



**«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-ПОЯСНИТЕЛЬНАЯ 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)

**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.

**З А Д А Н И Е
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 связанных 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

Введение.....	5
1 Анализ предметной области	6
2 Разработка бизнес-процессов.....	7
3 Проектирование базы данных.....	8
3.1 Выделение главной сущности	8
3.2 Описание таблиц	8
3.4 Схема базы данных	12
4 Заполнение таблиц	13
4.1 Создание базы данных.....	13
4.2 Заполнение таблиц	13
5 Разработка запросов.....	14
6 Разработка интерфейса	23
Заключение	28
Приложение А	29
Приложение Б	32
Приложение В.....	42
Приложение Г	44
Приложение Д.....	58

ВВЕДЕНИЕ

В данной работе создается являющаяся 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.

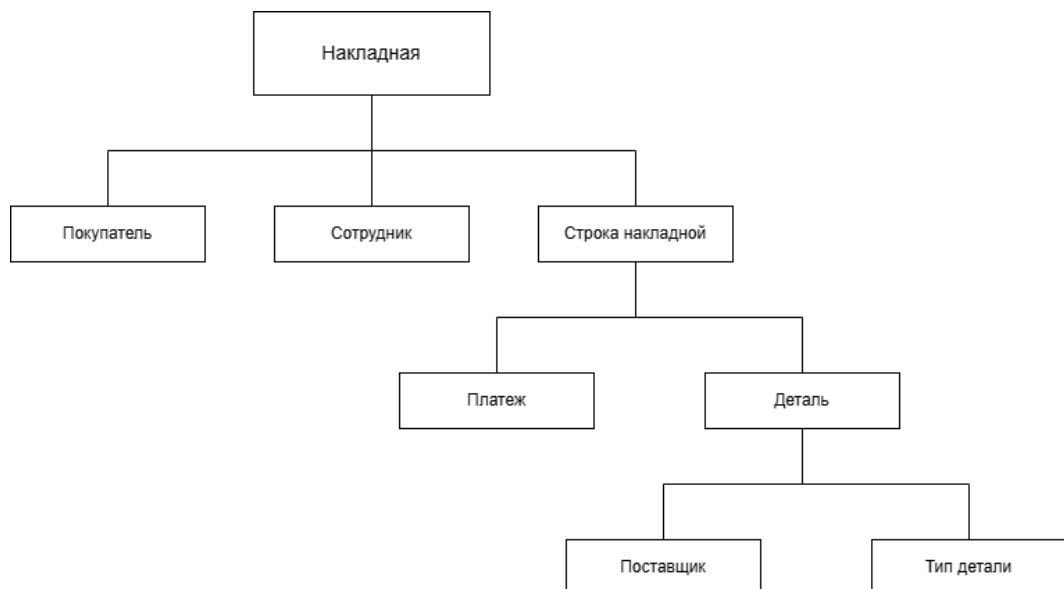


Рисунок 1 - Схема связи

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.

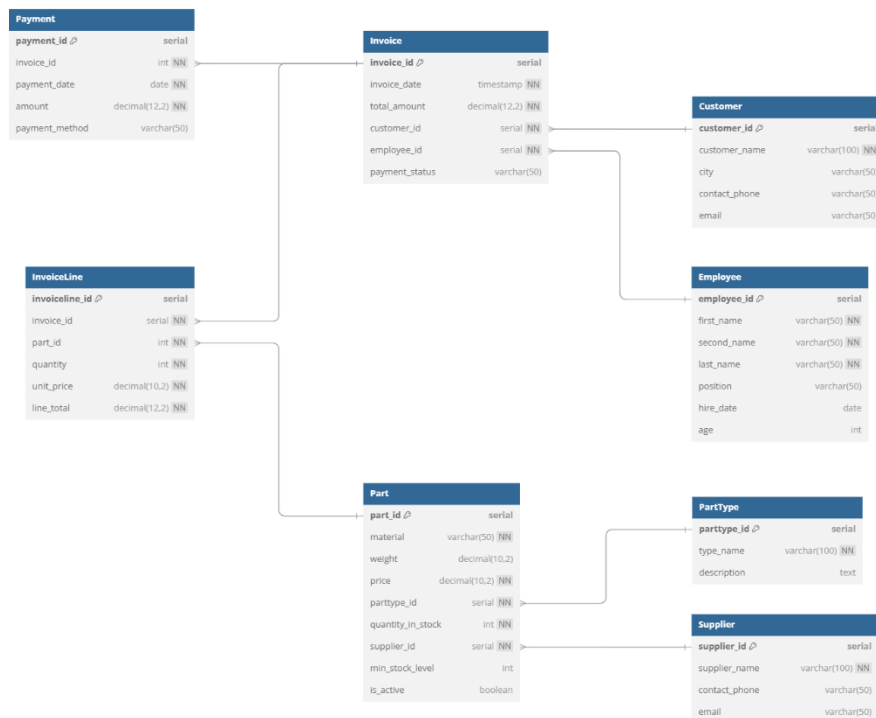


Рисунок 2 - Схема базы данных

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:

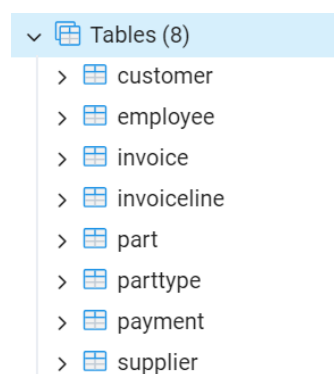


Рисунок 3 - Созданные таблицы

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.

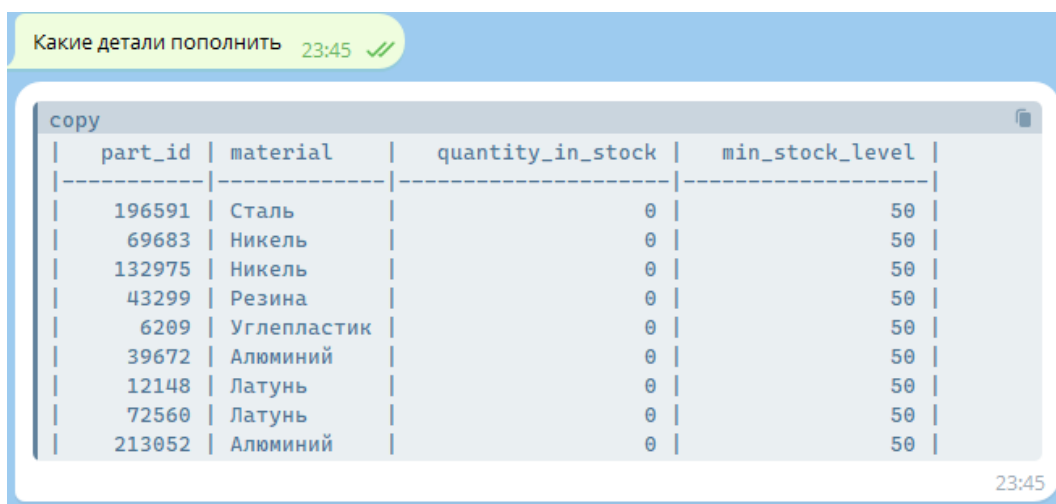
5 Разработка запросов

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:



Какие детали пополнить 23:45 ✓

copy

part_id	material	quantity_in_stock	min_stock_level
196591	Сталь	0	50
69683	Никель	0	50
132975	Никель	0	50
43299	Резина	0	50
6209	Углепластик	0	50
39672	Алюминий	0	50
12148	Латунь	0	50
72560	Латунь	0	50
213052	Алюминий	0	50

23:45

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:

Самые активные покупатели 23:46 ✓

copy

customer_id	customer	sold
89	Haynes International	31
147	DAF Trucks	30
21	Tenneco	30
187	UAZ	29
180	Marcopolo	29
57	Barrick Gold	29
126	JLG Industries	28
13	Continental AG	28
101	Tenova	27
9	BAE Systems	27

23:46

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:

Самые продаваемые детали 23:46 ✓✓

copy

part_id	material	type_name	summ
496131	Никель	Решетка армированная	246613
135034	Керамика	Штуцер фрикционная	246030
582616	Алюминий	Крышка крепежный	238932
870053	Графит	Решетка упорный	222645
155598	Углепластик	Муфта уплотнительная	222496
332219	Керамика	Подшипник клиновая	221199
381336	Бронза	Болт защитная	218562
652874	Керамика	Муфта крепежная	217947
494121	Углепластик	Пластина стопорный	217771
538016	Резина	Штуцер клиновая	214419

23:46

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:

Самые ценные сотрудники 23:46 ✓

full_name	invoices_total	average	total_amount
Лобанов Даниил Егорович	1577	58271.9	9.18948e+07
Котов Павел Александрович	1538	59362.8	9.12999e+07
Щукин Алексей Алексеевич	1568	58146.1	9.1173e+07
Дорофеев Иван Артемович	1554	58009.7	9.01471e+07
Антонов Александр Иванович	1554	57801	8.98227e+07
Громов Дмитрий Арсениевич	1560	57493.9	8.96905e+07
Горшков Иван Алексеевич	1525	58537.2	8.92692e+07
Горбачев Максим Геннадьевич	1544	57689.4	8.90725e+07
Лаврентьев Максим Даниилович	1557	57191.8	8.90476e+07
Буров Арсений Евгениевич	1558	57134.8	8.9016e+07

23:46

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:

Задолженности 23:47 ✓

copy

customer_id	customer_name	invoice_count	total_billed	total_paid	total_due
222	China North Industries Group, Пекин	1406	8.30239e+07	6.21636e+07	2.08603e+07
117	JCB, Рочестер	1441	8.42808e+07	6.4995e+07	1.92858e+07
99	Ternium, Сан-Николас-де-лос-Гарса	1410	7.94359e+07	6.07894e+07	1.86465e+07
82	Aichi Steel, Токай	1439	8.13338e+07	6.27674e+07	1.85664e+07
74	Kyocera, Киото	1313	7.46218e+07	5.6546e+07	1.80758e+07
180	Marcopolo, Кашнас-ду-Сул	1370	7.78112e+07	5.97879e+07	1.80233e+07
166	KAMAZ, Набережные Челны	1417	8.03319e+07	6.23832e+07	1.79487e+07
157	Isuzu Motors, Токио	1364	7.49237e+07	5.71231e+07	1.78007e+07
160	Tata Motors, Мумбаи	1371	7.81871e+07	6.04152e+07	1.77719e+07
120	Terex, Норуолк	1339	7.50788e+07	5.73245e+07	1.77543e+07

23:47

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:

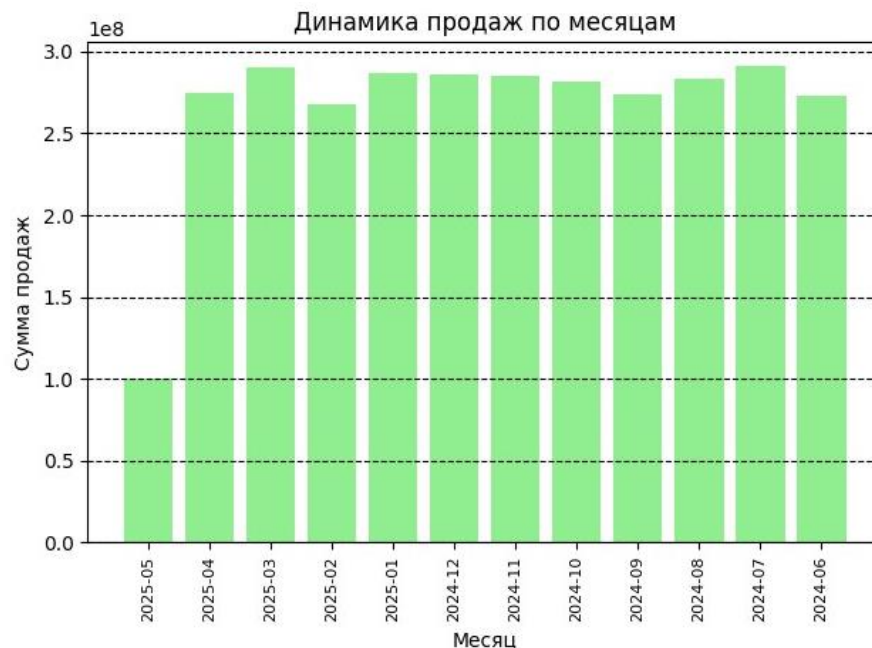


Рисунок 9 - Диаграмма динамики продаж

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:

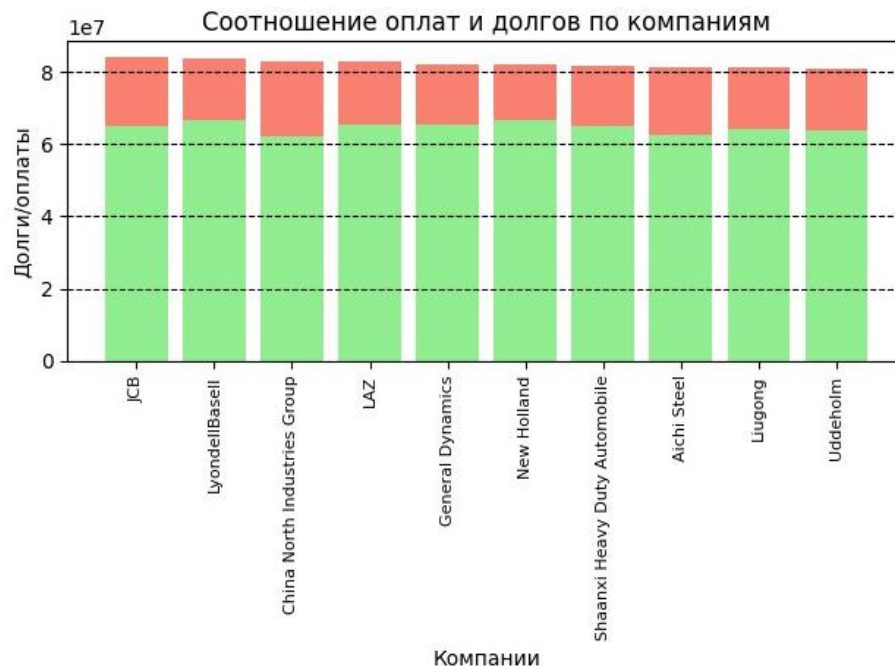


Рисунок 10 - Диаграмма задолженностей

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:

Какие детали пополнить 19:23 ✓

copy

part_id	material	quantity_in_stock	min_stock_level
196591	Сталь	0	50
751663	Резина	0	50
12148	Латунь	0	50
233449	Керамика	0	50
69683	Никель	0	50
144410	Чугун	0	50
420785	Резина	0	50
229656	Железо	0	50
841575	Титан	0	50
43299	Резина	0	50
258357	Сталь	0	50
132975	Никель	0	50
213052	Алюминий	0	50
39672	Алюминий	0	50
6209	Углепластик	0	50
72560	Латунь	0	50
462036	Титан	0	50
989942	Алюминий	0	50
816007	Алюминий	0	50
812959	Чугун	0	50
993959	Графит	0	50
645162	Никель	0	50
681062	Керамика	0	50
503271	Графит	0	50
797315	Чугун	0	50
983839	Алюминий	0	50
767775	Сталь	0	50
723355	Алюминий	0	50
936466	Алюминий	1	50

19:23

Что_пополнить.txt

2.0 MB

19:23

Что_пополнить.md

2.0 MB

19:23

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”:

Что_пополнить.txt – Блокнот

part_id	material	quantity_in_stock	min_stock_level
797315	Чугун	0	50
12148	Латунь	0	50
812959	Чугун	0	50
681062	Керамика	0	50
144410	Чугун	0	50
993959	Графит	0	50
420785	Резина	0	50
841575	Титан	0	50
132975	Никель	0	50
6209	Углепластик	0	50
69683	Никель	0	50
303837	Алюминий	0	50

Рисунок 13 - Содержимое файла txt

Что_пополнить

part_id	material	quantity_in_stock	min_stock_level
797315	Чугун	0	50
12148	Латунь	0	50
812959	Чугун	0	50
681062	Керамика	0	50
144410	Чугун	0	50
993959	Графит	0	50
420785	Резина	0	50
841575	Титан	0	50

Рисунок 14 - Содержимое файла md

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

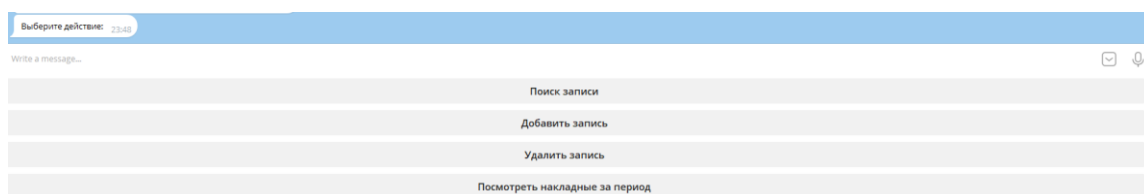


Рисунок 15 - Меню

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-Донатто-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-делос-Гарса', 'Милан', 'Милан', 'Буттрио', 'Дюссельдорф', 'Лондон', 'Трац', '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', 'Линь', 'Чжэнчжоу',
 'Цзинань', 'Чанчунь', 'Ухань', 'Сиань', 'Чунцин', 'Турин', 'Гетеборг',
 'Эйндховен', 'Сен-Приест', 'Штутгарт', 'Портленд', 'Портленд', 'Киркланд',
 'Дентон', 'Гринсборо', 'Лайл', 'Токио', 'Токио', 'Агео', 'Кавасаки', 'Мумбаи',
 'Ченнаи', 'Тургаон', 'Мумбаи', 'Набережные Челны', 'Нижний Новгород',
 'Набережные Челны', 'Минск', 'Кременчуг', 'Жодио', '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',
 'Жодио', 'Могилев', 'Кременчуг', 'Львов', 'Черкаскы', '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', 'gmh.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',
        'регулирующ', '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', 'Camozzì', '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', 'gmxx.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',
'Tитан', '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',
'Михайлович', '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',
'Sумма 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 parttype_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_date 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;
END;
```

```
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)

```