COALESCE((  SELECT SUM(p.amount)  FROM Payment p  WHERE p.invoice\_id = i.invoice\_id  ), 0)) AS total\_paid,  SUM(i.total\_amount - COALESCE((  SELECT SUM(p.amount)  FROM Payment p  WHERE p.invoice\_id = i.invoice\_id  ), 0)) AS total\_due  FROM Invoice i  GROUP BY i.customer\_id  HAVING SUM(i.total\_amount - COALESCE((  SELECT SUM(p.amount)  FROM Payment p  WHERE p.invoice\_id = i.invoice\_id  ), 0)) > 0  ORDER BY total\_due DESC |

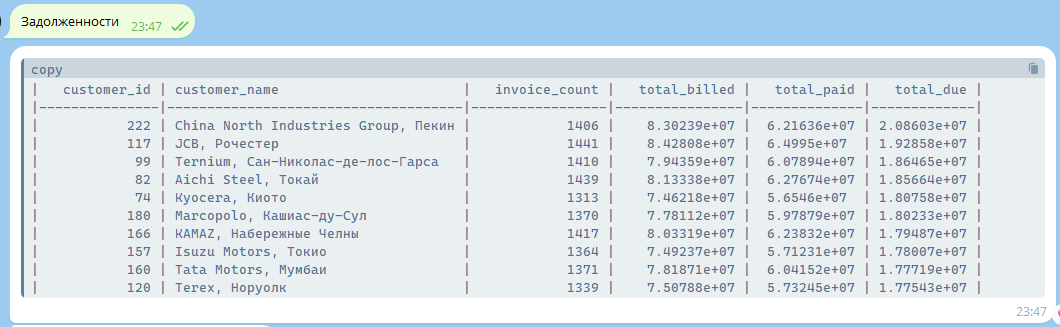
 Данный запрос выводит список компаний, общую сумму их заказов, общую сумму оплаты и их задолженности. Запрос содержит вложенные запросы для определения полного названия компании по их ключу, а также общей суммы заказов и платежей. Результат запроса продемонстрирован на рисунке 8:

Рисунок 8 - Результат запроса №6

6) Просмотр динамики продаж за месяц:

|  |
| --- |
| SELECT  TO\_CHAR(invoice\_date, 'YYYY-MM') AS month,  SUM(total\_amount) AS total\_sales  FROM Invoice  GROUP BY month  ORDER BY month DESC  LIMIT 12 |

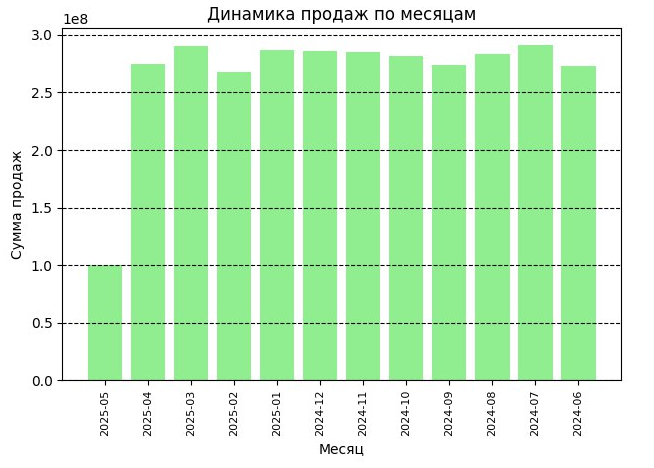
 Данный запрос выводит общую сумму проданного товара по месяцам. Запрос не является сложным, однако необходим для анализа продаж. На основании результатов данного запроса строится график при помощи Python скрипта, с которым можно ознакомиться в приложении В. Пример графика представлен на рисунке 9:

Рисунок 9 - Диаграмма динамики продаж

7) Просмотр соотношения платежей к долгам среди компаний покупателей:

|  |
| --- |
| SELECT  (SELECT customer\_name FROM Customer c WHERE c.customer\_id = i.customer\_id) AS customer\_name,  SUM(COALESCE((  SELECT SUM(p.amount)  FROM Payment p  WHERE p.invoice\_id = i.invoice\_id  ), 0)) AS total\_paid,  SUM(i.total\_amount - COALESCE((  SELECT SUM(p.amount)  FROM Payment p  WHERE p.invoice\_id = i.invoice\_id  ), 0)) AS total\_due  FROM Invoice i  GROUP BY i.customer\_id  HAVING SUM(i.total\_amount) > 0  ORDER BY SUM(i.total\_amount) DESC  LIMIT 10 |

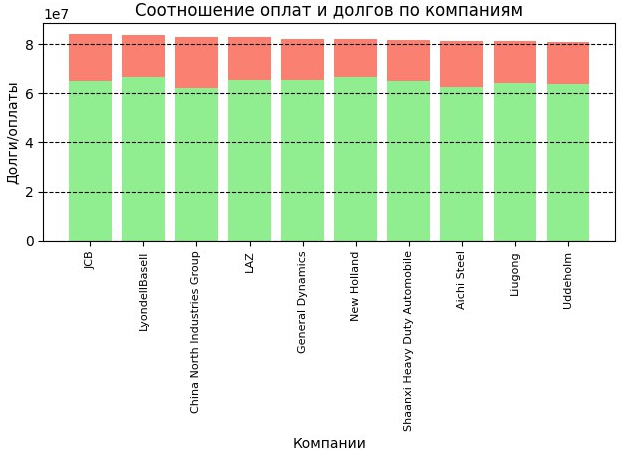
 Данный запрос выводит 10 компаний, отсортированных по убыванию общей оплаты. Данный запрос содержит подзапросы для определения общей суммы заказов и долгов каждой компании. На основании результатов данного запроса строится график при помощи Python скрипта, с которым можно ознакомиться в приложении В. Пример графика представлен на рисунке 10:

Рисунок 10 - Диаграмма задолженностей

8) Просмотр соотношения статусов платежей в накладных:

|  |
| --- |
| SELECT  payment\_status,  COUNT(\*) AS status\_count,  ROUND(COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM Invoice), 2) AS percentage  FROM Invoice  GROUP BY payment\_status |

Запрос не является сложным, однако необходим для анализа продаж. На основании результатов данного запроса строится график при помощи Python скрипта, с которым можно ознакомиться в приложении В. Пример графика представлен на рисунке 11:

Рисунок 11 - Диаграмма распределения статусов оплат

9) Просмотр накладных за определенный период:

|  |
| --- |
| SELECT  i.invoice\_id,  i.invoice\_date,  (SELECT customer\_name || ', ' || city FROM Customer c WHERE c.customer\_id = i.customer\_id) AS company\_name,  i.total\_amount,  i.payment\_status  FROM Invoice i  WHERE invoice\_date BETWEEN %s AND %s  ORDER BY invoice\_date; |

Данный запрос выводит список компаний и информацию о накладных, оформленных за период, указанный пользователем. Данный запрос содержит подзапрос для определения полного названия компании по ее ключу.

Для поддержания целостности данных, оптимизации запросов и предотвращения ошибок, были созданы триггеры для:

1. Проверки на занятость номера при добавлении и изменении записи в таблице “Поставщик”
2. Проверки на занятость названия и города при добавлении и изменении записи в таблице “Покупатель”
3. Проверки на занятость названия типа при добавлении и изменении записи в таблице “Тип детали”
4. Проверки на занятость материала и типа детали при добавлении и изменении записи в таблице “Деталь”
5. Проверки совершеннолетия сотрудника при добавлении записи в таблицу “Сотрудник”
6. Проверки наличия необходимого количества деталей на складе перед добавлением записи в таблицу “Строка накладной”
7. Автоматического уменьшения количества деталей на складе при добавлении записи в таблицу “Строка накладной”
8. Предотвращения ошибки, связанной с добавлением даты платежа до даты оформления накладной, при добавлении записи в таблицу “Платеж”
9. Предотвращения ошибки, связанной с добавлением суммы платежа, превышающей оставшуюся сумму долга при добавлении записи в таблицу “Платеж”
10. Предотвращения ошибки, связанной с попыткой оплатить уже оплаченную накладную, при добавлении записи в таблицу “Платеж”
11. Поиска минимального незанятого id при добавлении записи в таблицу “Покупатель”
12. Поиска минимального незанятого id при добавлении записи в таблицу “Поставщик”
13. Поиска минимального незанятого id при добавлении записи в таблицу “Платеж”
14. Поиска минимального незанятого id при добавлении записи в таблицу “Сотрудник”
15. Поиска минимального незанятого id при добавлении записи в таблицу “Деталь”
16. Поиска минимального незанятого id при добавлении записи в таблицу “Тип детали”
17. Поиска минимального незанятого id при добавлении записи в таблицу “Накладная”
18. Поиска минимального незанятого id при добавлении записи в таблицу “Строка накладной”
19. Автоматического обновления статуса наличия в продаже при изменении количества деталей на складе в таблице “Деталь”
20. Автоматического обновления статуса оплаты при добавлении записи в таблицу “Платеж”
21. Автоматического обновления общей суммы покупки в накладной при добавлении, удалении и изменении записи в таблице “Строка накладной”

Подробно ознакомиться с каждым из них можно в приложении Г.

**6 Разработка интерфейса**

Для реализации интерфейсной части было принято решение создать Telegram-бота с помощью библиотеки telebot в Python. Подробнее код бота описан в приложении Д. Это обеспечит простоту использования, кроссплатформенность и упрощенную аутентификацию, поскольку пользователь уже авторизован через Telegram. Взаимодействие между пользователем и базой данных реализовано при помощи библиотеки psycopg2. Ниже приведен пример реализации одного из запросов:

|  |
| --- |
| import psycopg2  def get\_connection():  return psycopg2.connect(\*\*database\_settings)  def check\_fill():  conn = get\_connection()  cur = conn.cursor()  if not cur.fetchone():  raise ValueError("Ничего пополнять не нужно")  cur.execute("""  SELECT  part\_id,  material,  quantity\_in\_stock,  min\_stock\_level  FROM Part  WHERE quantity\_in\_stock < min\_stock\_level  ORDER BY min\_stock\_level - quantity\_in\_stock DESC  LIMIT 30  """)  rows = cur.fetchall()  colnames = [desc[0] for desc in cur.description]  cur.close()  conn.close()  return colnames, rows |

Некоторые запросы видоизменены для того, чтобы подходить под ограничения на количество сообщений в Telegram. Однако, для того чтобы предоставить пользователю возможность просмотра более подробной информации, было принято решение вместе с фрагментом таблицы отправлять 2 файла в формате .txt и .md.

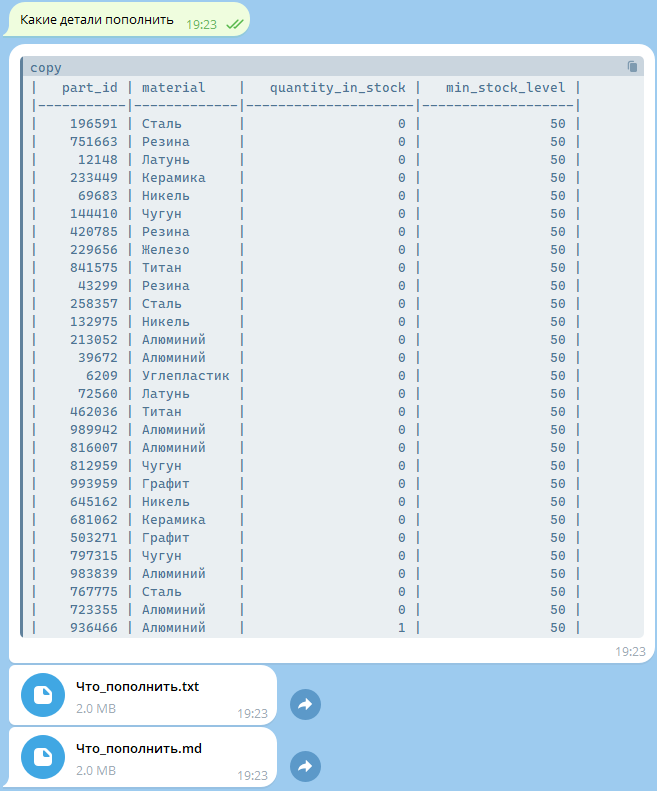
Вид от лица пользователя представлен на рисунке 12:

Рисунок 12 - Вид от лица пользователя

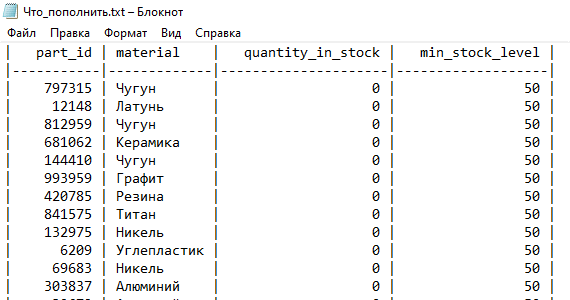
На рисунках 13 и 14 продемонстрирован формат, в котором находятся данные внутри файлов “Что\_пополнить.txt” и “Что\_пополнить.md”:

Рисунок 14 - Содержимое файла md

Рисунок 13 - Содержимое файла txt

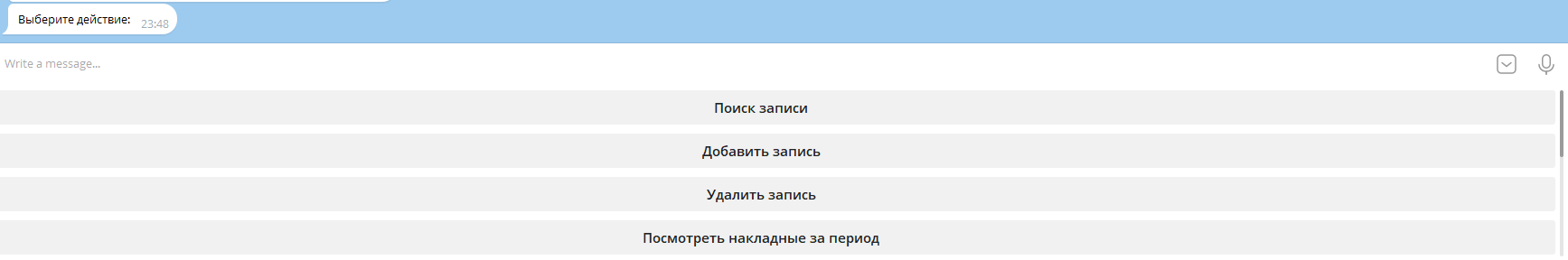
Для выбора выполняемой функции было реализовано меню. Оно представлено на рисунке 15:

Рисунок 15 - Меню

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы была разработана автоматизированная система учета продаж деталей строительного и промышленного назначения. Система основана на реляционной базе данных, реализованной с использованием СУБД PostgreSQL, и включает пользовательский интерфейс в виде Telegram-бота, созданного на языке Python.

В ходе реализации проекта были выполнены следующие этапы:

* Анализ предметной области, изучение бизнес-процессов и требований к системе
* Проектирование структуры базы данных, включая схемы таблиц, связи, индексы и триггеры
* Реализация SQL-скриптов для создания объектов базы данных и заполнения тестовыми данными
* Создание Telegram-бота с интуитивно понятным интерфейсом для взаимодействия пользователей с системой

Система демонстрирует эффективный подход к автоматизации учета в сфере продаж промышленных комплектующих и может быть адаптирована для аналогичных предметных областей.

**ПРИЛОЖЕНИЕ А**

**SQL-запрос для создания и заполнения базы данных**

CREATE TABLE PartType (

    parttype\_id SERIAL *PRIMARY KEY*,

    type\_name VARCHAR(100) NOT NULL,

    description TEXT

);

copy parttype FROM 'C:/tables/part\_types.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',', ENCODING 'UTF8');

CREATE TABLE Supplier (

    supplier\_id SERIAL *PRIMARY KEY*,

    supplier\_name VARCHAR(100) NOT NULL,

    contact\_phone VARCHAR(50),

    email VARCHAR(50)

);

copy supplier FROM 'C:/tables/suppliers.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',', ENCODING 'UTF8');

CREATE TABLE Part (

    part\_id SERIAL *PRIMARY KEY*,

    material VARCHAR(50) NOT NULL,

    weight DECIMAL(10,2),

    price DECIMAL(10,2) NOT NULL,

    parttype\_id SERIAL *REFERENCES* PartType(parttype\_id),

    quantity\_in\_stock INT NOT NULL,

    supplier\_id SERIAL *REFERENCES* Supplier(supplier\_id),

    min\_stock\_level INT *DEFAULT* 0,

    is\_active BOOLEAN *DEFAULT* TRUE

);

copy part FROM 'C:/tables/parts.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',', ENCODING 'UTF8');

CREATE TABLE Customer (

    customer\_id SERIAL *PRIMARY KEY*,

    customer\_name VARCHAR(100) NOT NULL,

    city VARCHAR(50),

    contact\_phone VARCHAR(50),

    email VARCHAR(50)

);

copy customer FROM 'C:/tables/customers.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',', ENCODING 'UTF8');

CREATE TABLE Employee (

    employee\_id SERIAL *PRIMARY KEY*,

    first\_name VARCHAR(50) NOT NULL,

    second\_name VARCHAR(50) NOT NULL,

    last\_name VARCHAR(50) NOT NULL,

    position VARCHAR(50),

    hire\_date DATE,

   age INT *CHECK* (age BETWEEN 18 AND 65)

);

copy employee FROM 'C:/tables/employees.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',', ENCODING 'UTF8');

CREATE TABLE Invoice (

    invoice\_id SERIAL *PRIMARY KEY*,

    invoice\_date TIMESTAMP NOT NULL,

    total\_amount DECIMAL(12,2) NOT NULL,

    customer\_id SERIAL *REFERENCES* Customer(customer\_id),

    employee\_id SERIAL *REFERENCES* Employee(employee\_id),

    payment\_status VARCHAR(50) *CHECK* (payment\_status IN ('Оплачено', 'Не оплачено', 'Частично оплачено')) *DEFAULT* 'Не оплачено'

);

copy invoice FROM 'C:/tables/invoices.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',', ENCODING 'UTF8');

CREATE TABLE InvoiceLine (

    invoiceline\_id SERIAL *PRIMARY KEY*,

    invoice\_id SERIAL *REFERENCES* Invoice(invoice\_id),

    part\_id INT *REFERENCES* Part(part\_id),

    quantity INT NOT NULL *CHECK* (quantity > 0),

    unit\_price DECIMAL(10,2) NOT NULL,

    line\_total DECIMAL(12,2) NOT NULL

);

copy invoiceline FROM 'C:/tables/invoice\_lines.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',', ENCODING 'UTF8');

CREATE TABLE Payment (

    payment\_id SERIAL *PRIMARY KEY*,

    invoice\_id INT *REFERENCES* Invoice(invoice\_id),

    payment\_date DATE NOT NULL,

    amount DECIMAL(12,2) NOT NULL *CHECK* (amount > 0),

    payment\_method VARCHAR(50) *CHECK* (payment\_method IN ('Наличный расчет', 'Безналичный расчет')) *DEFAULT* 'Безналичный расчет'

);

COPY payment FROM 'C:/tables/payments.csv' WITH (FORMAT csv, HEADER true, DELIMITER ',', ENCODING 'UTF8');

*-- Индексы для ускорения*

CREATE INDEX idx\_part\_parttype ON Part(parttype\_id);

CREATE INDEX idx\_part\_supplier ON Part(supplier\_id);

CREATE INDEX idx\_invoice\_customer ON Invoice(customer\_id);

CREATE INDEX idx\_invoice\_employee ON Invoice(employee\_id);

CREATE INDEX idx\_invoice\_date ON Invoice(invoice\_date);

CREATE INDEX idx\_invoiceline\_part ON InvoiceLine(part\_id);

CREATE INDEX idx\_invoiceline\_invoice ON InvoiceLine(invoice\_id);

CREATE INDEX idx\_payment\_invoice ON Payment(invoice\_id);

CREATE INDEX idx\_payment\_date ON Payment(payment\_date);

**ПРИЛОЖЕНИЕ Б**

**Код генератора данных**

import csv

import random

from datetime import datetime, timedelta

from faker import Faker

from natasha import MorphVocab

m = MorphVocab()

fake = Faker("ru\_RU")

*# Настройки*

NUM\_PARTS = 1\_000\_000 *#*

NUM\_PART\_TYPES = 100 *#*

NUM\_SUPPLIERS = 114 *#*

NUM\_CUSTOMERS = 222 *#*

NUM\_EMPLOYEES = 200 *#*

NUM\_INVOICES = 300\_000

MAX\_INVOICE\_LINES = 1\_000\_000

*# Генерация данных для таблицы Customer (Покупатели)*

def generate\_customers():

    customers = []

    companies = ['Doosan Heavy Industries', 'Rolls-Royce Holdings', 'General Dynamics', 'Lockheed Martin', 'Boeing', 'Northrop Grumman', 'Raytheon Technologies', 'L3Harris Technologies', 'BAE Systems', 'Safran', 'United Technologies', 'PACCAR', 'Continental AG', 'Magna International', 'Dana Incorporated', 'Aisin Seiki', 'NSK Ltd.', 'Schaeffler Group', 'GKN Automotive', 'Mahle GmbH', 'Tenneco', 'Yazaki Corporation', 'Sumitomo Electric', 'Furukawa Electric', 'Nexans', 'Prysmian Group', 'Legrand', 'Hubbell Incorporated', 'nVent Electric', 'Amphenol', 'TE Connectivity', 'Molex', 'Aptiv', 'Lear Corporation', 'Adient', 'BASF', 'Dow Chemical', 'DuPont', 'LyondellBasell', 'Linde plc', 'Air Liquide', 'Praxair', 'Mitsubishi Chemical', 'Sumitomo Chemical', 'Toray Industries', 'LG Chem', 'Samsung SDI', 'SK Innovation', 'ThyssenKrupp Steel', 'Alcoa', 'Rio Tinto', 'BHP', 'Glencore', 'Freeport-McMoRan', 'Vale S.A.', 'Anglo American', 'Barrick Gold', 'Newmont Corporation', 'Fluor Corporation', 'Bechtel', 'Jacobs Engineering', 'AECOM', 'KBR', 'McDermott International', 'TechnipFMC', 'Saipem', 'Subsea 7', 'Weatherford International', 'NOV (National Oilwell Varco)', 'Mapal', 'Guhring', 'Mitsubishi Materials', 'Sumitomo Electric Hardmetal', 'Kyocera', 'Ceratizit', 'Plansee Group', 'Hitachi Metals', 'Daido Steel', 'Nippon Steel & Sumitomo Metal', 'JFE Steel', 'Kobe Steel', 'Aichi Steel', 'Sanyo Special Steel', 'Ovako', 'Uddeholm', 'Böhler', 'Carpenter Technology', 'Allegheny Technologies', 'Haynes International', 'Special Metals Corporation', 'VDM Metals', 'Outokumpu Stainless', 'Steel Dynamics', 'AK Steel', 'U.S. Steel', 'Cleveland-Cliffs', 'CSN (Companhia Siderúrgica Nacional)', 'Usiminas', 'Ternium', 'Techint Group', 'Tenova', 'Danieli', 'SMS Group', 'Primetals Technologies', 'Andritz', 'Loesche', 'Gebr. Pfeiffer', 'Polysius', 'Claudius Peters', 'Schenck Process', 'Bühler Group', 'GEA Group', 'ITT Inc.', 'Kaeser Kompressoren', 'Hitachi Construction Machinery', 'Kobelco', 'JCB', 'CASE Construction', 'New Holland', 'Terex', 'Manitowoc', 'Tadano', 'Sennebogen', 'Palfinger', 'Haulotte', 'JLG Industries', 'Genie (Terex)', 'Skyjack', 'Manitou Group', 'Doosan Infracore', 'Hyundai Construction Equipment', 'Sany Heavy Industry', 'Zoomlion', 'XCMG', 'Liugong', 'Lonking', 'Shantui', 'SDLG', 'Yutong Heavy Industries', 'Sinotruk', 'FAW', 'Dongfeng Motor', 'Shaanxi Heavy Duty Automobile', 'Beiben Trucks', 'Iveco', 'Volvo Trucks', 'DAF Trucks', 'Renault Trucks', 'Mercedes-Benz Trucks', 'Freightliner', 'Western Star', 'Kenworth', 'Peterbilt', 'Mack Trucks', 'Navistar International', 'Hino Motors', 'Isuzu Motors', 'UD Trucks', 'Fuso', 'Tata Motors', 'Ashok Leyland', 'Eicher Motors', 'Mahindra Truck and Bus', 'Kamaz', 'GAZ Group', 'KAMAZ', 'MAZ', 'KrAZ', 'BelAZ', 'Scania-Vabis', 'Van Hool', 'VDL Groep', 'Daimler Buses', 'Volvo Buses', 'Irizar', 'Solaris Bus & Coach', 'MAN Bus', 'Neoplan', 'Setra', 'Marcopolo', 'Caio Induscar', 'Comil', 'Agrale', 'Tatra', 'Avtotor', 'Sollers', 'UAZ', 'ZIL', 'KAMAZ', 'BAZ', 'ZiL', 'AMO ZIL', 'UralAZ', 'KAvZ', 'PAZ', 'LiAZ', 'NefAZ', 'MAZ', 'BelAZ', 'MoAZ', 'KrAZ', 'LAZ', 'Bogdan', 'Etalon', 'Volkswagen Commercial Vehicles', 'Ford Trucks', 'Iveco Defence Vehicles', 'Oshkosh Corporation', 'Navistar Defense', 'Rheinmetall MAN Military Vehicles', 'BAE Systems Land & Armaments', 'General Dynamics Land Systems', 'Textron Marine & Land Systems', 'Patria', 'Nexter Systems', 'Krauss-Maffei Wegmann', 'Hyundai Rotem', 'Doosan DST', 'Hanwha Defense', 'Norinco', 'Sinomach', 'China North Industries Group']

    cities = ['Чханвон', 'Лондон', 'Рестон', 'Бетесда', 'Чикаго', 'Фолс-Черч', 'Уолтем', 'Мельбурн', 'Лондон', 'Париж', 'Фармингтон', 'Белвью', 'Ганновер', 'Орора', 'Моми', 'Карья', 'Токио', 'Херцогенаурах', 'Реддич', 'Штутгарт', 'Лейк-Форест', 'Токио', 'Осака', 'Токио', 'Париж', 'Милан', 'Лимож', 'Шелтон', 'Лондон', 'Уоллингфорд', 'Шаффхаузен', 'Лайл', 'Дублин', 'Саутфилд', 'Плимут', 'Людвигсхафен', 'Мидленд', 'Уилмингтон', 'Хьюстон', 'Гилфорд', 'Париж', 'Данбери', 'Токио', 'Токио', 'Токио', 'Сеул', 'Сеул', 'Сеул', 'Дуйсбург', 'Питтсбург', 'Мельбурн', 'Мельбурн', 'Баар', 'Финикс', 'Рио-де-Жанейро', 'Лондон', 'Торонто', 'Денвер', 'Ирвинг', 'Рестон', 'Даллас', 'Лос-Анджелес', 'Хьюстон', 'Хьюстон', 'Лондон', 'Сан-Донато-Миланезе', 'Лондон', 'Хьюстон', 'Хьюстон', 'Аален', 'Альбштадт', 'Токио', 'Осака', 'Киото', 'Маммер', 'Ройтте', 'Токио', 'Нагоя', 'Токио', 'Токио', 'Кобе', 'Токай', 'Химэдзи', 'Стокгольм', 'Хагфорс', 'Капфенберг', 'Рединг', 'Питтсбург', 'Кокомо', 'Хантингтон', 'Вердоль', 'Эспоо', 'Форт-Уэйн', 'Уэст-Честер', 'Питтсбург', 'Кливленд', 'Сан-Паулу', 'Белу-Оризонти', 'Сан-Николас-де-лос-Гарса', 'Милан', 'Милан', 'Буттрио', 'Дюссельдорф', 'Лондон', 'Грац', 'Дюссельдорф', 'Кайзерслаутерн', 'Эссен', 'Гамбург', 'Дармштадт', 'Уцвиль', 'Дюссельдорф', 'Уайт-Плейнс', 'Кобург', 'Токио', 'Токио', 'Рочестер', 'Расин', 'Турин', 'Норуолк', 'Манитовок', 'Такамацу', 'Штраубинг', 'Зальцбург', 'Лорм', 'Хейгерстаун', 'Редмонд', 'Гвелф', 'Ансени', 'Сеул', 'Сеул', 'Чанша', 'Чанша', 'Сюйчжоу', 'Лючжоу', 'Шанхай', 'Цзинин', 'Линьи', 'Чжэнчжоу', 'Цзинань', 'Чанчунь', 'Ухань', 'Сиань', 'Чунцин', 'Турин', 'Гетеборг', 'Эйндховен', 'Сен-Приест', 'Штутгарт', 'Портленд', 'Портленд', 'Киркланд', 'Дентон', 'Гринсборо', 'Лайл', 'Токио', 'Токио', 'Агео', 'Кавасаки', 'Мумбаи', 'Ченнаи', 'Гургаон', 'Мумбаи', 'Набережные Челны', 'Нижний Новгород', 'Набережные Челны', 'Минск', 'Кременчуг', 'Жодино', 'Седертелье', 'Конингсхойкт', 'Эйндховен', 'Штутгарт', 'Гетеборг', 'Ормайстеги', 'Болехово', 'Мюнхен', 'Штутгарт', 'Ульм', 'Кашиас-ду-Сул', 'Сан-Паулу', 'Эрешин', 'Кашиас-ду-Сул', 'Копрживнице', 'Калининград', 'Москва', 'Ульяновск', 'Москва', 'Набережные Челны', 'Брянск', 'Москва', 'Москва', 'Миасс', 'Курган', 'Павлово', 'Ликино-Дулево', 'Нефтекамск', 'Минск', 'Жодино', 'Могилев', 'Кременчуг', 'Львов', 'Черкассы', 'Москва', 'Ганновер', 'Стамбул', 'Больцано', 'Ошкош', 'Уоррен', 'Мюнхен', 'Стерлинг-Хайтс', 'Лондон', 'Новый Орлеан', 'Хельсинки', 'Версаль', 'Мюнхен', 'Чханвон', 'Сеул', 'Сеул', 'Пекин', 'Пекин', 'Пекин']

    mailboxes = ['gmail.com', 'yahoo.com', 'outlook.com', 'hotmail.com', 'aol.com', 'mail.com', 'protonmail.com', 'zoho.com', 'yandex.ru', 'mail.ru', 'icloud.com', 'gmx.com', 'tutanota.com', 'fastmail.com', 'hushmail.com', 'qq.com', '163.com', 'rediffmail.com', 'sina.com', 'naver.com']

    for i in range(0, NUM\_CUSTOMERS):

        customers.append({

            'customer\_id': i + 1,

            'customer\_name': companies[i],

            'city': cities[i],

            'contact\_phone': fake.phone\_number(),

            'email': f'{companies[i].replace(' ', '').lower()}@{random.choice(mailboxes)}'

        })

    return customers

*# Генерация данных для таблицы PartType (Типы деталей)*

def generate\_part\_types():

    part\_types = []

    types = [

    'Вал', 'Шестерня', 'Подшипник', 'Муфта',

    'Крышка', 'Болт', 'Гайка', 'Шайба',

    'Цепь', 'Ролик', 'Клапан', 'Фильтр',

    'Трубка', 'Штуцер', 'Пластина',

    'Диск', 'Лента', 'Рычаг', 'Решетка'

    ]

    adjectives = [

        'упорн', 'стопорн', 'переходн', 'соединительн',

        'защитн', 'крепежн', 'пружинн',

        'вращательн', 'клинов','уплотнительн',

        'армированн', 'обратн',

        'регулировочн', 'фрикционн'

    ]

    part\_functions = [

    'передачи вращательного момента', 'фиксации компонентов', 'уплотнения соединений',

    'снижения трения', 'передачи мощности', 'соединения валов', 'защиты механизмов',

    'крепления деталей', 'создания упора', 'регулировки зазоров', 'отвода жидкостей',

    'очистки рабочих сред', 'демпфирования вибраций', 'изменения передаточного отношения',

    'торможения системы', 'переключения скоростей', 'охлаждения узлов', 'фильтрации масел',

    'компенсации температурных расширений', 'перераспределения нагрузок'

    ]

    applications = [

        'в редукторах', 'в двигателях внутреннего сгорания', 'в коробках передач',

        'в насосном оборудовании', 'в компрессорах', 'в турбинных установках',

        'в станках ЧПУ', 'в гидравлических системах', 'в пневматических линиях',

        'в автомобильных трансмиссиях', 'в авиационных агрегатах', 'в железнодорожной технике',

        'в горнодобывающем оборудовании', 'в судовых механизмах', 'в энергетических установках'

    ]

    while len(part\_types) < NUM\_PART\_TYPES:

        type = random.choice(types)

        if m.parse(type)[0].feats['Gender'] == 'Masc':

            ending = 'ый'

        elif m.parse(type)[0].feats['Gender'] == 'Fem':

            ending = 'ая'

        type\_name = f'{random.choice(types)} {random.choice(adjectives) + ending}'

        if type\_name not in part\_types:

            part\_types.append({

                'parttype\_id': len(part\_types) + 1,

                'type\_name': type\_name,

                'description': f'Используется для {random.choice(part\_functions)} {random.choice(applications)}'

            })

    return part\_types

*# Генерация данных для таблицы Supplier (Поставщики)*

def generate\_suppliers():

    suppliers = []

    suppliers\_names = ['Bosch', 'Siemens', 'General Electric', 'Caterpillar', 'Honeywell', '3M', 'ABB', 'Schneider Electric', 'Mitsubishi Heavy Industries', 'Hitachi', 'Toshiba', 'Hyundai Heavy Industries', 'Doosan', 'ThyssenKrupp', 'Rolls-Royce', 'Alstom', 'Emerson Electric', 'Rockwell Automation', 'Cummins', 'John Deere', 'Volvo Group', 'Daimler Truck', 'Parker Hannifin', 'Eaton', 'ZF Friedrichshafen', 'Continental', 'Magna', 'BorgWarner', 'Valeo', 'Faurecia', 'Mahle', 'Knorr-Bremse', 'Wabtec', 'SKF', 'Timken', 'NSK', 'NTN', 'JTEKT', 'Schaeffler', 'GKN', 'Sandvik', 'Kennametal', 'Walter AG', 'Iscar', 'Seco Tools', 'MAPAL', 'Dormer Pramet', 'Gühring', 'Liebherr', 'Komatsu', 'Wärtsilä', 'MAN Energy Solutions', 'Sulzer', 'Atlas Copco', 'Ingersoll Rand', 'Gardner Denver', 'Sullair', 'Kaeser', 'Grundfos', 'KSB', 'Wilo', 'Xylem', 'Flowserve', 'ITT Inc', 'Pentair', 'Alfa Laval', 'GEA', 'SPX Flow', 'SMC Corporation', 'Festo', 'Pall', 'Donaldson', 'Parker Hannifin', 'Swagelok', 'Camozzi', 'Norgren', 'Rotork', 'AUMA', 'Siemens Healthineers', 'Philips', 'GE Healthcare', 'Baker Hughes', 'Halliburton', 'Schlumberger', 'Weatherford', 'National Oilwell Varco', 'Tenaris', 'Vallourec', 'TMK', 'SSAB', 'Voestalpine', 'ArcelorMittal', 'Nippon Steel', 'POSCO', 'Tata Steel', 'Outokumpu', 'Aperam', 'Nucor', 'Gerdau', 'Severstal', 'Metso Outotec', 'FLSmidth', 'Weir Group', 'KHD Humboldt Wedag', 'Claas', 'CNH Industrial', 'AGCO', 'Kubota', 'Yanmar', 'Deutz', 'Perkins', 'MTU', 'Scania', 'MAN Truck & Bus']

    mailboxes = ['gmail.com', 'yahoo.com', 'outlook.com', 'hotmail.com', 'aol.com', 'mail.com', 'protonmail.com', 'zoho.com', 'yandex.ru', 'mail.ru', 'icloud.com', 'gmx.com', 'tutanota.com', 'fastmail.com', 'hushmail.com', 'qq.com', '163.com', 'rediffmail.com', 'sina.com', 'naver.com']

    for i in range(0, NUM\_SUPPLIERS):

        suppliers.append({

            'supplier\_id': i + 1,

            'supplier\_name': suppliers\_names[i],

            'contact\_phone': fake.phone\_number(),

            'email': f'{suppliers\_names[i].replace(' ', '').lower()}@{random.choice(mailboxes)}'

        })

    return suppliers

*# Генерация данных для таблицы Part (Детали)*

def generate\_parts():

    materials = ['Сталь', 'Чугун', 'Алюминий', 'Медь', 'Латунь', 'Бронза', 'Титан', 'Никель', 'Железо', 'Резина', 'Углепластик', 'Керамика', 'Графит']

    parts = []

    while len(parts) < NUM\_PARTS:

        part = {

            'part\_id': len(parts) + 1,

            'material': random.choice(materials),

            'weight\_kg': round(random.uniform(0.01, 50.0), 3),

            'price\_usd': round(random.uniform(0.1, 500.0), 2),

            'parttype\_id': random.randint(1, NUM\_PART\_TYPES),

            'quantity\_in\_stock': random.randint(0, 1000),

            'supplier\_id': random.randint(1, NUM\_SUPPLIERS),

            'min\_stock\_level': random.randint(5, 50)

        }

        if part['quantity\_in\_stock'] < part['min\_stock\_level']:

            part['is\_active'] = False

        else:

            part['is\_active'] = True

        parts.append(part)

    return parts

*# Генерация данных для таблицы Employee (Сотрудники)*

def generate\_employees():

    positions = ['Менеджер по продажам', 'Специалист по закупкам', 'Логист', 'Технический консультант', 'Маркетолог', 'Складской работник', 'Сервисный инженер']

    employees = []

    for i in range(0, NUM\_EMPLOYEES):

        first\_names = ['Александр', 'Дмитрий', 'Максим', 'Сергей', 'Андрей', 'Алексей', 'Артем', 'Илья', 'Петр', 'Михаил', 'Геннадий', 'Матвей', 'Роман', 'Егор', 'Арсений', 'Иван', 'Денис', 'Евгений', 'Даниил', 'Павел']

        last\_names = ['Александрович', 'Дмитриевич', 'Максимович', 'Сергеевич', 'Андреевич', 'Алексеевич', 'Артемович', 'Ильич', 'Петрович', 'Михаилович', 'Геннадьевич', 'Матвеевич', 'Романович', 'Егорович', 'Арсениевич', 'Иванович', 'Денисович', 'Евгениевич', 'Даниилович', 'Павелович']

        age = random.randint(18, 65)

        employees.append({

            'employee\_id': i + 1,

            'first\_name': random.choice(first\_names),

            'second\_name': fake.last\_name\_male(),

            'last\_name': random.choice(last\_names),

            'position': random.choice(positions),

            'hire\_date': fake.date\_between(start\_date=f'-{(age-18)\*365+10}d', end\_date='-1d').isoformat(),

            'age': age

        })

    return employees

*# Генерация данных для таблицы Invoice (Накладные)*

def generate\_invoices():

    invoices = []

    for i in range(1, NUM\_INVOICES + 1):

        invoice\_date = fake.date\_time\_between(start\_date=datetime.now()-timedelta(days=365\*5), end\_date=datetime.now())

        invoices.append({

            'invoice\_id': i,

            'invoice\_date': invoice\_date.isoformat(),

            'total\_amount': 0,

            'customer\_id': random.randint(1, NUM\_CUSTOMERS),

            'employee\_id': random.randint(1, NUM\_EMPLOYEES),

            'payment\_status': 'Не оплачено'

        })

    return invoices

*# Генерация данных для таблицы InvoiceLine (Строки накладных)*

def generate\_invoice\_lines(invoices, parts):

    invoice\_lines = []

    line\_id = 1

    active\_parts = [p for p in parts if p['is\_active']]

    for invoice in invoices:

        num\_lines = random.randint(1, 8)

        invoice\_total = 0

        for \_ in range(num\_lines):

            part = random.choice(active\_parts)

            quantity = random.randint(1, 100)

            unit\_price = part['price\_usd']

            line\_total = round(quantity \* unit\_price, 2)

            invoice\_lines.append({

                'invoiceline\_id': line\_id,

                'invoice\_id': invoice['invoice\_id'],

                'part\_id': part['part\_id'],

                'quantity': quantity,

                'unit\_price': unit\_price,

                'line\_total': line\_total

            })

            invoice\_total += line\_total

            line\_id += 1

        invoice['total\_amount'] = round(invoice\_total, 2)

    return invoice\_lines

*# Генерация данных для таблицы Payments (Платежи)*

def generate\_payments(invoices):

    payments = []

    line\_id = 1

    for invoice in invoices:

        status\_chance = random.random()

        total = invoice['total\_amount']

        paid = 0

        if status\_chance < 0.7:

            num\_parts = random.randint(1, 5)

            part\_amounts = [round(total / num\_parts, 2) for \_ in range(num\_parts - 1)]

            part\_amounts.append(round(total - sum(part\_amounts), 2))

            for amount in part\_amounts:

                payment\_date = fake.date\_time\_between(

                    start\_date=datetime.fromisoformat(invoice['invoice\_date']),

                    end\_date=datetime.now()

                )

                payments.append({

                    'payment\_id': line\_id,

                    'invoice\_id': invoice['invoice\_id'],

                    'payment\_date': payment\_date.isoformat(),

                    'amount': amount,

                    'payment\_method': random.choices(

                        ['Наличный расчет', 'Безналичный расчет'], weights=[10, 90]

                    )[0]

                })

                line\_id += 1

                paid += amount

            invoice['payment\_status'] = 'Оплачено'

        elif status\_chance < 0.9:

            amount = round(random.uniform(0.1, total \* 0.9), 2)

            payment\_date = fake.date\_time\_between(

                start\_date=datetime.fromisoformat(invoice['invoice\_date']),

                end\_date=datetime.now()

            )

            payments.append({

                'payment\_id': line\_id,

                'invoice\_id': invoice['invoice\_id'],

                'payment\_date': payment\_date.isoformat(),

                'amount': amount,

                'payment\_method': random.choice(['Наличный расчет', 'Безналичный расчет'])

            })

            line\_id += 1

            invoice['payment\_status'] = 'Частично оплачено'

        else:

            invoice['payment\_status'] = 'Не оплачено'

    return payments

*# Функция для сохранения данных в CSV*

def save\_to\_csv(data, filename, fieldnames):

    with open(filename, 'w', newline='', encoding='utf-8') as csvfile:

        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()

        writer.writerows(data)

    print(f'Saved {len(data)} records to {filename}')

*# Основной процесс генерации данных*

def main():

    print('Generating customers...')

    customers = generate\_customers()

    save\_to\_csv(customers, 'tables/customers.csv', ['customer\_id', 'customer\_name', 'city', 'contact\_phone', 'email'])

    print('Generating part types...')

    part\_types = generate\_part\_types()

    save\_to\_csv(part\_types, 'tables/part\_types.csv', ['parttype\_id', 'type\_name', 'description'])

    print('Generating suppliers...')

    suppliers = generate\_suppliers()

    save\_to\_csv(suppliers, 'tables/suppliers.csv', ['supplier\_id', 'supplier\_name', 'contact\_phone', 'email'])

    print('Generating parts...')

    parts = generate\_parts()

    save\_to\_csv(parts, 'tables/parts.csv', ['part\_id', 'material', 'weight\_kg', 'price\_usd', 'parttype\_id', 'quantity\_in\_stock', 'supplier\_id', 'min\_stock\_level', 'is\_active'])

    print('Generating employees...')

    employees = generate\_employees()

    save\_to\_csv(employees, 'tables/employees.csv', ['employee\_id', 'first\_name', 'second\_name', 'last\_name', 'position', 'hire\_date', 'age'])

    print('Generating invoices...')

    invoices = generate\_invoices()

    print('Generating invoice lines...')

    invoice\_lines = generate\_invoice\_lines(invoices, parts)

    print('Generating invoice payments...')

    payments = generate\_payments(invoices)

    save\_to\_csv(invoice\_lines, 'tables/invoice\_lines.csv', ['invoiceline\_id', 'invoice\_id', 'part\_id', 'quantity', 'unit\_price', 'line\_total'])

    save\_to\_csv(payments, 'tables/payments.csv', ['payment\_id', 'invoice\_id', 'payment\_date', 'amount', 'payment\_method'])

    save\_to\_csv(invoices, 'tables/invoices.csv', ['invoice\_id', 'invoice\_date', 'total\_amount', 'customer\_id', 'employee\_id', 'payment\_status'])

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**ПРИЛОЖЕНИЕ В**

**Python-код для создания графиков**

from database import get\_sales\_dynamics, get\_payment\_status\_stats, get\_payments\_vs\_debts

from telebot import types

from commands.graphics.plot import create\_plot

def graphic\_types(bot, message, user\_state):

markup = types.ReplyKeyboardMarkup(resize\_keyboard=True)

markup.row('Динамика продаж за год', 'Статусы оплат')

markup.row('Оплаты и долги компаний', 'Назад')

bot.send\_message(message.chat.id, 'Выберите график:', reply\_markup=markup)

user\_state[message.chat.id] = {'action': 'analytics'}

def plot\_selection(bot, message, menu, user\_state):

try:

if message.text == 'Динамика продаж за год':

data = get\_sales\_dynamics()

plot = create\_plot(data, 'bar', 'Динамика продаж по месяцам', 'Месяц', 'Сумма продаж')

elif message.text == 'Статусы оплат':

data = get\_payment\_status\_stats()

plot = create\_plot(data, 'pie', 'Распределение статусов оплат')

elif message.text == 'Оплаты и долги компаний':

data = get\_payments\_vs\_debts()

plot = create\_plot(data, '2bars', 'Соотношение оплат и долгов по компаниям', 'Компании', 'Долги/оплаты')

bot.send\_photo(message.chat.id, plot, reply\_markup=menu)

except Exception as e:

bot.send\_message(message.chat.id, f'Ошибка:\n<pre>{e}</pre>', parse\_mode="HTML")

finally:

if message.chat.id in user\_state:

del user\_state[message.chat.id]

import matplotlib.pyplot as plt

from io import BytesIO

def create\_plot(data: list, plot\_type: str, title: str, xlabel='', ylabel=''):

plt.switch\_backend('Agg')

fig, ax = plt.subplots()

if plot\_type == 'bar':

labels = [row[0] for row in data]

values = [row[1] for row in data]

ax.bar(labels, values, color='lightgreen')

elif plot\_type == 'pie':

labels = [row[0] for row in data]

sizes = [row[1] for row in data]

ax.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['#FA8072', '#90EE90', '#FFFFCB'])

elif plot\_type == '2bars':

labels = [row[0] for row in data]

values1 = [row[1] for row in data]

values2 = [row[2] for row in data]

ax.bar(labels, values1, color='lightgreen')

ax.bar(labels, values2, bottom=values1, color='salmon')

plt.title(title)

plt.xlabel(xlabel,)

plt.ylabel(ylabel)

plt.xticks(rotation=90, fontsize=8)

plt.tight\_layout()

plt.grid(True, axis='y', linestyle='--', color='black')

buf = BytesIO()

plt.savefig(buf, format='png')

buf.seek(0)

plt.close()

return buf

**ПРИЛОЖЕНИЕ Г**

**Триггеры**

--- Файл: ./triggers\check\_customer\_duplicate.sql ---

CREATE OR REPLACE FUNCTION check\_customer\_duplicate()

RETURNS TRIGGER AS $$

BEGIN

-- Только если изменяется material или customertype\_id

IF (((TG\_OP = 'INSERT') OR (TG\_OP = 'UPDATE')) AND

(OLD.customer\_name IS DISTINCT FROM NEW.customer\_name OR

OLD.city IS DISTINCT FROM NEW.city)) THEN

IF EXISTS (

SELECT 1 FROM customer

WHERE customer\_name = NEW.customer\_name

AND city = NEW.city

AND customer\_id != NEW.customer\_id

) THEN

RAISE EXCEPTION 'Покупатель с названием "%" из города % уже существует',

NEW.customer\_name, NEW.city;

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_check\_customer\_duplicate

BEFORE INSERT ON customer

FOR EACH ROW

EXECUTE FUNCTION check\_customer\_duplicate();

--- Файл: ./triggers\check\_hire\_age.sql ---

CREATE OR REPLACE FUNCTION check\_hire\_age()

RETURNS TRIGGER AS $$

DECLARE

hire\_year INT;

current\_year INT;

age\_at\_hire INT;

BEGIN

-- Получаем текущий год и год найма

current\_year := EXTRACT(YEAR FROM CURRENT\_DATE);

hire\_year := EXTRACT(YEAR FROM NEW.hire\_date);

-- Вычисляем примерный возраст на момент найма

IF hire\_year = current\_year THEN

age\_at\_hire := NEW.age; -- Если наняли в этом году, возраст тот же

ELSE

age\_at\_hire := NEW.age - (current\_year - hire\_year);

END IF;

-- Проверяем, был ли сотрудник совершеннолетним на момент найма

IF age\_at\_hire < 18 THEN

RAISE EXCEPTION

'Сотрудник не мог быть нанят в % году в возрасте % лет (должно быть ≥18)',

hire\_year, age\_at\_hire;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Триггер для INSERT (добавление нового сотрудника)

CREATE TRIGGER trg\_check\_hire\_age\_insert

BEFORE INSERT ON Employee

FOR EACH ROW

EXECUTE FUNCTION check\_hire\_age();

-- Триггер для UPDATE (если изменили hire\_date или age)

CREATE TRIGGER trg\_check\_hire\_age\_update

BEFORE UPDATE ON Employee

FOR EACH ROW

WHEN (OLD.hire\_date IS DISTINCT FROM NEW.hire\_date OR OLD.age IS DISTINCT FROM NEW.age)

EXECUTE FUNCTION check\_hire\_age();

--- Файл: ./triggers\check\_parttype\_duplicate.sql ---

CREATE OR REPLACE FUNCTION check\_parttype\_duplicate()

RETURNS TRIGGER AS $$

BEGIN

-- Только если изменяется material или parttypetype\_id

IF (((TG\_OP = 'INSERT') OR (TG\_OP = 'UPDATE')) AND

(OLD.type\_name IS DISTINCT FROM NEW.type\_name)) THEN

IF EXISTS (

SELECT 1 FROM parttype

WHERE type\_name = NEW.type\_name

AND parttype\_id != NEW.parttype\_id

) THEN

RAISE EXCEPTION 'Тип с названием "%" уже существует',

NEW.type\_name;

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_check\_parttype\_duplicate

BEFORE INSERT ON parttype

FOR EACH ROW

EXECUTE FUNCTION check\_parttype\_duplicate();

--- Файл: ./triggers\check\_part\_duplicate.sql ---

CREATE OR REPLACE FUNCTION check\_part\_duplicate()

RETURNS TRIGGER AS $$

BEGIN

-- Только если изменяется material или parttype\_id

IF (((TG\_OP = 'INSERT') OR (TG\_OP = 'UPDATE')) AND

(OLD.material IS DISTINCT FROM NEW.material OR

OLD.parttype\_id IS DISTINCT FROM NEW.parttype\_id)) THEN

IF EXISTS (

SELECT 1 FROM Part

WHERE material = NEW.material

AND parttype\_id = NEW.parttype\_id

AND part\_id != NEW.part\_id

) THEN

RAISE EXCEPTION 'Деталь с материалом "%" и ID типа % уже существует',

NEW.material, NEW.parttype\_id;

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_check\_part\_duplicate

BEFORE INSERT ON part

FOR EACH ROW

EXECUTE FUNCTION check\_part\_duplicate();

--- Файл: ./triggers\check\_stock\_level.sql ---

CREATE OR REPLACE FUNCTION check\_stock\_level()

RETURNS TRIGGER AS $$

DECLARE

current\_stock INT;

BEGIN

current\_stock := (SELECT quantity\_in\_stock FROM Part WHERE part\_id = NEW.part\_id);

IF current\_stock < NEW.quantity THEN

RAISE EXCEPTION 'Недостаточно товара на складе. Доступно: %, запрошено: %',

current\_stock, NEW.quantity;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER before\_invoice\_line\_insert

BEFORE INSERT ON InvoiceLine

FOR EACH ROW

EXECUTE FUNCTION check\_stock\_level();

--- Файл: ./triggers\check\_supplier\_duplicate.sql ---

CREATE OR REPLACE FUNCTION check\_supplier\_duplicate()

RETURNS TRIGGER AS $$

BEGIN

-- Только если изменяется material или parttype\_id

IF (((TG\_OP = 'INSERT') OR (TG\_OP = 'UPDATE')) AND

(OLD.contact\_phone IS DISTINCT FROM NEW.contact\_phone)) THEN

IF EXISTS (

SELECT 1 FROM supplier

WHERE contact\_phone = NEW.contact\_phone

AND supplier\_id != NEW.supplier\_id

) THEN

RAISE EXCEPTION 'Номер телефона "%" уже занят',

NEW.contact\_phone;

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_check\_supplier\_duplicate

BEFORE INSERT ON supplier

FOR EACH ROW

EXECUTE FUNCTION check\_supplier\_duplicate();

--- Файл: ./triggers\decrease\_stock\_level.sql ---

CREATE OR REPLACE FUNCTION decrease\_stock\_on\_invoiceline\_insert()

RETURNS TRIGGER AS $$

BEGIN

UPDATE Part

SET quantity\_in\_stock = quantity\_in\_stock - NEW.quantity

WHERE part\_id = NEW.part\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_decrease\_stock\_on\_invoiceline\_insert

AFTER INSERT ON InvoiceLine

FOR EACH ROW

EXECUTE FUNCTION decrease\_stock\_on\_invoiceline\_insert();

--- Файл: ./triggers\prevent\_early\_payment.sql ---

CREATE OR REPLACE FUNCTION prevent\_early\_payment()

RETURNS TRIGGER AS $$

DECLARE

invoice\_datee TIMESTAMP;

BEGIN

SELECT invoice\_date INTO invoice\_datee

FROM Invoice

WHERE invoice\_id = NEW.invoice\_id;

IF NEW.payment\_date < invoice\_datee THEN

RAISE EXCEPTION 'Дата платежа не может быть раньше даты накладной';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_prevent\_early\_payment

BEFORE INSERT ON Payment

FOR EACH ROW

EXECUTE FUNCTION prevent\_early\_payment();

--- Файл: ./triggers\prevent\_overpayment.sql ---

CREATE OR REPLACE FUNCTION prevent\_overpayment()

RETURNS TRIGGER AS $$

DECLARE

current\_paid DECIMAL := 0;

invoice\_total DECIMAL := 0;

BEGIN

-- Сумма уже внесённых платежей

SELECT COALESCE(SUM(amount), 0)

INTO current\_paid

FROM Payment

WHERE invoice\_id = NEW.invoice\_id;

-- Общая сумма накладной

SELECT total\_amount

INTO invoice\_total

FROM Invoice

WHERE invoice\_id = NEW.invoice\_id;

-- Если платеж превысит допустимую сумму

IF current\_paid + NEW.amount > invoice\_total THEN

RAISE EXCEPTION 'Платёж приведет к переплате по накладной';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_prevent\_overpayment

BEFORE INSERT ON Payment

FOR EACH ROW

EXECUTE FUNCTION prevent\_overpayment();

--- Файл: ./triggers\prevent\_payment\_on\_paid.sql ---

CREATE OR REPLACE FUNCTION prevent\_payment\_on\_paid()

RETURNS TRIGGER AS $$

BEGIN

IF (

SELECT COALESCE(SUM(amount), 0)

FROM Payment

WHERE invoice\_id = NEW.invoice\_id

) >= (

SELECT total\_amount

FROM Invoice

WHERE invoice\_id = NEW.invoice\_id

) THEN

RAISE EXCEPTION 'Накладная уже полностью оплачена';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_prevent\_paymenton\_paid

BEFORE INSERT ON Payment

FOR EACH ROW

EXECUTE FUNCTION prevent\_payment\_on\_paid();

--- Файл: ./triggers\reuse\_customer\_id.sql ---

CREATE OR REPLACE FUNCTION reuse\_customer\_id()

RETURNS TRIGGER AS $$

BEGIN

-- Пытаемся использовать минимальный свободный ID

SELECT MIN(customer\_id) + 1 INTO NEW.customer\_id

FROM customer p1

WHERE NOT EXISTS (

SELECT 1 FROM customer p2 WHERE p2.customer\_id = p1.customer\_id + 1

);

-- Если не нашли, используем стандартное поведение

IF NEW.customer\_id IS NULL THEN

NEW.customer\_id := nextval('customer\_customer\_id\_seq');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_reuse\_customer\_id

BEFORE INSERT ON customer

FOR EACH ROW

EXECUTE FUNCTION reuse\_customer\_id();

--- Файл: ./triggers\reuse\_employee\_id.sql ---

CREATE OR REPLACE FUNCTION reuse\_employee\_id()

RETURNS TRIGGER AS $$

BEGIN

-- Пытаемся использовать минимальный свободный ID

SELECT MIN(employee\_id) + 1 INTO NEW.employee\_id

FROM employee p1

WHERE NOT EXISTS (

SELECT 1 FROM employee p2 WHERE p2.employee\_id = p1.employee\_id + 1

);

-- Если не нашли, используем стандартное поведение

IF NEW.employee\_id IS NULL THEN

NEW.employee\_id := nextval('employee\_employee\_id\_seq');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_reuse\_employee\_id

BEFORE INSERT ON employee

FOR EACH ROW

EXECUTE FUNCTION reuse\_employee\_id();

--- Файл: ./triggers\reuse\_invoiceline\_id.sql ---

CREATE OR REPLACE FUNCTION reuse\_invoiceline\_id()

RETURNS TRIGGER AS $$

BEGIN

-- Пытаемся использовать минимальный свободный ID

SELECT MIN(invoiceline\_id) + 1 INTO NEW.invoiceline\_id

FROM invoiceline p1

WHERE NOT EXISTS (

SELECT 1 FROM invoiceline p2 WHERE p2.invoiceline\_id = p1.invoiceline\_id + 1

);

-- Если не нашли, используем стандартное поведение

IF NEW.invoiceline\_id IS NULL THEN

NEW.invoiceline\_id := nextval('invoiceline\_invoiceline\_id\_seq');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_reuse\_invoiceline\_id

BEFORE INSERT ON invoiceline

FOR EACH ROW

EXECUTE FUNCTION reuse\_invoiceline\_id();

--- Файл: ./triggers\reuse\_invoice\_id.sql ---

CREATE OR REPLACE FUNCTION reuse\_invoice\_id()

RETURNS TRIGGER AS $$

BEGIN

-- Пытаемся использовать минимальный свободный ID

SELECT MIN(invoice\_id) + 1 INTO NEW.invoice\_id

FROM invoice p1

WHERE NOT EXISTS (

SELECT 1 FROM invoice p2 WHERE p2.invoice\_id = p1.invoice\_id + 1

);

-- Если не нашли, используем стандартное поведение

IF NEW.invoice\_id IS NULL THEN

NEW.invoice\_id := nextval('invoice\_invoice\_id\_seq');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_reuse\_invoice\_id

BEFORE INSERT ON invoice

FOR EACH ROW

EXECUTE FUNCTION reuse\_invoice\_id();

--- Файл: ./triggers\reuse\_parttype\_id.sql ---

CREATE OR REPLACE FUNCTION reuse\_parttype\_id()

RETURNS TRIGGER AS $$

BEGIN

-- Пытаемся использовать минимальный свободный ID

SELECT MIN(parttype\_id) + 1 INTO NEW.parttype\_id

FROM parttype p1

WHERE NOT EXISTS (

SELECT 1 FROM parttype p2 WHERE p2.parttype\_id = p1.parttype\_id + 1

);

-- Если не нашли, используем стандартное поведение

IF NEW.parttype\_id IS NULL THEN

NEW.parttype\_id := nextval('parttype\_parttype\_id\_seq');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_reuse\_parttype\_id

BEFORE INSERT ON parttype

FOR EACH ROW

EXECUTE FUNCTION reuse\_parttype\_id();

--- Файл: ./triggers\reuse\_part\_id.sql ---

CREATE OR REPLACE FUNCTION reuse\_part\_id()

RETURNS TRIGGER AS $$

BEGIN

-- Пытаемся использовать минимальный свободный ID

SELECT MIN(part\_id) + 1 INTO NEW.part\_id

FROM Part p1

WHERE NOT EXISTS (

SELECT 1 FROM Part p2 WHERE p2.part\_id = p1.part\_id + 1

);

-- Если не нашли, используем стандартное поведение

IF NEW.part\_id IS NULL THEN

NEW.part\_id := nextval('part\_part\_id\_seq');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_reuse\_part\_id

BEFORE INSERT ON Part

FOR EACH ROW

EXECUTE FUNCTION reuse\_part\_id();

--- Файл: ./triggers\reuse\_payment\_id.sql ---

CREATE OR REPLACE FUNCTION reuse\_payment\_id()

RETURNS TRIGGER AS $$

BEGIN

-- Пытаемся использовать минимальный свободный ID

SELECT MIN(payment\_id) + 1 INTO NEW.payment\_id

FROM payment p1

WHERE NOT EXISTS (

SELECT 1 FROM payment p2 WHERE p2.payment\_id = p1.payment\_id + 1

);

-- Если не нашли, используем стандартное поведение

IF NEW.payment\_id IS NULL THEN

NEW.payment\_id := nextval('payment\_payment\_id\_seq');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_reuse\_payment\_id

BEFORE INSERT ON payment

FOR EACH ROW

EXECUTE FUNCTION reuse\_payment\_id();

--- Файл: ./triggers\reuse\_supplier\_id.sql ---

CREATE OR REPLACE FUNCTION reuse\_supplier\_id()

RETURNS TRIGGER AS $$

BEGIN

-- Пытаемся использовать минимальный свободный ID

SELECT MIN(supplier\_id) + 1 INTO NEW.supplier\_id

FROM supplier p1

WHERE NOT EXISTS (

SELECT 1 FROM supplier p2 WHERE p2.supplier\_id = p1.supplier\_id + 1

);

-- Если не нашли, используем стандартное поведение

IF NEW.supplier\_id IS NULL THEN

NEW.supplier\_id := nextval('supplier\_supplier\_id\_seq');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_reuse\_supplier\_id

BEFORE INSERT ON supplier

FOR EACH ROW

EXECUTE FUNCTION reuse\_supplier\_id();

--- Файл: ./triggers\update\_active\_status.sql ---

CREATE OR REPLACE FUNCTION update\_part\_active\_status()

RETURNS TRIGGER AS $$

DECLARE

min\_stock INT;

BEGIN

SELECT min\_stock\_level INTO min\_stock

FROM Part

WHERE part\_id = NEW.part\_id;

IF NEW.quantity\_in\_stock < min\_stock THEN

NEW.is\_active := FALSE;

ELSE

NEW.is\_active := TRUE;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_update\_part\_active\_status

BEFORE UPDATE OF quantity\_in\_stock ON Part

FOR EACH ROW

EXECUTE FUNCTION update\_part\_active\_status();

--- Файл: ./triggers\update\_invoice\_payment\_status.sql ---

CREATE OR REPLACE FUNCTION update\_invoice\_payment\_status()

RETURNS TRIGGER AS $$

DECLARE

total\_paid DECIMAL := 0;

total\_due DECIMAL := 0;

BEGIN

SELECT COALESCE(SUM(amount), 0) INTO total\_paid

FROM Payment

WHERE invoice\_id = NEW.invoice\_id;

SELECT total\_amount INTO total\_due

FROM Invoice

WHERE invoice\_id = NEW.invoice\_id;

UPDATE Invoice

SET payment\_status = CASE

WHEN total\_paid = 0 THEN 'Неоплачено'

WHEN total\_paid < total\_due THEN 'Частично оплачено'

WHEN total\_paid >= total\_due THEN 'Оплачено'

END

WHERE invoice\_id = NEW.invoice\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_update\_invoice\_status

AFTER INSERT ON Payment

FOR EACH ROW

EXECUTE FUNCTION update\_invoice\_payment\_status();

--- Файл: ./triggers\update\_invoice\_total.sql ---

CREATE OR REPLACE FUNCTION update\_invoice\_total()

RETURNS TRIGGER AS $$

BEGIN

UPDATE Invoice

SET total\_amount = (

SELECT COALESCE(SUM(line\_total), 0)

FROM InvoiceLine

WHERE invoice\_id = NEW.invoice\_id

)

WHERE invoice\_id = NEW.invoice\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_invoiceline\_insert

AFTER INSERT ON InvoiceLine

FOR EACH ROW

EXECUTE FUNCTION update\_invoice\_total();

CREATE TRIGGER trg\_invoiceline\_update

AFTER UPDATE ON InvoiceLine

FOR EACH ROW

EXECUTE FUNCTION update\_invoice\_total();

CREATE TRIGGER trg\_invoiceline\_delete

AFTER DELETE ON InvoiceLine

FOR EACH ROW

EXECUTE FUNCTION update\_invoice\_total();

**ПРИЛОЖЕНИЕ Д**

**Код с описанием поведения бота**

from imports import \*

from datetime import datetime as dt

bot = telebot.TeleBot(bot\_settings['Token'])

user\_state = {}

actions = ['Поиск записи', 'Добавить запись', 'Удалить запись',

'Посмотреть накладные за период', 'Графики аналитики',

'Самые активные покупатели', 'Самые продаваемые детали',

'Самые ценные сотрудники', 'Задолженности',

'Какие детали пополнить']

menu = types.ReplyKeyboardMarkup(resize\_keyboard=True)

for i in actions:

menu.add(i)

@bot.message\_handler(commands=['start'])

def handle\_start(message):

start(bot, message, menu)

@bot.message\_handler(commands=['documents'])

def handle\_docs(message):

bot.send\_message(message.chat.id, 'Скоро здесь будут документы :)', reply\_markup=menu)

@bot.message\_handler(func=lambda msg: msg.text == 'Поиск записи')

def handle\_search\_record(message):

search\_record(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.chat.id in user\_state and user\_state[msg.chat.id]['action'] == 'search\_type' and msg.text in ['Поиск по ID', 'Поиск по содержимому'])

def handle\_search\_type(message):

search\_type(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.chat.id in user\_state and user\_state[msg.chat.id]['action'] == 'search\_table' and msg.text in ['Накладная', 'Строка накладной', 'Поставщик', 'Покупатель', 'Деталь', 'Тип детали', 'Сотрудник', 'Платеж'])

def handle\_search\_table(message):

search\_table(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.chat.id in user\_state and user\_state[msg.chat.id].get('action') in ['search\_table'] and 'table' in user\_state[msg.chat.id])

def handle\_search\_query(message):

search\_query(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.text == 'Добавить запись')

def handle\_add\_record(message):

add\_record(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.text == 'Удалить запись')

def handle\_delete\_record(message):

delete\_record(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.text == 'Назад')

def handle\_return\_back(message):

return\_back(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.text in ['Накладная', 'Строка накладной', 'Поставщик', 'Покупатель', 'Деталь', 'Тип детали', 'Сотрудник', 'Платеж'])

def handle\_table\_selection(message):

table\_selection(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.chat.id in user\_state and user\_state[msg.chat.id]['action'] == 'add' and 'table' in user\_state[msg.chat.id])

def handle\_data\_input(message):

data\_input(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.chat.id in user\_state and user\_state[msg.chat.id]['action'] == 'delete' and 'table' in user\_state[msg.chat.id])

def handle\_delete\_data(message):

delete\_data(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.text == 'Посмотреть накладные за период')

def handle\_ask\_period(message):

ask\_period(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.chat.id in user\_state and user\_state[msg.chat.id]['action'] == 'watch')

def handle\_show\_invoices(message):

show\_invoices(bot, message, menu, user\_state)

@bot.message\_handler(func=lambda msg: msg.text == 'Какие детали пополнить')

def handle\_check\_fill(message):

checkfill(bot, message, menu)

@bot.message\_handler(func=lambda msg: msg.text == 'Самые активные покупатели')

def handle\_mv\_customers(message):

mv\_customers(bot, message, menu)

@bot.message\_handler(func=lambda msg: msg.text == 'Самые продаваемые детали')

def handle\_ms\_parts(message):

ms\_parts(bot, message, menu)

@bot.message\_handler(func=lambda msg: msg.text == 'Самые ценные сотрудники')

def handle\_mv\_employees(message):

mv\_employee(bot, message, menu)

@bot.message\_handler(func=lambda msg: msg.text == 'Задолженности')

def handle\_most\_due(message):

mst\_due(bot, message, menu)

@bot.message\_handler(func=lambda msg: msg.text == 'Графики аналитики')

def handle\_graphic\_types(message):

graphic\_types(bot, message, user\_state)

@bot.message\_handler(func=lambda msg: msg.chat.id in user\_state and user\_state[msg.chat.id]['action'] == 'analytics' and msg.text in ['Динамика продаж за год', 'Статусы оплат', 'Оплаты и долги компаний'])

def handle\_plot\_selection(message):

plot\_selection(bot, message, menu, user\_state)

@bot.message\_handler(content\_types=['text', 'audio', 'photo', 'video', 'sticker', 'location', 'voice', 'document', 'contact'])

def on\_flood(message):

bot.send\_message(message.chat.id, 'Выберите действие:', reply\_markup=menu)

print(f'Бот запущен {dt.now()}')

bot.polling(none\_stop=True)