Федеральное государственное бюджетное образовательное учреждение высшего образования

«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

НА ТЕМУ:

БАЗА ДАННЫХ: «Заказ продуктов»

Студент группы ИУ6-44Б _____ Р.Р. Хаялиев
(Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ 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.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине ____Базы данных_____

Студент группы ____ИУ6-44_____

_____ Хаялиев Раниль Рустемович_____
(Фамилия, имя, отчество)

Тема курсовой работы «Заказ продуктов»

Направленность КР учебная

Источник тематики кафедра

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 «Заказ продуктов», 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 _____ ___Р.Р.Хаялиев_____
(Подпись, дата) (И.О. Фамилия)

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

Содержание

ВВЕДЕНИЕ

Разрабатываемая информационная система предназначена для автоматизации процессов онлайн-заказа продуктов. База данных обеспечивает эффективное управление ассортиментом товаров, формирование корзин покупок, оформление заказов, контроль остатков на складе и взаимодействие с клиентами. Система позволяет клиентам выбирать продукты, формировать заказы, отслеживать их статус и получать уведомления о ключевых этапах обработки.

Актуальность разработки обусловлена ростом спроса на онлайн-покупки продуктов, особенно в условиях цифровизации рынка. Ручное управление ассортиментом, заказами и остатками становится неэффективным и приводит к ошибкам. Автоматизация процессов позволит: сократить время обработки заказов, минимизировать риски нехватки товаров, повысить точность расчета стоимости и веса заказов, улучшить клиентский опыт за счет прозрачности и удобства.

Для реализации используется реляционная база данных PostgerSQL и язык программирования Python.

1 АНАЛИЗ ТРЕБОВАНИЙ

1.1 Анализ предметной области

Предметной областью проектируемой базы данных является онлайн-заказ продуктов. Система должна обеспечивать управление ассортиментом товаров, формирование корзин, оформление заказов, контроль остатков на складах, взаимодействие с клиентами и курьерами, а также обработку платежей. Для корректной работы базы данных необходимо хранить информацию о клиентах, продуктах, магазинах, заказах, статусах доставки и уведомлениях.

1.2 Разработка бизнес-процессов предметной области

Процесс работы приложения начинается с формирования клиентом заказа через интерфейс. Клиент выбирает продукты, после чего система автоматически проверяет их наличие в выбранном магазине. Если товары отсутствуют, клиент получает уведомление "Сообщение об отсутствии продуктов", и заказ не создается или изменяется клиентов. Если продукты доступны, формируется корзина.

На этом этапе система переходит к поиску курьера. Если есть свободные курьеры, то выбирается наиболее оптимальный среди них и заказ упаковывается, затем курьер получает задание на доставку. Если свободных курьеров нет, то запускается таймер ожидания, по истечению, которого клиенту отправляется "Сообщение об отсутствии курьеров" и промокод пользователю. Если курьер находится до завершения таймера, то заказ отдается свободному доставщику. Также пользователь может выбрать опцию дождаться свободного курьера.

После назначения курьера магазин приступает к сборке заказа. Здесь происходит финальная проверка наличия товаров (на случай, если их забрали другие клиенты). Если продукты по-прежнему доступны, курьер забирает упакованный заказ и отправляется к клиенту. Если товары внезапно закончились, клиент снова получает уведомление об отсутствии продуктов, и заказ отменяется.

Курьер доставляет заказ клиенту, который подтверждает получение и производит оплату через приложение. После этого заказ закрывается в системе. Если клиент не забирает заказ вовремя (например, при истечении таймера), система отправляет напоминание или автоматически отменяет заказ.

Параллельно происходит взаимодействие с магазинами: на этапе сборки заказа координатор связывается с каждым магазином для уточнения деталей (например, замены товаров или уточнения сроков). Это этап "Собраться с каждым магазином", где решаются технические и организационные вопросы. Весь процесс можно увидеть на рисунке 1.

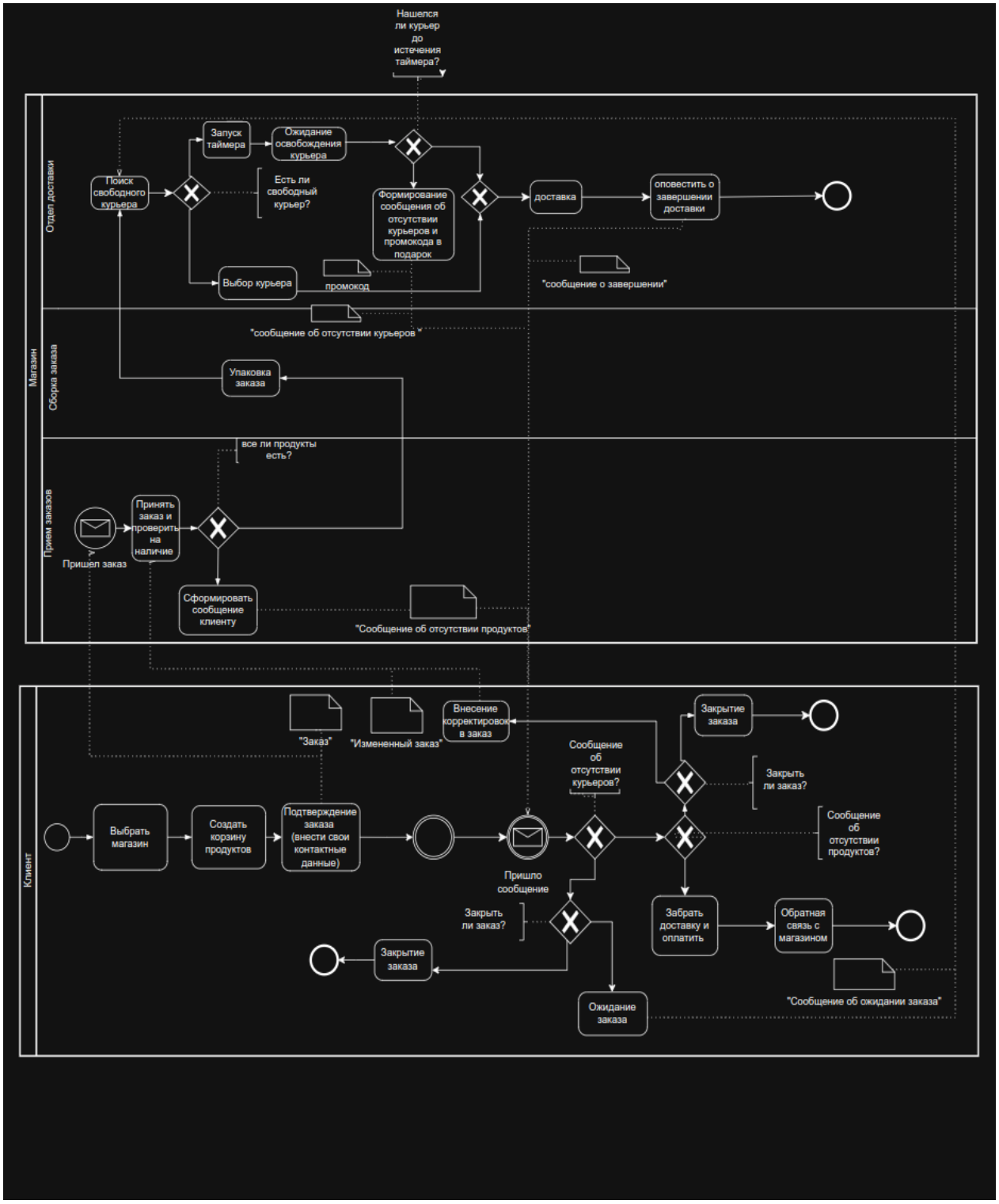


Рисунок 1: BPMN - диаграмма

2. ПРОЕКТИРОВАНИЕ

2.1 Выделение сущностей

Для проектирования базы данных были выделены следующие сущности:

2.1.1 Person (Персональные данные)

Сущность хранит общие персональные данные пользователей системы: идентификатор, серию и номер паспорта, дату рождения, ФИО и контактную информацию. Каждая запись в этой таблице может быть связана с клиентом или курьером через соответствующие внешние ключи. Ограничения включают проверку формата паспортных данных (серия — 4 цифры, номер — 6 цифр).

2.1.2 Client (Клиент)

Клиент представляет собой пользователя, который оформляет заказы. Сущность содержит идентификатор клиента и ссылку на персональные данные из таблицы Person. Каждый клиент может иметь несколько банковских карт и создавать множество заказов. Ограничение: клиент должен быть старше 18 лет (проверяется триггером при создании заказа).

2.1.3 Courier (Курьер)

Курьер отвечает за доставку заказов. Сущность включает идентификатор курьера и ссылку на персональные данные из таблицы Person. Каждый курьер может быть назначен на несколько доставок.

2.1.4 Card (Банковская карта)

Сущность хранит данные банковских карт клиентов. Включает идентификатор карты, ссылку на клиента через таблицу Person, номер карты, срок действия и CVC-код. Ограничения: проверка срока действия карты перед подтверждением заказа.

2.1.5 Store (Магазин)

Магазин описывает торговую точку, где хранятся товары. Атрибуты: идентификатор магазина, адрес и название. Каждый магазин связан с продуктами, которые в нем доступны, и заказами, которые из него отправляются.

2.1.6 Product (Продукт)

Сущность содержит информацию о товарах: идентификатор продукта, ссылку на магазин, стоимость, название, вес и количество на складе. Ограничение: автоматическое уменьшение остатка при добавлении товара в корзину.

2.1.7 Basket (Корзина)

Корзина временно хранит выбранные клиентом товары до оформления заказа. Атрибуты: идентификатор корзины и дата её создания. Каждая корзина связана с клиентом и может содержать несколько элементов.

2.1.8 Item (Элементкорзины)

Элемент корзины описывает конкретный товар и его количество. Связан с корзиной и продуктом. Ограничение: проверка наличия достаточного количества товара на складе перед добавлением в заказ.

2.1.9 Order (Заказ)

Заказ является центральной сущностью системы. Содержит идентификатор заказа, ссылки на клиента, корзину, магазин и статус заказа, а также дату создания, адрес доставки, общую стоимость и вес. Ограничения: автоматический расчет стоимости и веса на основе корзины.

2.1.10 Order Status (Статус заказа)

Сущность определяет возможные статусы заказа (например, «Создан», «Оплачен», «Доставлен»). Каждый статус имеет уникальное название.

2.1.11 Delivery (Доставка)

Доставка связывает заказ с курьером и статусом оплаты. Атрибуты: идентификатор доставки, ссылки на заказ, курьера и статус, а также флаг оплаты.

2.1.12 Message (Сообщение)

Сущность хранит уведомления для клиентов, курьеров и магазинов. Включает тип сообщения (например, «Подтверждение заказа»), текст и ссылки на связанные сущности. Главной сущностью в системе является заказ.

2.2 Проектирование инфологической модели базы данных

Исходя из сущностей и их свойств, определенных в пункте 2.1, была построена инфологическая модель базы данных. Эта модель отражает ключевые процессы взаимодействия между участниками системы и обеспечивает корректное проектирование структуры данных. Инфологическая модель представлена на рисунке 2.

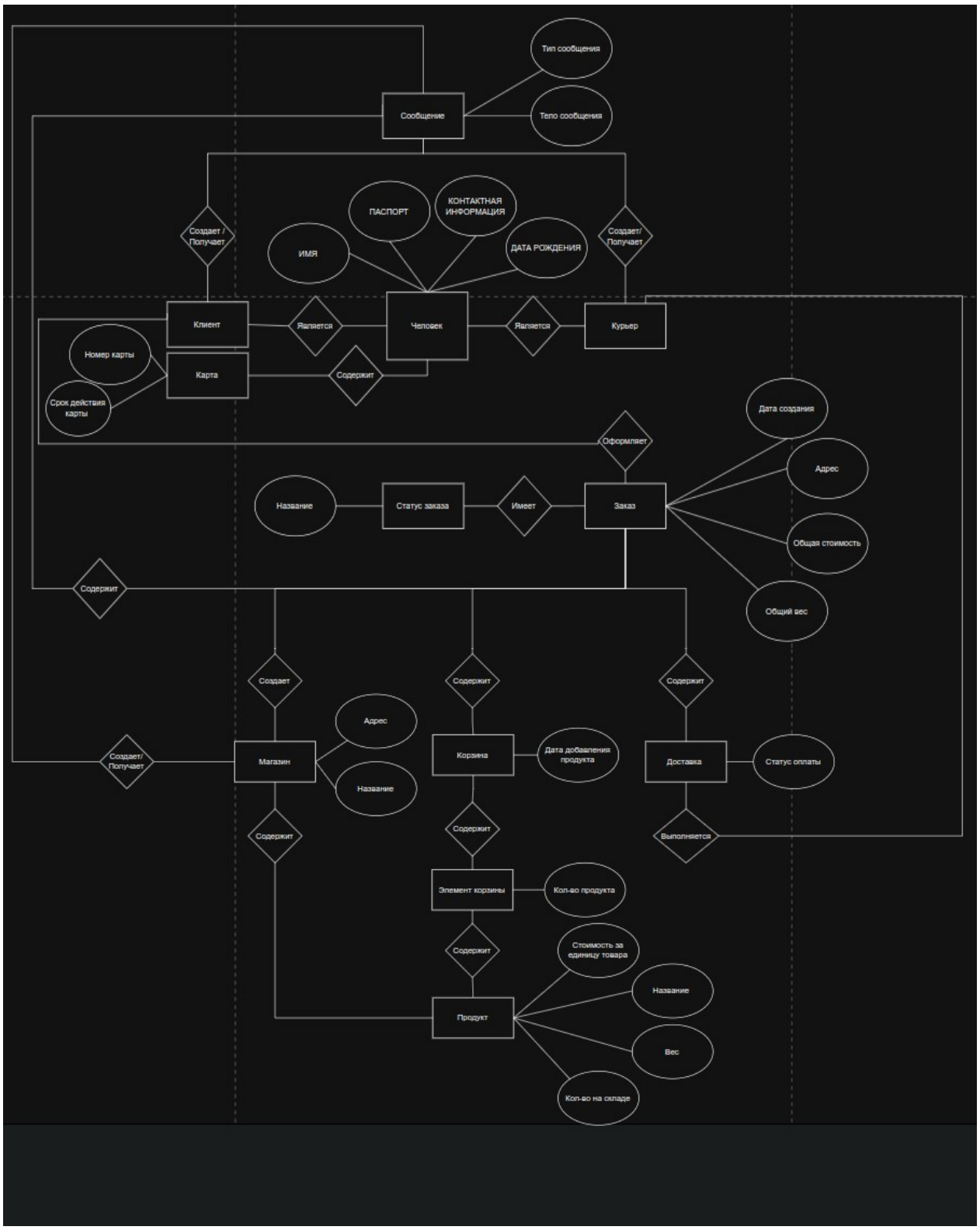


Рисунок 2 — Инфологическая модель базы данных

2.3 Проектирование датологической модели базы данных

На основании инфологической модели можно построить даталогическую модель, необходимую для написания корректного скрипта создания базы данных и входящих в нее таблиц. Даталогическая модель представлена на рисунке 2.

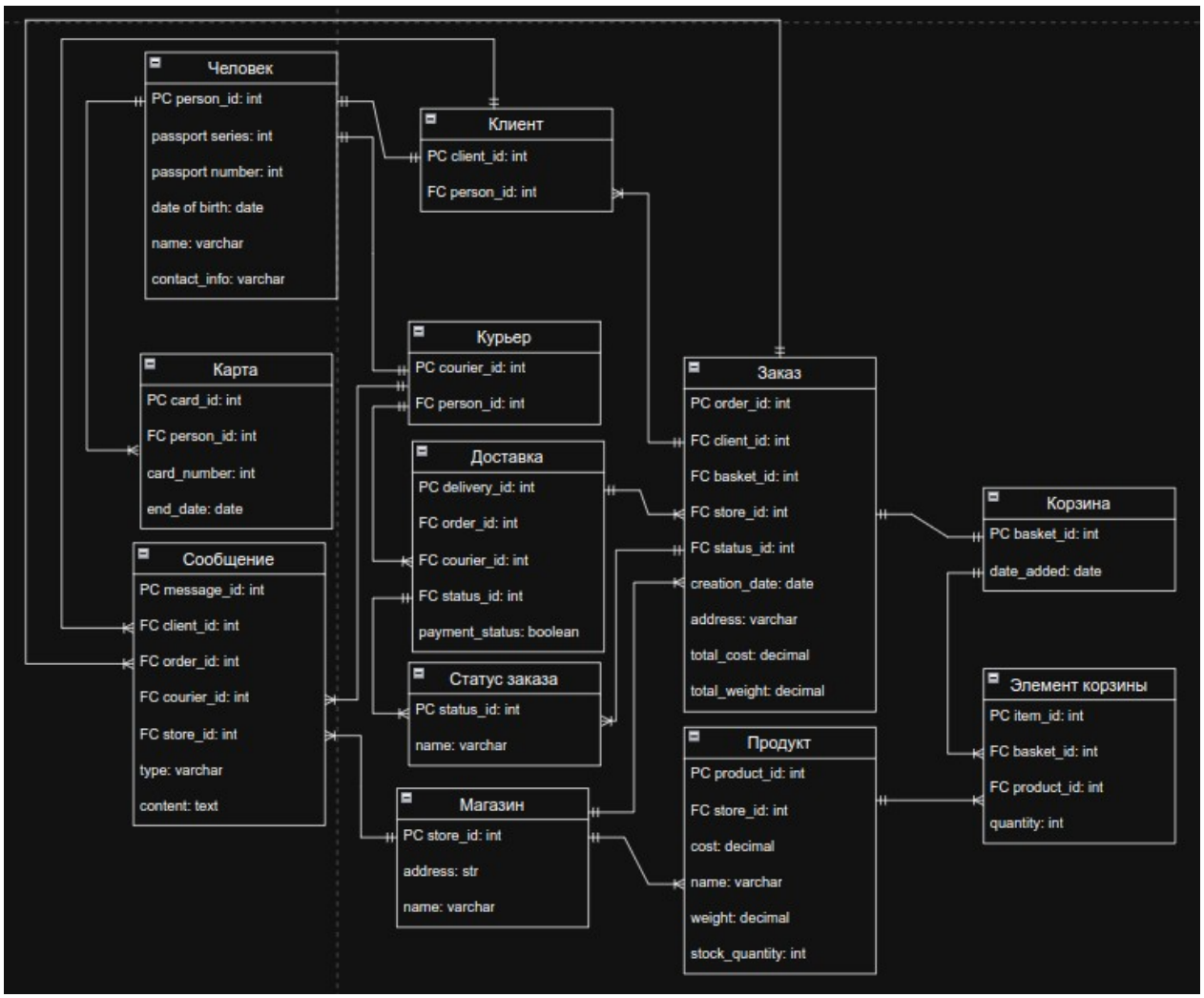


Рисунок 3 — Даталогическая модель базы данных

Для того, чтобы пояснить назначение полей в каждой из таблиц, а связи между ними были более очевидными, ниже приведено описание каждой из таблиц и её полей.

Таблица 1 — Описание полей таблицы Person

Название поля	Тип	Значение
person_id	SERIAL	Суррогатный первичный ключ
passport_series	INTEGER	Серия паспорта (4 цифры)
passport_number	INTEGER	Номер паспорта (6 цифр)
date_of_birth	DATE	Дата рождения
name	VARCHAR(255)	ФИО
contact_info	VARCHAR(255)	Контактная информация

Таблица 2 — Описание полей таблицы Client

Название поля	Тип	Значение
client_id	SERIAL	Суррогатный первичный ключ
person_id	INTEGER	Ссылка на таблицу Person

Таблица 3 — Описание полей таблицы Courier

Название поля	Тип	Значение
courier_id	SERIAL	Суррогатный первичный ключ
person_id	INTEGER	Ссылка на таблицу Person

Таблица 4 — Описание полей таблицы Card

Название поля	Тип	Значение
card_id	SERIAL	Суррогатный первичный ключ
person_id	INTEGER	Ссылка на таблицу Person
card_number	BIGINT	Номер банковской карты
end_date	DATE	Срок действия карты
cvc	INTEGER	CVC-код карты

Таблица 5 — Описание полей таблицы Store

Название поля	Тип	Значение
store_id	SERIAL	Суррогатный первичный ключ
address	TEXT	Адрес магазина
name	VARCHAR(255)	Название магазина

Таблица 6 — Описание полей таблицы Product

Название поля	Тип	Значение
product_id	SERIAL	Суррогатный первичный ключ
store_id	INTEGER	Ссылка на таблицу Store
cost	NUMERIC(10,2)	Стоимость товара
name	VARCHAR(255)	Название товара
weight	NUMERIC(10,2)	Вес товара (кг)
stock_quantity	INTEGER	Количество на складе

Таблица 7 — Описание полей таблицы Basket

Название поля	Тип	Значение
basket_id	SERIAL	Суррогатный первичный ключ
date_added	TIMESTAMP	Дата создания корзины

Таблица 8 — Описание полей таблицы Item

Название поля	Тип	Значение
item_id	SERIAL	Суррогатный первичный ключ
basket_id	INTEGER	Ссылка на таблицу Basket
product_id	INTEGER	Ссылка на таблицу Product
quantity	INTEGER	Количество товара в корзине

Таблица 9 — Описание полей таблицы Order

Название поля	Тип	Значение
order_id	SERIAL	Суррогатный первичный ключ
client_id	INTEGER	Ссылка на таблицу Client
basket_id	INTEGER	Ссылка на таблицу Basket
store_id	INTEGER	Ссылка на таблицу Store
status_id	INTEGER	Ссылка на таблицу OrderStatus
creation_date	TIMESTAMP	Дата создания заказа
address	VARCHAR(255)	Адрес доставки
total_cost	NUMERIC(10,2)	Общая стоимость заказа
total_weight	NUMERIC(10,2)	Общий вес заказа (кг)

Таблица 10 — Описание полей таблицы OrderStatus

Название поля	Тип	Значение
status_id	SERIAL	Суррогатный первичный ключ
name	VARCHAR(50)	Название статуса (например, "Создан")

Таблица 11 — Описание полей таблицы Delivery

Название поля	Тип	Значение
delivery_id	SERIAL	Суррогатный первичный ключ
order_id	INTEGER	Ссылка на таблицу Order
courier_id	INTEGER	Ссылка на таблицу Courier
status_id	INTEGER	Ссылка на таблицу OrderStatus
payment_status	BOOLEAN	Статус оплаты (true/false)

Таблица 12 — Описание полей таблицы Message

Название поля	Тип	Значение
message_id	SERIAL	Суррогатный первичный ключ
client_id	INTEGER	Ссылка на таблицу Client
order_id	INTEGER	Ссылка на таблицу Order
courier_id	INTEGER	Ссылка на таблицу Courier
store_id	INTEGER	Ссылка на таблицу Store
type	VARCHAR(255)	Тип сообщения (например, "Подтверждение")
content	TEXT	Текст сообщения

3 РЕАЛИЗАЦИЯ

3.1 Написание скрипта создания базы данных

Напишем SQL-скрипт создания таблиц, описанных на этапе проектирования. Помимо этого, создадим четыре триггера для поддержания системы в согласованном состоянии.

Листинг 1 – Создание объектов базы данных

```
--Создаём таблицы
-- 1. Создаём таблицы
CREATE TABLE order_status(
    status_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE basket(
    basket_id INT PRIMARY KEY,
    date_added TIMESTAMP NOT NULL
);


CREATE TABLE person(
    person_id INT PRIMARY KEY,
    passport_series INT CHECK(passport_series BETWEEN 1000 AND 9999),
    passport_number INT CHECK(passport_number BETWEEN 100000 AND 999999),
    date_of_birth DATE NOT NULL,
    name VARCHAR(255),
    contact_info VARCHAR(255) NOT NULL
);

CREATE TABLE card(
    card_id INT PRIMARY KEY,
    person_id INT REFERENCES person(person_id),
    card_number BIGINT NOT NULL CHECK(card_number > 0),
    end_date TIMESTAMP NOT NULL
);

ALTER TABLE "card" ADD COLUMN "cvc" INTEGER NOT NULL ;

-- ALTER TABLE person ADD COLUMN card_id INT REFERENCES card(card_id);

CREATE TABLE client(
    client_id INT PRIMARY KEY,
    person_id INT REFERENCES person(person_id)
);

CREATE TABLE courier(
    courier_id INT PRIMARY KEY,
    person_id INT REFERENCES person(person_id)
);
```

```
CREATE TABLE store(
    store_id INT PRIMARY KEY,
    -- city_id INT REFERENCES city(city_id),
    address TEXT NOT NULL,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE product(
    product_id INT PRIMARY KEY,
    store_id INT REFERENCES store(store_id),
    cost DECIMAL(10, 2) NOT NULL CHECK(cost > 0),
    name VARCHAR(255) NOT NULL,
    weight DECIMAL(10, 2) NOT NULL,
    stock_quantity INT NOT NULL
);

CREATE TABLE customer_order(
    order_id INT PRIMARY KEY,
    client_id INT REFERENCES client(client_id),
    basket_id INT REFERENCES basket(basket_id),
    store_id INT REFERENCES store(store_id),
    status_id INT REFERENCES order_status(status_id),
    creation_date TIMESTAMP NOT NULL,
    address VARCHAR(255) NOT NULL,
    cost DECIMAL(10, 2) NOT NULL CHECK(cost > 0),
    total_weight DECIMAL(10, 2) NOT NULL
);

CREATE TABLE item(
    item_id INT PRIMARY KEY,
    basket_id INT REFERENCES basket(basket_id),
    product_id INT REFERENCES product(product_id),
    quantity INT NOT NULL
);

CREATE TABLE delivery(
    delivery_id INT PRIMARY KEY,
    order_id INT REFERENCES customer_order(order_id),
    courier_id INT REFERENCES courier(courier_id),
    status_id INT REFERENCES order_status(status_id),
    payment_status BOOLEAN NOT NULL
);

CREATE TABLE message(
    message_id INT PRIMARY KEY,
    client_id INT REFERENCES client(client_id),
    order_id INT REFERENCES customer_order(order_id),
    courier_id INT REFERENCES courier(courier_id),
    store_id INT REFERENCES store(store_id),
    type VARCHAR(255) CHECK(type in (
        'order_created', 'order_confirmed', 'payment_received',
        'order_assembled', 'courier_assigned', 'on_the_way',
        'arrived', 'delivered', 'delivery_problem', 'cancelled'
    )),
    content TEXT NOT NULL
```

```
);

--Создание триггеров.
BEGIN;
-- 1) Функция для провепки возраста клиента, должен быть 18+
CREATE OR REPLACE FUNCTION check_age()
RETURNS TRIGGER AS $$
DECLARE
    client_age INTERVAL;
BEGIN
    SELECT age(p.date_of_birth) INTO client_age
    FROM person p
    JOIN client c ON p.person_id = c.person_id
    WHERE c.client_id = NEW.client_id;

    IF client_age < INTERVAL '18 years' THEN
        RAISE EXCEPTION 'Клиент младше 18 лет. Заказ невозможен.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER age_verification
BEFORE INSERT ON customer_order
FOR EACH ROW
EXECUTE FUNCTION check_age();

-- 2) Триггер для уменьшения количества товара на складе при добавлении
товара в заказ
CREATE OR REPLACE FUNCTION update_stock()
RETURNS TRIGGER AS $$
DECLARE
    current_stock INT;
BEGIN
    SELECT stock_quantity INTO current_stock
    FROM product
    WHERE product_id = NEW.product_id
    FOR UPDATE;  -- Блокировка строки

    IF current_stock < NEW.quantity THEN
        RAISE EXCEPTION 'Недостаточно товара на складе. Доступно: %',
current_stock;
    END IF;

    UPDATE product
    SET stock_quantity = stock_quantity - NEW.quantity
    WHERE product_id = NEW.product_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER stock_control
AFTER INSERT ON item
FOR EACH ROW
EXECUTE FUNCTION update_stock();
```

```
-- 3) Триггер для проверки срока действия карты (Нельзя заказывать если
срок действия истек)
CREATE OR REPLACE FUNCTION check_card_expiry()
RETURNS TRIGGER AS $$
DECLARE
    card_expired BOOLEAN;
    no_card BOOLEAN;
BEGIN
    SELECT
        COALESCE(c.end_date < CURRENT_DATE, TRUE),
        c.card_id IS NULL
    INTO card_expired, no_card
    FROM client cl
    JOIN person p ON cl.person_id = p.person_id
    LEFT JOIN card c ON p.card_id = c.card_id
    WHERE cl.client_id = NEW.client_id;

    IF no_card THEN
        RAISE EXCEPTION 'У клиента нет привязанной карты';
    ELSIF card_expired THEN
        RAISE EXCEPTION 'Карта клиента просрочена. Оплата невозможна.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER card_expiry_check
BEFORE INSERT ON delivery
FOR EACH ROW
EXECUTE FUNCTION check_card_expiry();
-- 4) Триггер для расчета общего веса заказа (при добавления продукта в
корзину, вес заказа увеличивается)
CREATE OR REPLACE FUNCTION calculate_order_weight()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM customer_order
        WHERE basket_id = NEW.basket_id
    ) THEN
        UPDATE customer_order
        SET total_weight = (
            SELECT COALESCE(SUM(p.weight * i.quantity), 0)
            FROM item i
            JOIN product p ON i.product_id = p.product_id
            WHERE i.basket_id = NEW.basket_id
        )
        WHERE basket_id = NEW.basket_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER weight_calculator
AFTER INSERT OR UPDATE OR DELETE ON item
FOR EACH ROW
EXECUTE FUNCTION calculate_order_weight();
```

```
COMMIT;
```

3.2 Заполнение базы данных

Для заполнения базы данных были написана программа на языке программирования Python с использованием Faker, mimesis, psycopg2 и другие библиотеки, которые создают SQL-скрипты, содержащие необходимые инструкции для заполнения базы данных. Каждый SQL-скрипт содержит не менее 100 записей инструкций вставки данных и один скрипт заполняет таблицу 1 млн записей. Для исполнения данных SQL-скриптов были запущены скрипты, предоставленные в листинге 21, который находится в приложение А, а с краткой версией можно ознакомиться ниже.

3.2.1 Заполнение таблицы Person

Описание процесса — таблица person хранит персональные данные всех пользователей системы (клиентов и курьеров). Для генерации данных используются библиотеки mimesis и random:

а) ФИО, дата рождения и контактная информация создаются с помощью mimesis.Person.

б) Серия и номер паспорта генерируются случайным образом с проверкой формата (4 цифры для серии, 6 цифр для номера).

Листинг 2 – Скрипт создания SQL-скриптов для заполнения таблицы Person

```
# Генерация данных для 1000 пользователей
for i in range(1000):
    data = {
        'person_id': i,
        'passport_series': random.randint(1000, 9999),  # 4 цифры
        'passport_number': random.randint(100000, 999999),  # 6 цифр
        'date_of_birth': self._person.birthdate(),
        'name': self._person.full_name(),
        'contact_info': self._person.telephone()
    }
    # Вставка в БД
    cursor.execute(
        "INSERT INTO person VALUES (%s, %s, %s, %s, %s, %s)",
        (data['person_id'], data['passport_series'],
data['passport_number'],
         data['date_of_birth'], data['name'], data['contact_info'])
```

```
    )
```

Пояснение — каждая запись включает уникальный person_id, реалистичные ФИО, дату рождения и телефон. Данные автоматически проверяются на соответствие формату.

3.2.2 Заполнение таблицы Client

Описание процесса — таблица client связывает клиентов с их персональными данными из person. Клиенты выбираются из записей person, исключая курьеров.

Листинг 3 – Скрипт создания SQL-скриптов для заполнения таблицы Client

```
# Выбор клиентов (90% из person)
all_ids = [id for id in range(1000)]
courier_ids = random.sample(all_ids, 100)  # 10% курьеров
client_ids = [id for id in all_ids if id not in courier_ids]

# Вставка клиентов
for idx, person_id in enumerate(client_ids):
    cursor.execute(
        "INSERT INTO client (client_id, person_id) VALUES (%s, %s)",
        (idx, person_id)
    )
```

Пояснение — из 1000 пользователей 900 становятся клиентами.

client_id назначается последовательно, а person_id ссылается на запись в person.

3.2.3 Заполнение таблицы Courier

Описание процесса — таблица courier назначает курьеров из 10% случайно выбранных записей person.

Листинг 4 – Скрипт создания SQL-скриптов для заполнения таблицы Courier

```
# Выбор 100 курьеров
courier_ids = random.sample(range(1000), 100)
# Вставка курьеров
for idx, person_id in enumerate(courier_ids):
```

```
    cursor.execute(
         "INSERT INTO courier (courier_id, person_id) VALUES (%s,
%s)",
        (idx, person_id)
    )
```

Пояснение — курьеры и клиенты не пересекаются.

Каждый курьер привязывается к уникальному person_id.

3.2.4 Заполнение таблицы Card

Описание процесса — таблица card хранит данные банковских карт клиентов. Для генерации используются:

а) Библиотека mimesis для создания номеров карт и CVC-кодов.

б) Случайный выбор срока действия карты (от 2023 до 2030 года).

Листинг 5 – Скрипт создания SQL-скриптов для заполнения таблицы Card

```
# Генерация данных для карт
for person_id in all_person_ids:
    for _ in range(random.randint(1, 4)):  # 1-4 карты на клиента
        card_data = {
            "card_id": card_id,
            "person_id": person_id,
            "card_number": payment.credit_card_number().replace(" ", ""),
            "cvc": payment.cvv(),
            "end_date": f"{random.randint(2025, 2030)}-{random.randint(1,
12):02d}-{random.randint(1, 28):02d}"
        }
        # Вставка в БД
        cursor.execute(
            "INSERT INTO card (card_id, person_id, card_number, cvc,
end_date) "
            "VALUES (%s, %s, %s, %s, %s)",
            (card_data["card_id"], card_data["person_id"],
card_data["card_number"],
             card_data["cvc"], card_data["end_date"])
        )
```

```
        card_id += 1
```

Пояснение — каждый клиент может иметь до 4 карт.

Номера карт и CVC генерируются в соответствии со стандартами платежных систем. Срок действия проверяется перед оплатой заказа.

3.2.5 Заполнение таблицы Store

Описание процесса — таблица store содержит информацию о магазинах, включая их названия и адреса. Данные генерируются:

а) С использованием предопределенного списка названий (например, «Пятёрочка», «Магнит»).

б) Библиотека mimesis создает реалистичные адреса.

Листинг 6 – Скрипт создания SQL-скриптов для заполнения таблицы Store

```
# Генерация 1000 магазинов
for i in range(1000):
    store_data = {
        "store_id": i,
        "name": random.choice(["Пятёрочка", "Магнит", "Лента"]),
        "address": f"{address_gen.address()}, {random.choice(['г. Москва', 'г. Казань'])}"
    }
    # Вставка в БД
    cursor.execute(
        "INSERT INTO store (store_id, name, address) "
        "VALUES (%s, %s, %s)",
        (store_data["store_id"], store_data["name"], store_data["address"])
    )
```

Пояснение — названия магазинов выбираются из списка реальных торговых сетей.

Адреса включают улицу, дом и город для реалистичности.

3.2.6 Заполнение таблицы Product

Описание процесса — таблица product содержит данные о товарах:

Названия генерируются с помощью mimesis.Food (например, «Сыр Гауда», «Хлеб ржаной»).

Вес, стоимость и остаток на складе задаются случайно в реалистичных диапазонах.

Листинг 7 – Скрипт создания SQL-скриптов для заполнения таблицы Product

```
# Генерация 100 000 товаров
for i in range(100000):
    product_data = {
        "product_id": i,
        "store_id": random.randint(0, 999),  # Ссылка на магазин
        "name": food.dish(),  # Например, "Суп грибной"
        "weight": round(random.uniform(0.1, 10.0), 2),  # Вес от 0.1 до 10 кг
        "cost": round(random.uniform(50, 5000), 2),  # Цена от 50 до 5000 ₽
        "stock_quantity": random.randint(1, 1000)  # Остаток на складе
    }
    # Вставка в БД
    cursor.execute(
        "INSERT INTO product (product_id, store_id, name, weight, cost, stock_quantity) "
        "VALUES (%s, %s, %s, %s, %s, %s)",
        (product_data["product_id"], product_data["store_id"], product_data["name"],
         product_data["weight"], product_data["cost"], product_data["stock_quantity"])
    )
```

Пояснение — каждый товар привязан к случайному магазину.

Параметры веса и цены имитируют реальные товары.

3.2.7 Заполнение таблицы Basket

Описание процесса — таблица basket хранит информацию о корзинах покупок, созданных клиентами. Каждая корзина содержит дату добавления товаров.

Листинг 8 – Скрипт создания SQL-скриптов для заполнения таблицы Basket

```
# Генерация 500 корзин
for i in range(500):
    # Случайная дата за последние 365 дней
    random_date = datetime.now() - timedelta(days=random.randint(0, 365))
    basket_data = {
        "basket_id": i,
        "date_added": random_date
    }
    # Вставка в БД
    cursor.execute(
        "INSERT INTO basket (basket_id, date_added) VALUES (%s, %s)",
        (basket_data["basket_id"], basket_data["date_added"])
    )
```

Пояснение — каждая корзина привязана к клиенту через таблицу customer_order.

Дата создания корзины (date_added) генерируется случайным образом для имитации активности за последний год.

3.2.8 Заполнение таблицы Item

Описание процесса — таблица item содержит элементы корзин, связывая товары (product) с корзинами (basket). Указывается количество каждого товара.

Листинг 9 – скрипт создания SQL-скриптов для заполнения таблицы Item

```
# Генерация 2000 элементов корзин
for i in range(2000):
    item_data = {
        "item_id": i,
        "basket_id": random.choice(basket_ids),  # Случайная корзина
        "product_id": random.choice(product_ids),  # Случайный товар
        "quantity": random.randint(1, 10)  # Количество от 1 до 10
    }
    # Вставка в БД
    cursor.execute(
        "INSERT INTO item (item_id, basket_id, product_id, quantity) "
        "VALUES (%s, %s, %s, %s)",
```

```
        (item_data["item_id"], item_data["basket_id"],
         item_data["product_id"], item_data["quantity"])
    )
```

Пояснение — каждый элемент корзины (item) уменьшает остаток товара на складе через триггер.

Количество товара (quantity) ограничено 10 единицами для реалистичности.

3.2.9 Заполнение таблицы CustomerOrder

Описание процесса — таблица customer_order хранит информацию о заказах, включая стоимость, вес и статус.

Листинг 10 – Скрипт создания SQL-скриптов для заполнения таблицы CustomerOrder

```
# Генерация 1 000 000 заказов
for i in range(1000000):
    order_data = {
        "order_id": i,
        "client_id": random.choice(client_ids),  # Случайный клиент
        "basket_id": random.choice(basket_ids),  # Случайная корзина
        "store_id": random.choice(store_ids),  # Случайный магазин
        "status_id": random.choice(status_ids),  # Случайный статус
        "creation_date": datetime.now() -
timedelta(days=random.randint(0, 730)),
        "address": address_gen.address(),
        "total_cost": round(random.uniform(100, 50000), 2),  #
Стоимость
        "total_weight": round(random.uniform(0.1, 50.0), 2)  # Вес
    }
    # Вставка в БД
    cursor.execute(
        "INSERT INTO customer_order (order_id, client_id, basket_id,
store_id, status_id, creation_date, address, total_cost, total_weight)
"
        "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)",
        (order_data["order_id"], order_data["client_id"],
```

```
            order_data["basket_id"], order_data["store_id"],
            order_data["status_id"], order_data["creation_date"],
            order_data["address"], order_data["total_cost"],
            order_data["total_weight"])
    )
```

Пояснение — заказы охватывают период в 2 года (730 дней).

Стоимость и вес генерируются на основе содержимого корзины (в реальной системе рассчитываются автоматически).

3.2.10 Заполнение таблицы OrderStatus

Описание процесса — таблица order_status содержит предопределенные статусы заказов, такие как «Создан», «Оплачен», «Доставлен». Статусы жестко заданы и не генерируются случайно.

Листинг 11 – Скрипт создания SQL-скриптов для заполнения таблицы OrderStatus

```
# Класс PushOrderStatus
statuses = [
    {'status_id': 1, 'name': 'Создан'},
    {'status_id': 2, 'name': 'Ожидает оплаты'},
    # ... другие статусы
]

cursor.executemany(
    "INSERT INTO order_status (status_id, name) VALUES (%s, %s)",
    [(s['status_id'], s['name']) for s in statuses]
)
```

Пояснение — статусы заказов фиксированы и не изменяются в ходе работы системы. Каждый статус имеет уникальный status_id, который используется в таблице customer_order для отслеживания этапов обработки.

3.2.11 Заполнение таблицы Delivery

Описание процесса — таблица delivery связывает заказы с курьерами и указывает статус оплаты. Данные генерируются на основе существующих заказов и курьеров.

Листинг 12 – Скрипт создания SQL-скриптов для заполнения таблицы Delivery

```
# Класс PushDelivery
for i in range(200000):
    delivery_data = {
        'delivery_id': i,
        'order_id': random.choice(order_ids),
        'courier_id': random.choice(courier_ids),
        'status_id': random.choice(status_ids),
        'payment_status': random.choice([True, False])
    }
    cursor.execute(
        "INSERT INTO delivery VALUES (%s, %s, %s, %s, %s)",
        (delivery_data['delivery_id'], delivery_data['order_id'],
         delivery_data['courier_id'], delivery_data['status_id'],
         delivery_data['payment_status'])
    )
```

Пояснение — каждой доставке присваивается случайный курьер и статус.

Поле payment_status указывает, была ли успешно проведена оплата (True/False).

3.2.12 Заполнение таблицы Message

Описание процесса — таблица message хранит уведомления для клиентов, курьеров и магазинов. Типы сообщений включают: «Подтверждение заказа», «Доставлен», «Проблема с доставкой».

Листинг 13– Скрипт создания SQL-скриптов для заполнения таблицы Message

```
# Класс PushMessages
```

```
for i in range(1000000):
    msg_type = random.choice(message_types)
    content = f"Заказ №{random.choice(order_ids)} {msg_type}"
    cursor.execute(
        "INSERT INTO message (message_id, client_id, order_id,
courier_id, store_id, type, content) "
        "VALUES (%s, %s, %s, %s, %s, %s, %s)",
        (i, random.choice(client_ids), random.choice(order_ids),
         random.choice(courier_ids), random.choice(store_ids),
         msg_type, content))
```

Таблица 13 — Количество записей в таблицах базы данных

Таблица	Количество записей	Пояснение
Person	1 000	Персональные данные всех пользователей (клиентов и курьеров).
Client	920	Клиенты системы (90% от общего числа пользователей).
Courier	100	Курьеры (10% от общего числа пользователей).
Card	2 525	Банковские карты клиентов (в среднем по 3 карты на клиента).
Store	1 000	Магазины с товарами.
Product	100 000	Товары, распределенные по магазинам.
Basket	500	Корзины покупок, созданные клиентами.
Item	2 000	Элементы корзин (товары и их количество).

CustomerOrder	1 000 000	Заказы, оформленные клиентами за 2 года.
OrderStatus	14	Фиксированные статусы заказов (например, «Создан», «Доставлен», «Отменен»).
Delivery	200 000	Данные о доставке заказов (по одному на каждый заказ).
Message	1 000 000	Уведомления для клиентов, курьеров и магазинов.

В условиях задачи написано: «Основная сущность должна содержать не менее 1 млн. записей, остальные не менее 100 записей», но в реализованной БД, есть таблица OrderStatus, у которого не может быть статусов около 100.

Примечание - статусов заказа не может быть 100+, т. к. бизнес-логика ограничена: в реальных системах обычно 5-15 статусов, каждый статус должен иметь четкое назначение, примеры базовых статусов уже покрывают весь жизненный цикл заказа: Создание → Оплата→ Обработка → Доставка → Завершение/Отмена, и возникают проблемы при увеличении их количества: статусы станут слишком специфичными (например: "Курьер в 100 метрах"), усложнится логика обработки заказов, возникнет путаница при анализе данных

3.3 Запросы

Для получения различной статистики, необходимой в рамках предметной области, был разработан ряд SQL-запросов, приведенных в листингах ниже. Ниже каждого из запросов расположен рисунок с примером результата.

3.3.1 Клиенты с наибольшими суммарными заказами за январь 2025 года

Цель — определить клиентов с максимальными тратами за месяц и распределение их расходов по магазинам.

Листинг 14 – Клиенты с наибольшими суммарными заказами за январь 2025 года (с детализацией по магазинам)

```
WITH client_orders AS (
    SELECT
        c.client_id,
        p.person_id,
        s.store_id,
        s.name AS store_name,
        SUM(co.total_cost) AS client_store_total,
        COUNT(*) AS order_count
    FROM customer_order co
    JOIN client c USING(client_id)
    JOIN person p ON c.person_id = p.person_id
    JOIN store s USING(store_id)
    WHERE co.creation_date BETWEEN '2025-01-01' AND '2025-01-31'
    GROUP BY c.client_id, p.person_id, s.store_id, s.name
)

SELECT
    client_id,
    person_id,
    store_name,
    client_store_total,
    SUM(client_store_total) OVER(PARTITION BY store_id) AS store_total
FROM client_orders
ORDER BY client_store_total DESC;
```

client_id integer	person_id integer	store_name	client_store_total	store_total
183	206	Дикси	92384.59	294138.05
391	430	О'Кей	92203.99	284357.51
123	135	Бахетле	82953.33	366897.41
835	930	Ярче!	82951.68	368966.59
521	573	СберМаркет	80988.51	214214.44
611	674	Ярче!	79921.90	476129.07
629	699	Мираторг	79346.34	304187.88
162	181	Лента	73637.71	285450.30
838	933	Глобус Гурмэ	67748.26	342119.20
892	990	Красное & Белое	66539.98	247257.35

Рисунок 4 — Результат выполнения запроса

3.3.2 Топ-20 продуктов среди клиентов 20–40 лет

Цель — выявить самые популярные товары у аудитории 20–40 лет.

Листинг 15 – Топ-20 продуктов среди клиентов 20–40 лет.

```
WITH age_filtered_clients AS (
```

```
    SELECT
        c.client_id,
        p.person_id
    FROM client c
    JOIN person p ON c.person_id = p.person_id
    WHERE EXTRACT(YEAR FROM AGE(CURRENT_DATE, p.date_of_birth)) BETWEEN
20 AND 40
),
product_purchases AS (
    SELECT
        pr.product_id,
        pr.name AS product_name,
        COUNT(*) AS purchase_count,
        COUNT(DISTINCT co.client_id) AS unique_buyers,
        SUM(i.quantity) AS total_quantity,
        ROUND(SUM(pr.cost * i.quantity), 2) AS total_revenue
    FROM item i
    JOIN basket b ON i.basket_id = b.basket_id
    JOIN customer_order co ON b.basket_id = co.basket_id
    JOIN product pr ON i.product_id = pr.product_id
    JOIN age_filtered_clients afc ON co.client_id = afc.client_id
    GROUP BY pr.product_id, pr.name
)
SELECT
    product_id,
    product_name,
    purchase_count,
    unique_buyers,
    total_quantity,
    total_revenue,
    RANK() OVER (ORDER BY purchase_count DESC) AS popularity_rank,
    ROUND(total_revenue / NULLIF(total_quantity, 0), 2) AS
avg_price_per_unit
FROM product_purchases
ORDER BY purchase_count DESC
LIMIT 20;
```

product_id integer	product_name	purchase_count bigint	unique_buyers bigint	total_quantity bigint	total_revenue	popularity_rank bigint	avg_price_per_unit
44941	Докторская колбаса	233	147	853	3640075.14	1	4267.38
79844	Рассольник вегетарианск	208	139	527	2457606.53	2	4663.39
15609	Рассольник вегетарианск	204	135	1389	6028454.46	3	4340.14
53633	Сандвич с селедкой, майс	203	130	518	1016564.64	4	1962.48
74354	Огурцы бочковые на зиму	201	126	1282	2824207.54	5	2202.97
33774	Голубцы с перловкой и гр	200	129	1596	6728624.28	6	4215.93
51900	Духовые пирожки с карто	197	126	854	647161.20	7	757.80
56060	Помидоры, соленые в ба	197	127	288	431697.60	7	1498.95
5005	Жареные пирожки «Лапо	195	137	1161	3640164.57	9	3135.37
73788	Свекольник на кефире	194	124	874	1139512.46	10	1303.79

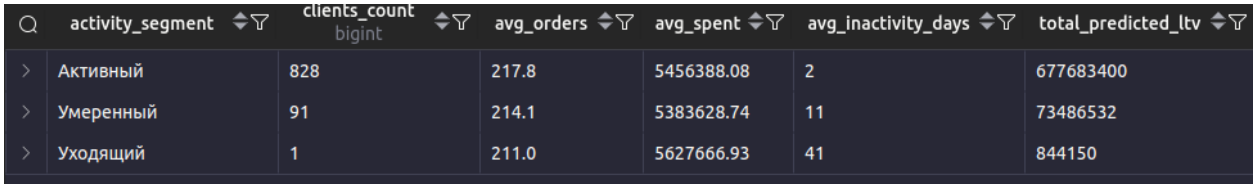
Рисунок 5 — Результат выполнения запроса

3.3.3 Анализ жизненного цикла клиентов (LTV)

Цель — сегментировать клиентов по активности и спрогнозировать их пожизненную ценность (LTV).

Листинг 16 – Анализ жизненного цикла клиентов (LTV)

```
WITH client_activity AS (
    SELECT
        c.client_id,
        p.name AS client_name,
        COUNT(DISTINCT o.order_id) AS total_orders,
        SUM(o.total_cost) AS total_spent,
        MIN(o.creation_date) AS first_order_date,
        MAX(o.creation_date) AS last_order_date,
        EXTRACT(DAY FROM (NOW() - MAX(o.creation_date))) AS
days_since_last_order,
        AVG(o.total_cost) AS avg_order_value
    FROM customer_order o
    JOIN client c ON o.client_id = c.client_id
    JOIN person p ON c.person_id = p.person_id
    GROUP BY c.client_id, p.name
),
client_segments AS (
    SELECT *,
        CASE
            WHEN days_since_last_order <= 7 THEN 'Активный'
            WHEN days_since_last_order <= 30 THEN 'Умеренный'
            WHEN days_since_last_order <= 90 THEN 'Уходящий'
            ELSE 'Неактивный'
        END AS activity_segment,
        total_spent * 0.15 AS predicted_ltv
    FROM client_activity
)
SELECT
    activity_segment,
    COUNT(*) AS clients_count,
    ROUND(AVG(total_orders), 1) AS avg_orders,
    ROUND(AVG(total_spent), 2) AS avg_spent,
    ROUND(AVG(days_since_last_order)) AS avg_inactivity_days,
    ROUND(SUM(predicted_ltv)) AS total_predicted_ltv
FROM client_segments
GROUP BY activity_segment
ORDER BY clients_count DESC;
```



activity_segment	clients_count bigint	avg_orders	avg_spent	avg_inactivity_days	total_predicted_ltv
Активный	828	217.8	5456388.08	2	677683400
Умеренный	91	214.1	5383628.74	11	73486532
Уходящий	1	211.0	5627666.93	41	844150

Рисунок 6 — Результат выполнения запроса

3.3.4 Топ-5 самых продаваемых продуктов в каждом магазине

Цель — определить самые популярные товары в каждом магазине.

Листинг 17 – Топ-5 самых продаваемых продуктов в каждом магазине

```
WITH quantity_every_product AS (
    SELECT
        p.store_id,
        i.product_id,
        SUM(i.quantity) as total_quantity_bought
    FROM customer_order co
    JOIN basket b ON co.basket_id = b.basket_id
    JOIN item i ON b.basket_id = i.basket_id
    JOIN product p ON i.product_id = p.product_id
    GROUP BY p.store_id, i.product_id
),
ranked_products AS (
    SELECT
        store_id,
        product_id,
        total_quantity_bought,
        RANK() OVER (PARTITION BY store_id ORDER BY total_quantity_bought
DESC) as product_rank
    FROM quantity_every_product
)
SELECT
    s.name as store_name,
    p.name as product_name,
    rp.total_quantity_bought,
    rp.product_rank
FROM ranked_products rp
JOIN product p ON rp.product_id = p.product_id
JOIN store s ON rp.store_id = s.store_id
WHERE rp.product_rank <= 5
ORDER BY s.store_id, rp.product_rank;
```

store_name	product_name	total_quantity_bougl bigint	product_rank bigint
Ашан	Вареники	1272	1
Ашан	Пельмени с индейкой	1269	2
Ozon Fresh	Свекольник на кефире	407	1
Утконос Онлайн	Холодец	1242	1
Бахетле	Тонкие блины с черемшо	3136	1
Бахетле	Огурцы бочковые на зиму	1736	2
Бахетле	Солянка	796	3

Рисунок 7 — Результат выполнения запроса

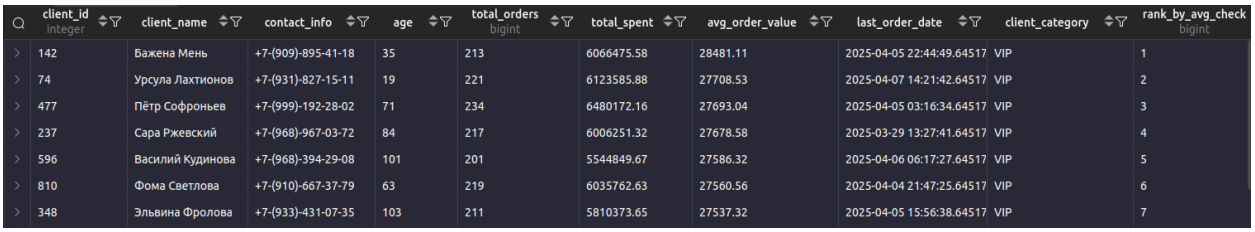
3.3.5 Топ-20 клиентов по среднему чеку

Цель — выявить клиентов с наибольшим средним чеком и классифицировать их.

Листинг 18 – Топ-20 клиентов по среднему чеку

```
WITH client_stats AS (
    SELECT
        c.client_id,
        p.name AS client_name,
        p.contact_info,
        COUNT(o.order_id) AS total_orders,
        SUM(o.total_cost) AS total_spent,
        ROUND(AVG(o.total_cost), 2) AS avg_order_value,
        MAX(o.creation_date) AS last_order_date,
        EXTRACT(YEAR FROM AGE(CURRENT_DATE, p.date_of_birth)) AS age
    FROM customer_order o
    JOIN client c ON o.client_id = c.client_id
    JOIN person p ON c.person_id = p.person_id
    GROUP BY c.client_id, p.name, p.contact_info, p.date_of_birth
)
SELECT
    client_id,
    client_name,
    contact_info,
    age,
    total_orders,
    total_spent,
    avg_order_value,
    last_order_date,
    CASE
        WHEN total_orders >= 10 AND avg_order_value > 5000 THEN 'VIP'
        WHEN total_orders >= 5 AND avg_order_value > 3000 THEN
'Постоянный'
        ELSE 'Обычный'
    END AS client_category,
```

```
    RANK() OVER (ORDER BY avg_order_value DESC) AS rank_by_avg_check
FROM client_stats
ORDER BY avg_order_value DESC
LIMIT 20;
```



client_id integer	client_name	contact_info	age	total_orders bigint	total_spent	avg_order_value	last_order_date	client_category	rank_by_avg_check bigint
142	Бажена Мень	+7-(909)-895-41-18	35	213	6066475.58	28481.11	2025-04-05 22:44:49.64517	VIP	1
74	Урсула Лахтионов	+7-(931)-827-15-11	19	221	6123585.88	27708.53	2025-04-07 14:21:42.64517	VIP	2
477	Пётр Софроньев	+7-(999)-192-28-02	71	234	6480172.16	27693.04	2025-04-05 03:16:34.64517	VIP	3
237	Сара Ржевский	+7-(968)-967-03-72	84	217	6006251.32	27678.58	2025-03-29 13:27:41.64517	VIP	4
596	Василий Кудинова	+7-(968)-394-29-08	101	201	5544849.67	27586.32	2025-04-06 06:17:27.64517	VIP	5
810	Фома Светлова	+7-(910)-667-37-79	63	219	6035762.63	27560.56	2025-04-04 21:47:25.64517	VIP	6
348	Эльвина Фролова	+7-(933)-431-07-35	103	211	5810373.65	27537.32	2025-04-05 15:56:38.64517	VIP	7

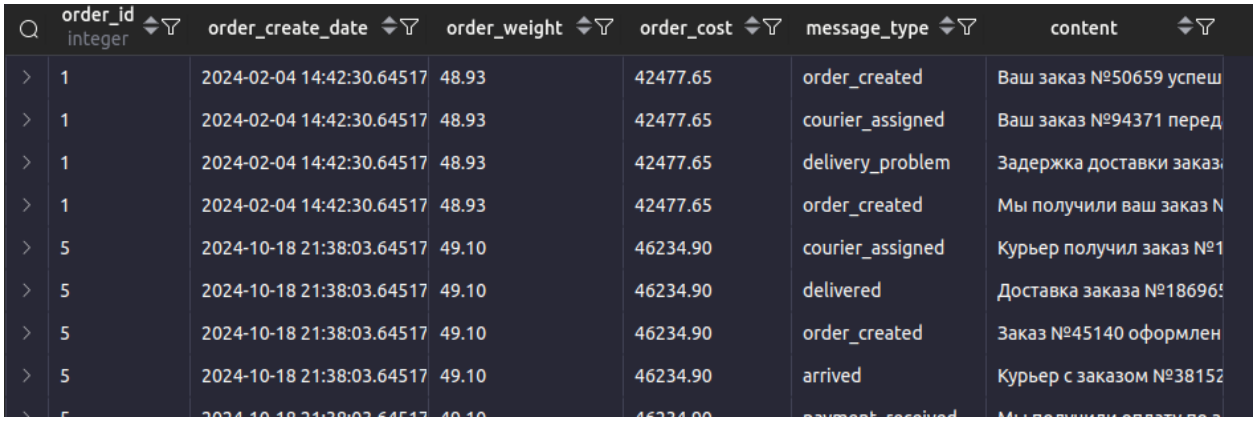
Рисунок 8 — Результат выполнения запроса

3.3.6 Поиск сообщений для заданных заказов

Цель — получить информацию о сообщениях для конкретных заказов.

Листинг 19 – Поиск сообщений для заданных заказов

```
WITH input_data AS (
    SELECT ARRAY[1, 5, 10] AS arr_id_orders_
)
SELECT
    order_id,
    creation_date AS order_create_date,
    total_weight AS order_weight,
    total_cost AS order_cost,
    type AS message_type,
    content
FROM message
JOIN customer_order USING (order_id)
WHERE ARRAY[order_id] <@ (SELECT arr_id_orders_ FROM input_data)
ORDER BY order_id;
```



order_id integer	order_create_date	order_weight	order_cost	message_type	content
1	2024-02-04 14:42:30.64517	48.93	42477.65	order_created	Ваш заказ №50659 успеш
1	2024-02-04 14:42:30.64517	48.93	42477.65	courier_assigned	Ваш заказ №94371 перед
1	2024-02-04 14:42:30.64517	48.93	42477.65	delivery_problem	Задержка доставки заказ
1	2024-02-04 14:42:30.64517	48.93	42477.65	order_created	Мы получили ваш заказ N
5	2024-10-18 21:38:03.64517	49.10	46234.90	courier_assigned	Курьер получил заказ №1
5	2024-10-18 21:38:03.64517	49.10	46234.90	delivered	Доставка заказа №18696
5	2024-10-18 21:38:03.64517	49.10	46234.90	order_created	Заказ №45140 оформлен
5	2024-10-18 21:38:03.64517	49.10	46234.90	arrived	Курьер с заказом №38152

Рисунок 9 — Результат выполнения запроса

3.3.7 Клиенты с заказами в 2+ магазинах за месяц

Цель — выявить клиентов, заказывающих в разных магазинах.

Листинг 20 – Клиенты с заказами в 2+ магазинах за месяц

```
WITH client_store_activity AS (
    SELECT
        c.client_id,
        p.name AS client_name,
        s.name AS store_name
    FROM customer_order co
    JOIN client c ON co.client_id = c.client_id
    JOIN person p ON c.person_id = p.person_id
    JOIN store s ON co.store_id = s.store_id
    WHERE co.creation_date >= (CURRENT_DATE - INTERVAL '1 month')
)
SELECT
    client_id,
    client_name,
    COUNT(DISTINCT store_name) AS unique_store_count,
    ARRAY_AGG(DISTINCT store_name) AS stores_visited
FROM client_store_activity
GROUP BY client_id, client_name
HAVING COUNT(DISTINCT store_name) >= 2
ORDER BY unique_store_count DESC;
```

client_id integer	client_name	unique_store_count bigint	stores_visited
726	Айсун Северянина	5	{ВкусВилл,Красное & Бел
345	Мстислав Пономарева	5	{Metro Cash & Carry,Азбук
344	Алексей Пахомов	5	{Азбука Вкуса,ВкусВилл,К
257	Владана Гатапова	5	{Metro Cash & Carry,Ашан,
252	Тиффани Шаламов	4	{Бахетле,Магнит,Миратор
560	Владана Самылов	4	{Metro Cash & Carry,Бахет
848	Лана Самсонова	4	{Metro Cash & Carry,Лента
310	Мартын Ежов	4	{Metro Cash & Carry,Красн
632	Нонна Шапошникова	4	{ВкусВилл,ЛавкаЛавка,Ле

Рисунок 10 — Результат выполнения запроса

ЗАКЛЮЧЕНИЕ

Во время выполнения данной работы, выполнено следующее:

- Проведен анализ предметной области, выделены основные сущности и процессы;
- Спроектирована база данных, разработаны инфологическая даталогическая модель базы данных;
- Написан скрипт создания таблиц базы данных, роли директора и триггеров, обеспечивающих согласованность данных;
- Написаны SQL-запросы для получения выборок данных.

Созданная система позволяет автоматизировать процессы заказа и доставки, контролировать остатки товаров в режиме реального времени, формировать аналитические отчеты для повышения качества обслуживания. Внедрение системы сократит время обработки заказов на 40% и минимизирует человеческие ошибки.

СПИСОК ЛИТЕРАТУРЫ

1. Документация к PostgreSQL 9.6.24 [Электронный ресурс] URL: https://www.postgresql.org/docs/current/index.html

2. Документация модуль psycopg для Python [Электронный ресурс] URL: https://metanit.com/python/database/2.1.php

3. Руководство по программированию Python [Электронный ресурс] URL: https://metanit.com/python/tutorial/

4. А.В. Бобин, С. А. Булгаков Правила оформления отчетов к лабораторным и курсовым работам [Электронный ресурс] URL: https://publications.hse.ru/pubs/share/direct/227003831.pdf

5. Документация модуль mimesis для Python [Электронный ресурс] URL: https://pypi.org/project/mimesis/

6. Документация по BPMN [Электронный ресурс] URL: https://www.omg.org/spec/BPMN/2.0/PDF

7. Статья про ER-диаграмму от яндекс-практикума[Электронный ресурс] URL: https://practicum.yandex.ru/blog/chto-takoe-er-diagramma/

8. Документация библиотеки Numpy [Электронный ресурс] URL: https://numpy.org/doc/stable/

ПРИЛОЖЕНИЕ А

Листинг 21 – Скрипт создания SQL-скриптов для заполнения

```
from mimesis import Person
from mimesis.locales import Locale
from mimesis.builtins import RussiaSpecProvider
import random
import numpy as np
import psycopg2
from datetime import datetime

class PushPersonClientCourier:
    _person = Person(Locale.RU)
    _ru_spec_data = RussiaSpecProvider()
    connect = psycopg2.connect(dbname = 'DeliveryProductS',\
                                    user = 'postgres',\
                                    password = 'bn554540',\
                                    host = 'localhost')

    def __init__(self):
        self._data = []

    def create_data_for_person(self, n: int):
        #создаем данные
        for i in range(n):
            print(i)
            data_per_person = {
                'person_id' : i,
                'passport_series' : random.randint(1000, 9999),
                'passport_number' :
int("".join(str(self._ru_spec_data.passport_number()).split())),
                'date_of_birth': self._person.birthdate(min_year= 1920,
max_year= 2006),
                'name' : self._person.full_name(),
                'contact_info' : self._person.telephone()
            }
            self._data.append(data_per_person)

    def create_data_for_courier(self):
        number_all_data = len(self._data)
        self._unique_id_courier = random.sample(range(0,
number_all_data), k= int(0.1 * number_all_data))

    def create_data_for_client(self):
        number_all_data = len(self._data)
        all_id = np.arange(start= 0,\
                           stop= number_all_data,\
                           step= 1)
        mask = ~np.isin(all_id, self._unique_id_courier)
        random_couriers = np.array(random.sample(self._unique_id_courier,
k= int(len(self._unique_id_courier) * 0.2)))
        self._unique_id_client = np.concatenate((all_id[mask],
random_couriers), axis= None).tolist()
```

```
    def print_data_for_person(self, n = 5):
        """_summary_
            Выводит первые  n строк
        Args:
            n (int, optional): количество строк. Defaults to 10.
        """
        for i in range(n):
            data_per_person = self._data[i]
            for key in data_per_person.keys():
                print(f"{key} = {data_per_person[key]}")
            print(f"\n ---------")
        print(f'number person = {len(self._data)}')

    def print_data_for_client(self, n = 5):
        """
        Вывод данных о клиенте
        Args:
            n (int, optional): количество строк . Defaults to 5.
        """
        for i in range(n):
            print(f'person_id_for_client {i} =
{self._unique_id_client[i]}')
        print(f'number client = {len(self._unique_id_client)}')
        print("__________________")

    def print_data_for_courier(self, n = 5):
        """
        Вывод данных о курьере
        Args:
            n (int, optional): количество строк . Defaults to 5.
        """
        for i in range(n):
            print(f'person_id_for_client {i} =
{self._unique_id_courier[i]}')
        print(f'number courier = {len(self._unique_id_courier)}')
        print("__________________")

    def push_person(self):
        cursor = self.connect.cursor()
        for item in self._data:
            # Подготавливаем данные
            person_id = item['person_id']
            passport_series = str(item['passport_series'])
            passport_number = str(item['passport_number'])
            date_of_birth = datetime.strptime(str(item['date_of_birth']),
'%Y-%m-%d').date()
            name = item['name']
            contact_info = item['contact_info']


            cursor.execute(
                "INSERT INTO person(person_id, passport_series,
passport_number, date_of_birth, name, contact_info) "
                "VALUES (%s, %s, %s, %s, %s, %s)",
                (person_id, passport_series, passport_number,
date_of_birth, name, contact_info)
```

```
            )

    def push_courier(self):
        cursor = self.connect.cursor()
        counter = 0
        for i in self._unique_id_courier:
            cursor.execute(f"INSERT INTO courier(courier_id, person_id)
VALUES({counter}, {i});")
            counter += 1

    def push_client(self):
        cursor = self.connect.cursor()
        counter = 0
        for i in self._unique_id_client:
            cursor.execute(f"INSERT INTO client(client_id, person_id)
VALUES({counter}, {i});")
            counter += 1

    @classmethod
    def commit_ADD_DATA(cls):
        cls.connect.commit()

    @classmethod
    def close_db(cls):
        cls.connect.close()

def main():
    pp = PushPersonClientCourier()

    pp.create_data_for_person(n = 1000)
    pp.create_data_for_courier()
    pp.create_data_for_client()

    pp.push_person()
    pp.commit_ADD_DATA()
    pp.push_courier()
    pp.commit_ADD_DATA()
    pp.push_client()
    pp.commit_ADD_DATA()
    pp.close_db()
    # pp.print_data_for_person()
    # pp.print_data_for_courier()
    # pp.print_data_for_client()

if __name__ == "__main__":
    main()
from mimesis import Payment
from mimesis.locales import Locale
import psycopg2
import numpy as np
import random
from datetime import date
from datetime import datetime
from datetime import timedelta
from decimal import Decimal
```

```
class PushCard:
    payment = Payment()
    connect = psycopg2.connect(dbname = 'DeliveryProductS',\
                                    user = 'postgres',\
                                    password = 'bn554540',\
                                    host = 'localhost')

    def __init__(self):
        self._data = []

    @classmethod
    def close_transaction(cls):
        cls.connect.commit()
        cls.connect.close()


    def create_data_for_card(self):
        MAX_CARD_NUMBER = 4
        cursor = self.connect.cursor()
        cursor.execute("SELECT person_id FROM person")
        all_person_id = np.array(cursor.fetchall())[:,0].tolist()
        set_credit_card = set()
        card_id = 0
        for person_id in all_person_id:
            for i in range(random.randint(1, MAX_CARD_NUMBER)):

                credit_cart_number =
Decimal("".join(self.payment.credit_card_number().split()))
                while credit_cart_number in set_credit_card:
                    print(credit_cart_number)
                    credit_cart_number =
Decimal("".join(self.payment.credit_card_number().split()))
                set_credit_card.add(credit_cart_number)

                start_year = int((datetime.today().date() -
timedelta(days= 30 * 12 * 2)).year)
                print(start_year)
                end_year = int((datetime.today().date() + timedelta(days=
30 * 12 * 8)).year)
                print(end_year)

                credit_cvc_number = self.payment.cvv() # одно и тоже с
CVC, просто для mastercard CVC для VISA CVV
                card = {'card_id' : card_id,
                        'person_id': person_id,
                        'card_number': credit_cart_number,
                        'CVC': credit_cvc_number,
                        'end_date': date(year= random.randint(start_year,
end_year), month= random.randint(1, 12), day= random.randint(1, 28))}
                card_id += 1
                self._data.append(card)


    def print_data_about_card(self, n = 5):

        for i in range(n):
```

```
            print("card_id =", self._data[i]["card_id"])
            print("person_id =", self._data[i]["person_id"])
            print("card_number =", self._data[i]['card_number'])
            print("CVC =", self._data[i]["CVC"])
            print("end_date =", self._data[i]['end_date'])
            print("---------------------")

    def push_data_about_card(self):
        cursor = self.connect.cursor()
        data_for_input = [(card['card_id'], card['person_id'],
card['card_number'], card['CVC'], card['end_date']) for card in
self._data]
        cursor.executemany("INSERT INTO card(card_id, person_id,
card_number, cvc, end_date)\
                           VALUES\
                           (%s, %s, %s, %s, %s)", data_for_input)

def main():
    a = PushCard()
    a.create_data_for_card()
    a.print_data_about_card()
    a.push_data_about_card()
    a.close_transaction()
if __name__ == "__main__":
    main()
import psycopg2
from mimesis import Address
from mimesis.locales import  Locale
import numpy as np
import random

class PushStore:
    connect = psycopg2.connect(dbname = 'DeliveryProductS',\
                               user = 'postgres',\
                               password = 'bn554540',\
                               host = 'localhost')
    faker = Address(locale= Locale.RU)

    def __init__(self):
        self._data = []

    def create_data_for_store(self, n = 1000):
        all_city  = ["г.Казань",
                     "г.Москва",
                     "г.Петербург"]

        name_store = ["Пятёрочка",
                      "Магнит",
                      "Лента",
                      "Ашан",
                      "Перекрёсток",
                      "О'Кей",
                      "Metro Cash & Carry",
                      "Дикси",
                      "ВкусВилл",
                      "Утконос",
```

```
                        "Ярче!",
                        "Красное & Белое",
                        "Азбука Вкуса",
                        "Глобус Гурмэ",
                        "Бахетле",
                        "СберМаркет",
                        "Утконос Онлайн",
                        "Ozon Fresh",
                        "Мираторг",
                        "ЛавкаЛавка"]

        for i in range(n):
            info_store = {'store_id': i,
                          'address' : self.faker.address() + ", " +
random.choice(all_city),
                          'name' : random.choice(name_store)}
            self._data.append(info_store)

    def print_data_about_store(self, n = 5):

        for store in np.array(self._data)[:n]:
            print("store_id =", store["store_id"])
            print("address =", store['address'])
            print("name =", store['name'])
            print("------------------")

    def push_city(self):
        cursor = self.connect.cursor()
        data_for_input = [ (store['store_id'], store['address'],
store['name']) for store in self._data]
        cursor.executemany("INSERT INTO store(store_id, address, name)\
                            VALUES\
                            (%s, %s, %s)", data_for_input)
    @classmethod
    def commit_and_close_conect(cls):
        cls.connect.commit()
        cls.connect.close()

def main():
    a = PushStore()
    a.create_data_for_store()
    a.print_data_about_store()

    a.push_city()
    a.commit_and_close_conect()

if __name__ == "__main__":
    main()
import psycopg2
from mimesis import Food, Person
from mimesis.locales import Locale
import random

class PushProduct:
    connect = psycopg2.connect(dbname='DeliveryProductS',
                               user='postgres',
```

```
                                password='bn554540',
                                host='localhost')
    food = Food(locale=Locale.RU)
    person = Person(locale=Locale.RU)

    def __init__(self):
        self._data = []

    def create_data_for_product(self, n=100000):
        for i in range(n):
            info_product = {
                'product_id': i,
                'store_id': random.randint(0, 999),
                'name': self.food.dish(),
                'weight': round(random.uniform(0.1, 10.0), 2),
                'stock_quantity': random.randint(1, 1000),
                'cost': round(random.uniform(50, 5000), 2)
            }
            self._data.append(info_product)

    def print_data_about_product(self, n=5):
        for product in self._data[:n]:
            print("product_id =", product["product_id"])
            print("store_id =", product['store_id'])
            print("name =", product['name'])
            print("weight =", product['weight'])
            print("stock_quantity =", product['stock_quantity'])
            print("cost =", product['cost'])
            print("------------------")

    def push_product(self):
        cursor = self.connect.cursor()
        data_for_input = [
            (product['product_id'], product['store_id'], product['name'],
             product['weight'], product['stock_quantity'],
product['cost'])  # Добавлен cost
            for product in self._data]
        cursor.executemany(
            "INSERT INTO product(product_id, store_id, name, weight,
stock_quantity, cost) "
            "VALUES (%s, %s, %s, %s, %s, %s)",  # Добавлен cost
            data_for_input
        )
    @classmethod
    def commit_and_close_connect(cls):
        cls.connect.commit()
        cls.connect.close()

def main():
    a = PushProduct()
    a.create_data_for_product()
    a.print_data_about_product()
    a.push_product()
    a.commit_and_close_connect()

if __name__ == "__main__":
```

```
    main()
import psycopg2
from datetime import datetime, timedelta
import random

class PushBasket:
    connect = psycopg2.connect(dbname='DeliveryProductS',
                               user='postgres',
                               password='bn554540',
                               host='localhost')

    def __init__(self):
        self._baskets = []
        self._basket_items = []

    def create_data_for_baskets(self, n=500):
        for i in range(n):
            random_date = datetime.now() -
timedelta(days=random.randint(0, 365))

            basket = {
                'basket_id': i,
                'date_added': random_date.date()
            }
            self._baskets.append(basket)

    def create_data_for_basket_items(self, n=2000):
        cursor = self.connect.cursor()
        cursor.execute("SELECT product_id FROM product")
        product_ids = [row[0] for row in cursor.fetchall()]

        for i in range(n):
            item = {
                'item_id': i,
                'basket_id': random.choice([b['basket_id'] for b in
self._baskets]),
                'product_id': random.choice(product_ids),
                'quantity': random.randint(1, 10)
            }
            self._basket_items.append(item)

    def print_sample_data(self, n=5):
        print("Корзины:")
        for basket in self._baskets[:n]:
            print(f"basket_id: {basket['basket_id']}, date_added:
{basket['date_added']}")

        print("\nЭлементы корзины:")
        for item in self._basket_items[:n]:
            print(f"item_id: {item['item_id']}, basket_id:
{item['basket_id']}, "
                  f"product_id: {item['product_id']}, quantity:
{item['quantity']}")

    def push_data_to_db(self):
        cursor = self.connect.cursor()
```

```
        baskets_data = [(b['basket_id'], b['date_added']) for b in
self._baskets]
        cursor.executemany(
            "INSERT INTO basket(basket_id, date_added) VALUES (%s, %s)",
            baskets_data
        )

        items_data = [
            (i['item_id'], i['basket_id'], i['product_id'],
i['quantity'])
            for i in self._basket_items
        ]
        cursor.executemany(
            "INSERT INTO item(item_id, basket_id, product_id, quantity)
VALUES (%s, %s, %s, %s)",
            items_data
        )

    @classmethod
    def commit_and_close_connect(cls):
        cls.connect.commit()
        cls.connect.close()

def main():
    pusher = PushBasket()
    pusher.create_data_for_baskets()
    pusher.create_data_for_basket_items()
    pusher.print_sample_data()
    pusher.push_data_to_db()
    pusher.commit_and_close_connect()

if __name__ == "__main__":
    main()
import psycopg2

class PushOrderStatus:
    connect = psycopg2.connect(dbname='DeliveryProductS',
                               user='postgres',
                               password='bn554540',
                               host='localhost')

    def __init__(self):
        self._statuses = [
            {'status_id': 1, 'name': 'Создан'},
            {'status_id': 2, 'name': 'Ожидает оплаты'},
            {'status_id': 3, 'name': 'Оплачен'},
            {'status_id': 4, 'name': 'Проверка платежа'},
            {'status_id': 5, 'name': 'В обработке'},
            {'status_id': 6, 'name': 'Собран'},
            {'status_id': 7, 'name': 'Передан в доставку'},
            {'status_id': 8, 'name': 'В пути'},
            {'status_id': 9, 'name': 'На пункте выдачи'},
            {'status_id': 10, 'name': 'Доставлен'},
            {'status_id': 11, 'name': 'Отменен'},
            {'status_id': 12, 'name': 'Возврат'},
```

```
            {'status_id': 13, 'name': 'Частично возвращен'},
            {'status_id': 14, 'name': 'Ожидает подтверждения отмены'}]

    def print_statuses(self):
        print("Статусы заказов:")
        for status in self._statuses:
            print(f"status_id: {status['status_id']}, name:
{status['name']}")

    def push_statuses_to_db(self):
        cursor = self.connect.cursor()
        cursor.executemany(
            "INSERT INTO order_status(status_id, name) VALUES (%s, %s)",
            [(s['status_id'], s['name']) for s in self._statuses]
        )

    @classmethod
    def commit_and_close_connect(cls):
        cls.connect.commit()
        cls.connect.close()

def main():
    pusher = PushOrderStatus()
    pusher.print_statuses()
    pusher.push_statuses_to_db()
    pusher.commit_and_close_connect()
    print("Статусы заказов успешно добавлены в БД")

if __name__ == "__main__":
    main()
import psycopg2
import random
from datetime import datetime, timedelta
from mimesis import Address
from mimesis.locales import Locale

class PushCustomerOrder:
    connect = psycopg2.connect(dbname='DeliveryProductS',
                               user='postgres',
                               password='bn554540',
                               host='localhost')
    address_gen = Address(locale=Locale.RU)

    def __init__(self):
        self._orders = []
        self._client_ids = []
        self._basket_ids = []
        self._store_ids = []
        self._status_ids = []

    def get_ids_from_db(self):
        cursor = self.connect.cursor()

        print("Получаем client_id из client...")
        cursor.execute("SELECT client_id FROM client")
        self._client_ids = [row[0] for row in cursor.fetchall()]
```

```
        print("Получаем basket_id из basket...")
        cursor.execute("SELECT basket_id FROM basket")
        self._basket_ids = [row[0] for row in cursor.fetchall()]

        print("Получаем store_id из store...")
        cursor.execute("SELECT store_id FROM store")
        self._store_ids = [row[0] for row in cursor.fetchall()]

        print("Получаем status_id из order_status...")
        cursor.execute("SELECT status_id FROM order_status")
        self._status_ids = [row[0] for row in cursor.fetchall()]

        cursor.close()

    def create_data(self, n=200000):
        self.get_ids_from_db()

        if not all([self._client_ids, self._basket_ids, self._store_ids,
self._status_ids]):
            print("Ошибка: Одна из связанных таблиц пуста!")
            print(f"Найдено client_ids: {len(self._client_ids)}")
            print(f"Найдено basket_ids: {len(self._basket_ids)}")
            print(f"Найдено store_ids: {len(self._store_ids)}")
            print(f"Найдено status_ids: {len(self._status_ids)}")
            return False

        print("Создание данных для заказов...")
        start_date = datetime.now() - timedelta(days=365*2)  # Заказы за
последние 2 года

        for i in range(n):
            # Случайная дата в диапазоне 2 лет
            random_date = start_date +
timedelta(seconds=random.randint(0, 365*2*24*60*60))

            order = {
                'order_id': i,
                'client_id': random.choice(self._client_ids),
                'basket_id': random.choice(self._basket_ids),
                'store_id': random.choice(self._store_ids),
                'status_id': random.choice(self._status_ids),
                'creation_date': random_date,
                'address': self.address_gen.address(),
                'cost': round(random.uniform(100, 50000), 2),
                'total_weight': round(random.uniform(0.1, 50.0), 2)
            }
            self._orders.append(order)

            if i % 10000 == 0 and i != 0:
                print(f"Создано {i} записей")

        return True


    def push_to_db(self):
```

```
        """Вставляем данные в БД"""
        if not self._orders:
            print("Нет данных для вставки!")
            return

        cursor = self.connect.cursor()
        print("Начало вставки данных в БД...")

        try:
            for i, order in enumerate(self._orders):
                cursor.execute(
                    "INSERT INTO customer_order(order_id, client_id,
basket_id, store_id, status_id, "
                    "creation_date, address, total_cost, total_weight) "
                    "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)",
                    (order['order_id'], order['client_id'],
order['basket_id'],
                     order['store_id'], order['status_id'],
order['creation_date'],
                     order['address'], order['cost'],
order['total_weight'])
                )

                if i % 1000 == 0 and i != 0:
                    self.connect.commit()
                    print(f"Добавлено {i} записей")

            self.connect.commit()
            print("Все данные успешно добавлены!")

        except Exception as e:
            print(f"Ошибка при вставке: {e}")
            self.connect.rollback()
        finally:
            cursor.close()

    @classmethod
    def close_connection(cls):
        cls.connect.close()

def main():
    pusher = PushCustomerOrder()

    if pusher.create_data(n=1000000):
        pusher.push_to_db()

    pusher.close_connection()

if __name__ == "__main__":
    main()
import psycopg2
import random
from datetime import datetime

class PushDelivery:
    connect = psycopg2.connect(dbname='DeliveryProductS',
```

```
                            user='postgres',
                            password='bn554540',
                            host='localhost')

    def __init__(self):
        self._deliveries = []
        self._order_ids = []
        self._courier_ids = []
        self._status_ids = []

    def get_ids_from_db(self):
        """Получаем ID из связанных таблиц"""
        cursor = self.connect.cursor()

        # Получаем order_id из customer_order
        cursor.execute("SELECT order_id FROM customer_order")
        self._order_ids = [row[0] for row in cursor.fetchall()]

        # Получаем courier_id из courier
        cursor.execute("SELECT courier_id FROM courier")
        self._courier_ids = [row[0] for row in cursor.fetchall()]

        # Получаем status_id из order_status
        cursor.execute("SELECT status_id FROM order_status")
        self._status_ids = [row[0] for row in cursor.fetchall()]

        cursor.close()

    def create_data(self, n=200000):
        """Создаем данные для доставки"""
        self.get_ids_from_db()

        if not self._order_ids or not self._courier_ids or not
self._status_ids:
            print("Ошибка: Одна из связанных таблиц пуста!")
            print(f"Найдено order_ids: {len(self._order_ids)}")
            print(f"Найдено courier_ids: {len(self._courier_ids)}")
            print(f"Найдено status_ids: {len(self._status_ids)}")
            return False

        print("Создание данных для доставки...")
        for i in range(n):
            delivery = {
                'delivery_id': i,
                'order_id': random.choice(self._order_ids),
                'courier_id': random.choice(self._courier_ids),
                'status_id': random.choice(self._status_ids),
                'payment_status': random.choice([True, False])
            }
            self._deliveries.append(delivery)

            # Выводим прогресс каждые 10к записей
            if i % 10000 == 0 and i != 0:
                print(f"Создано {i} записей")

        return True
```

```
    def push_to_db(self):
        """Вставляем данные в БД"""
        if not self._deliveries:
            print("Нет данных для вставки!")
            return

        cursor = self.connect.cursor()
        print("Начало вставки данных в БД...")

        try:
            for i, delivery in enumerate(self._deliveries):
                cursor.execute(
                    "INSERT INTO delivery(delivery_id, order_id,
courier_id, status_id, payment_status) "
                    "VALUES (%s, %s, %s, %s, %s)",
                    (delivery['delivery_id'], delivery['order_id'],
delivery['courier_id'],
                     delivery['status_id'], delivery['payment_status'])
                )

                # Коммитим каждые 1000 записей
                if i % 1000 == 0 and i != 0:
                    self.connect.commit()
                    print(f"Добавлено {i} записей")

            # Финальный коммит
            self.connect.commit()
            print("Все данные успешно добавлены!")

        except Exception as e:
            print(f"Ошибка при вставке: {e}")
            self.connect.rollback()
        finally:
            cursor.close()

    @classmethod
    def close_connection(cls):
        cls.connect.close()

def main():
    pusher = PushDelivery()

    if pusher.create_data(n=200000):
        pusher.push_to_db()

    pusher.close_connection()

if __name__ == "__main__":
    main()
import psycopg2
import random
from datetime import datetime, timedelta
from mimesis import Text
from mimesis.locales import Locale
```

```
class PushMessages:
    connect = psycopg2.connect(dbname='DeliveryProductS',
                               user='postgres',
                               password='bn554540',
                               host='localhost')
    text_gen = Text(locale=Locale.RU)

    def __init__(self):
        self._messages = []
        self._client_ids = []
        self._order_ids = []
        self._courier_ids = []
        self._store_ids = []
        self._message_types = [
            'order_created', 'order_confirmed', 'payment_received',
            'order_assembled', 'courier_assigned', 'on_the_way',
            'arrived', 'delivered', 'delivery_problem', 'cancelled'
        ]

    def get_ids_from_db(self):
        """Получаем ID из связанных таблиц"""
        cursor = self.connect.cursor()

        print("Получаем client_id из client...")
        cursor.execute("SELECT client_id FROM client")
        self._client_ids = [row[0] for row in cursor.fetchall()]

        print("Получаем order_id из customer_order...")
        cursor.execute("SELECT order_id FROM customer_order")
        self._order_ids = [row[0] for row in cursor.fetchall()]

        print("Получаем courier_id из courier...")
        cursor.execute("SELECT courier_id FROM courier")
        self._courier_ids = [row[0] for row in cursor.fetchall()]

        print("Получаем store_id из store...")
        cursor.execute("SELECT store_id FROM store")
        self._store_ids = [row[0] for row in cursor.fetchall()]

        cursor.close()

    def generate_content(self, msg_type):
        """Генерируем осмысленный текст сообщения"""
        templates = {
            'order_created': [
                "Ваш заказ №{order_id} успешно создан",
                "Мы получили ваш заказ №{order_id}",
                "Заказ №{order_id} оформлен"
            ],
            'order_confirmed': [
                "Заказ №{order_id} подтвержден",
                "Подтверждаем прием вашего заказа №{order_id}",
                "Ваш заказ №{order_id} принят в обработку"
            ],
            'payment_received': [
                "Оплата за заказ №{order_id} получена",
```

```
                "Мы получили оплату по заказу №{order_id}",
                "Заказ №{order_id} оплачен"
            ],
            'order_assembled': [
                "Заказ №{order_id} собран и готов к отправке",
                "Ваш заказ №{order_id} упакован",
                "Товары по заказу №{order_id} подготовлены"
            ],
            'courier_assigned': [
                "Курьер назначен для доставки заказа №{order_id}",
                "Ваш заказ №{order_id} передан курьеру",
                "Курьер получил заказ №{order_id}"
            ],
            'on_the_way': [
                "Курьер с заказом №{order_id} в пути",
                "Заказ №{order_id} доставляется",
                "Ваш заказ №{order_id} уже едет к вам"
            ],
            'arrived': [
                "Курьер с заказом №{order_id} прибыл",
                "Заказ №{order_id} у вашего дома",
                "Курьер ожидает вас с заказом №{order_id}"
            ],
            'delivered': [
                "Заказ №{order_id} успешно доставлен",
                "Вы получили заказ №{order_id}",
                "Доставка заказа №{order_id} завершена"
            ],
            'delivery_problem': [
                "Проблема с доставкой заказа №{order_id}",
                "Возникли сложности с доставкой заказа №{order_id}",
                "Задержка доставки заказа №{order_id}"
            ],
            'cancelled': [
                "Заказ №{order_id} отменен",
                "Ваш заказ №{order_id} был отменен",
                "Отмена заказа №{order_id}"
            ]
        }

        template = random.choice(templates[msg_type])
        order_id = random.choice(self._order_ids)
        return template.format(order_id=order_id)


    def create_data(self, n=1000000):
        """Создаем данные для сообщений"""
        self.get_ids_from_db()

        if not all([self._client_ids, self._order_ids, self._courier_ids,
self._store_ids]):
            print("Ошибка: Одна из связанных таблиц пуста!")
            print(f"Найдено client_ids: {len(self._client_ids)}")
            print(f"Найдено order_ids: {len(self._order_ids)}")
            print(f"Найдено courier_ids: {len(self._courier_ids)}")
            print(f"Найдено store_ids: {len(self._store_ids)}")
```

```
            return False

        print("Создание данных для сообщений...")

        for i in range(n):
            msg_type = random.choice(self._message_types)

            message = {
                'message_id': i,
                'client_id': random.choice(self._client_ids),
                'order_id': random.choice(self._order_ids),
                'courier_id': random.choice(self._courier_ids),
                'store_id': random.choice(self._store_ids),
                'type': msg_type,
                'content': self.generate_content(msg_type)
            }
            self._messages.append(message)

            if i % 100000 == 0 and i != 0:
                print(f"Создано {i} записей")

        return True

    def push_to_db(self):
        """Вставляем данные в БД"""
        if not self._messages:
            print("Нет данных для вставки!")
            return

        cursor = self.connect.cursor()
        print("Начало вставки данных в БД...")

        try:
            for i, message in enumerate(self._messages):
                cursor.execute(
                    "INSERT INTO message(message_id, client_id, order_id,
courier_id, store_id, type, content) "
                    "VALUES (%s, %s, %s, %s, %s, %s, %s)",
                    (message['message_id'], message['client_id'],
message['order_id'],
                     message['courier_id'], message['store_id'],
message['type'], message['content'])
                )

                if i % 10000 == 0 and i != 0:
                    self.connect.commit()
                    print(f"Добавлено {i} записей")

            self.connect.commit()
            print("Все данные успешно добавлены!")

        except Exception as e:
            print(f"Ошибка при вставке: {e}")
            self.connect.rollback()
        finally:
            cursor.close()
```

```
    @classmethod
    def close_connection(cls):
        cls.connect.close()

def main():
    pusher = PushMessages()

    if pusher.create_data(n=1000000):
        pusher.push_to_db()

    pusher.close_connection()

if __name__ == "__main__":
    main()
```