Создание RAW слоя в ClickHouse

Представим себе ситуацию, что к нам обратился сетевой магазин "Пикча" с запросом сделать для них аналитическую систему, которая позволит им выгоднее и эффективней продавать товары.

В качестве исходных данных мы имеем контур заказчика NoSQL, куда хаотично льются данные со стороны всех магазинов.

Обозначим следующие магазины -

1. Большая Пикча (помещение более 200 кв.м.). Количество магазинов по стране - 30. Некоторые в одном городе.

2. Маленькая Пикча (помещение менее 100 кв.м.). Количество магазинов по стране - 15. Некоторые в одном городе.

Заказчик **НЕ СОГЛАСЕН** выдать доступ к NoSQL хранилищу. Предварительно он хочет увидеть демо-версию рабочего инструмента и лишь потом готов выдать доступ к контуру NoSQL

Внутри каждого магазина существует товары пяти основных продовольственных групп, а именно -

1. 🥖 **Зерновые и хлебобулочные изделия**
2. 🥩 **Мясо, рыба, яйца и бобовые**
3. 🥛 **Молочные продукты**
4. 🍎 **Фрукты и ягоды**
5. 🥦 **Овощи и зелень**

В каждую из групп входит не менее 20 позиций (итого 100 товаров в ассортименте). Так, например в овощи и зелень входят - шпинат, капуста, лук, чеснок и прочее.

У каждого товара есть свой набор атрибутов, так например выглядит схема какого-то творога.

```
{
  "id": "prd-0037",
  "name": "Творог 5%",
  "group": "Молочные",
  "description": "Творог из коровьего молока, 5% жирности",
  "kbju": {
    "calories": 121,
    "protein": 16.5,
    "fat": 5,
    "carbohydrates": 3.3
  },
  "price": 89.99,
  "unit": "упаковка",
  "origin_country": "Россия",
  "expiry_days": 12,
  "is_organic": false,
  "barcode": "4600111222333",
  "manufacturer": {
    "name": "ООО Молочный Комбинат",
    "country": "Россия",
    "website": "https://moloko.ru",
    "inn": "7723456789"
  }
}
```


Ниже дана расшифровка атрибутов от заказчика.

Поле	Тип	Описание
"id"	string	Уникальный идентификатор товара в системе. Например, prd-0037.
"name"	string	Название продукта: "Творог 5%".
"group"	string	Продовольственная группа. Здесь — "Молочные".
"description"	string	Краткое описание товара. В данном случае: "Творог из коровьего молока, 5% жирности".
"kbju"	object	Пищевая ценность на 100 г продукта (КБЖУ). Содержит вложенные поля:
calories	number	Энергетическая ценность — 121 ккал.
protein	number	Белки — 16.5 г.
fat	number	Жиры — 5 г.
carbohydrates	number	Углеводы — 3.3 г.
"price"	number	Стоимость товара за единицу: 89.99 руб.
"unit"	string	Единица измерения: "упаковка". Может быть также "кг", "л", "шт".
"origin_country"	string	Страна происхождения товара: "Россия".
"expiry_days"	integer	Срок годности в днях: 12 дней.
"is_organic"	boolean	Органический ли продукт: false — то есть, **не органический**.
"barcode"	string	Штрихкод товара (EAN-13): "4600111222333".
"manufacturer"	object	Информация о производителе, включает вложенные поля:
name	string	Название производителя: "ООО Молочный Комбинат".
country	string	Страна производителя: "Россия".
website	string (uri)	Сайт компании: "https://moloko.ru".
inn	string	ИНН производителя: "7723456789".

Портрет покупателя в магазине Пикча не представляется возможным без бонусной карты. То есть если покупатель пришел, купил что-то и ушел - то для магазина он никто в плане цифрового портрета. Схема покупателя выглядит следующим образом -

```
{
  "customer_id": "cus-102345",
  "first_name": "Алексей",
  "last_name": "Иванов",
  "email": "alexey.ivanov@example.com",
  "phone": "+7-900-123-45-67",
  "birth_date": "1990-04-15",
  "gender": "male",
  "registration_date": "2023-08-20T14:32:00Z",
  "is_loyalty_member": true,
  "loyalty_card_number": "LOYAL-987654321",
  "purchase_location": {
    "store_id": "store-001",
    "store_name": "Большая Пикча — Магазин на Тверской",
    "store_network": "Большая Пикча",
    "store_type_description": "Супермаркет площадью более 200 кв.м. Входит в
федеральную сеть из 30 магазинов.",
    "country": "Россия",
    "city": "Москва",
    "street": "ул. Тверская",
    "house": "15",
    "postal_code": "125009"
  },
  "delivery_address": {
    "country": "Россия",
    "city": "Москва",
    "street": "ул. Ленина",
    "house": "12",
    "apartment": "45",
    "postal_code": "101000"
  },
  "preferences": {
    "preferred_language": "ru",
    "preferred_payment_method": "card",
    "receive_promotions": true
  }
}
```


А вот и описание полей.

Поле	Описание
`customer_id`	Уникальный идентификатор клиента
`first_name` / `last_name`	Имя и фамилия
`email`	Адрес электронной почты
`phone`	Номер телефона
`birth_date`	Дата рождения (для акций, идентификации)
`gender`	Пол (`male`, `female`, `other`, `null`)
`registration_date`	Дата регистрации в системе
`is_loyalty_member`	Признак участия в программе лояльности
`loyalty_card_number`	Номер карты лояльности
`purchase_location`	Адрес магазина, где была совершена покупка
`store_id`	Идентификатор магазина
`store_name`	Название торговой точки
`delivery_address`	Адрес доставки (если покупка была оформлена онлайн)

Очевидно, что у каждого магазина существует свой набор атрибутов.


```
{
  "store_id": "store-001",
  "store_name": "Большая Пикча — Магазин на Тверской",
  "store_network": "Большая Пикча",
  "store_type_description": "Супермаркет площадью более 200 кв.м. Входит в федеральную сеть из 30 магазинов, некоторые из которых находятся в одном городе.",
  "type": "offline",
  "categories": [
    "🥖 Зерновые и хлебобулочные изделия",
    "🥩 Мясо, рыба, яйца и бобовые",
    "🥛 Молочные продукты",
    "🍎 Фрукты и ягоды",
    "🥦 Овощи и зелень"
  ],
  "manager": {
    "name": "Светлана Петрова",
    "phone": "+7-900-555-33-22",
    "email": "manager@tverskoy-store.ru"
  },
  "location": {
    "country": "Россия",
    "city": "Москва",
    "street": "ул. Тверская",
    "house": "15",
    "postal_code": "125009",
    "coordinates": {
      "latitude": 55.7575,
      "longitude": 37.6156
    }
  },
  "opening_hours": {
    "mon_fri": "09:00-21:00",
    "sat": "10:00-20:00",
    "sun": "10:00-18:00"
  },
  "accepts_online_orders": true,
  "delivery_available": true,
  "warehouse_connected": true,
  "last_inventory_date": "2025-06-28"
}
```


И, конечно же, нам нужны покупки, которые купил покупатель в определенном магазине.


```
{
  "purchase_id": "ord-20250710-001",
  "customer": {
    "customer_id": "cus-102345",
    "first_name": "Алексей",
    "last_name": "Иванов",
    "email": "alexey.ivanov@example.com",
    "phone": "+7-900-123-45-67",
    "is_loyalty_member": true,
    "loyalty_card_number": "LOYAL-987654321"
  },
  "store": {
    "store_id": "store-001",
    "store_name": "Большая Пикча — Магазин на Тверской",
    "store_network": "Большая Пикча",
```

```
    "store_type_description": "Супермаркет площадью более 200 кв.м. Входит в федеральную сеть из 30 магазинов.",
    "location": {
      "city": "Москва",
      "street": "ул. Тверская",
      "house": "15",
      "postal_code": "125009"
    }
  },
  "items": [
    {
      "product_id": "prd-0037",
      "name": "Творог 5%",
      "category": "🥛 Молочные продукты",
      "quantity": 2,
      "unit": "упаковка",
      "price_per_unit": 89.99,
      "total_price": 179.98,
      "kbju": {
        "calories": 121,
        "protein": 16.5,
        "fat": 5,
        "carbohydrates": 3.3
      },
      "manufacturer": {
        "name": "ООО Молочный Комбинат",
        "country": "Россия",
        "website": "https://moloko.ru",
        "inn": "7723456789"
      }
    },
    {
      "product_id": "prd-0012",
      "name": "Хлеб ржаной",
      "category": "🌾 Зерновые и хлебобулочные изделия",
      "quantity": 1,
      "unit": "булка",
      "price_per_unit": 45.00,
      "total_price": 45.00
    }
  ],
  "total_amount": 224.98,
  "payment_method": "card",
  "is_delivery": true,
  "delivery_address": {
    "city": "Москва",
    "street": "ул. Ленина",
    "house": "12",
    "apartment": "45",
    "postal_code": "101000"
  },
  "purchase_datetime": "2025-07-10T17:45:00Z"
}
```

Ну и куда же мы без расшифровки.

Блок	Содержит
customer	ID клиента, имя, карта лояльности и контакты
store	Название, сеть, местоположение
items	Список товаров с количеством, КБЖУ, производителем
total_amount	Общая сумма заказа
payment_method	Способ оплаты (card, cash, online_wallet, …)
is_delivery	Было ли оформлено на доставку
delivery_address	Адрес, если доставка выбрана
purchase_datetime	Дата и время покупки в ISO-формате

Таким образом мы получаем структуру - магазин, клиент, товары, покупки. **И, увы, все это лежит вместе в NoSQL хранилище.** Поэтому на текущий момент необходимо разработать схемы хранения данных и выбрать для этого какое-то хранилище + выбрать корректные типы данных.

От Вас требуется наличие :

1. 45 JSON файлов, описывающих каждый магазин. Структура - Выше.
2. Необходимо создать не менее 20 JSON файлов, которые будут описывать товары - структура выше.
3. Необходимо создать минимум 1 покупателя в каждом магазине. Структура - Выше.
4. Не менее 200 покупок от различных покупателей в разных магазинах. Структура выше.

В качестве генератора можно рассмотреть что-то такое -

```
import os
import json
import random
import uuid
from datetime import datetime, timedelta
from faker import Faker

fake = Faker("ru_RU")

# Создание директорий
os.makedirs("data/stores", exist_ok=True)
os.makedirs("data/products", exist_ok=True)
os.makedirs("data/customers", exist_ok=True)
os.makedirs("data/purchases", exist_ok=True)

# Категории
categories = [
    "🥖 Зерновые и хлебобулочные изделия",
    "🥩 Мясо, рыба, яйца и бобовые",
    "🥛 Молочные продукты",
    "🍎 Фрукты и ягоды",
    "🥦 Овощи и зелень"
]
```

```
store_networks = [("Большая Пикча", 30), ("Маленькая Пикча", 15)]
stores = []

# === 1. Генерация магазинов ===
for network, count in store_networks:
    for i in range(count):
        store_id = f"store-{len(stores)+1:03}"
        city = fake.city()
        store = {
            "store_id": store_id,
            "store_name": f"{network} — Магазин на {fake.street_name()}",
            "store_network": network,
            "store_type_description": f"{'Супермаркет более 200 кв.м.' if
network == 'Большая Пикча' else 'Магазин у дома менее 100 кв.м.'} Входит в
сеть из {count} магазинов.",
            "type": "offline",
            "categories": categories,
            "manager": {
                "name": fake.name(),
                "phone": fake.phone_number(),
                "email": fake.email()
            },
            "location": {
                "country": "Россия",
                "city": city,
                "street": fake.street_name(),
                "house": str(fake.building_number()),
                "postal_code": fake.postcode(),
                "coordinates": {
                    "latitude": float(fake.latitude()),
                    "longitude": float(fake.longitude())

                }
            },
            "opening_hours": {
                "mon_fri": "09:00-21:00",
                "sat": "10:00-20:00",
                "sun": "10:00-18:00"
            },
            "accepts_online_orders": True,
            "delivery_available": True,
            "warehouse_connected": random.choice([True, False]),
            "last_inventory_date": datetime.now().strftime("%Y-%m-%d")
        }
        stores.append(store)
        with open(f"data/stores/{store_id}.json", "w", encoding="utf-8") as
f:
            json.dump(store, f, ensure_ascii=False, indent=2)

# === 2. Генерация товаров ===
products = []
for i in range(20):
    product = {
        "id": f"prd-{1000+i}",
        "name": fake.word().capitalize(),
        "group": random.choice(categories),
        "description": fake.sentence(),
        "kbju": {
            "calories": round(random.uniform(50, 300), 1),
            "protein": round(random.uniform(0.5, 20), 1),
            "fat": round(random.uniform(0.1, 15), 1),
            "carbohydrates": round(random.uniform(0.5, 50), 1)
```

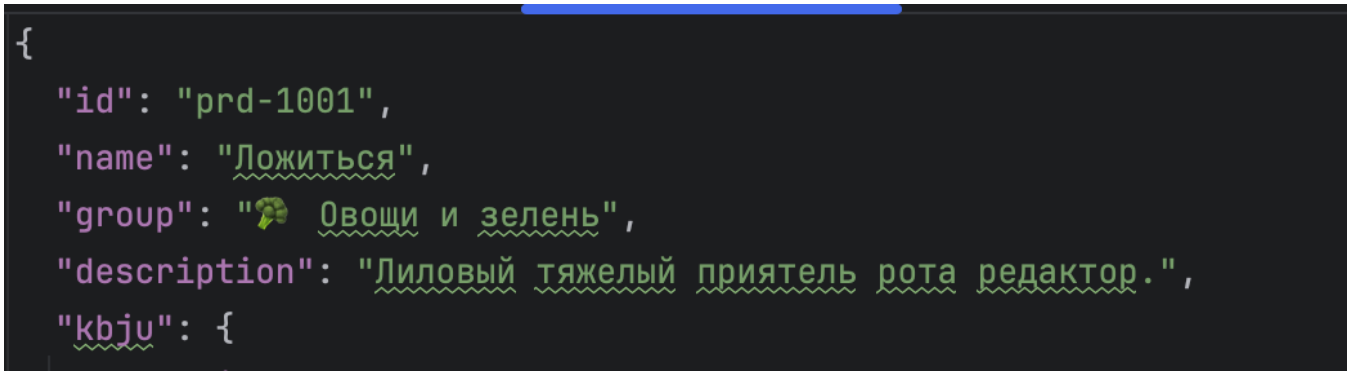
```
        },
        "price": round(random.uniform(30, 300), 2),
        "unit": "шт",
        "origin_country": "Россия",
        "expiry_days": random.randint(5, 30),
        "is_organic": random.choice([True, False]),
        "barcode": fake.ean(length=13),
        "manufacturer": {
            "name": fake.company(),
            "country": "Россия",
            "website": f"https://{fake.domain_name()}",
            "inn": fake.bothify(text='##########')
        }
    }
    products.append(product)
    with open(f"data/products/{product['id']}.json", "w", encoding="utf-8")
as f:
        json.dump(product, f, ensure_ascii=False, indent=2)

# === 3. Генерация покупателей ===
customers = []
for store in stores:
    customer_id = f"cus-{1000 + len(customers)}"
    customer = {
        "customer_id": customer_id,
        "first_name": fake.first_name(),
        "last_name": fake.last_name(),
        "email": fake.email(),
        "phone": fake.phone_number(),
        "birth_date": fake.date_of_birth(minimum_age=18,
maximum_age=70).isoformat(),
        "gender": random.choice(["male", "female"]),
        "registration_date": datetime.now().isoformat(),
        "is_loyalty_member": True,
        "loyalty_card_number": f"LOYAL-{uuid.uuid4().hex[:10].upper()}",
        "purchase_location": store["location"],
        "delivery_address": {
            "country": "Россия",
            "city": store["location"]["city"],
            "street": fake.street_name(),
            "house": str(fake.building_number()),
            "apartment": str(random.randint(1, 100)),
            "postal_code": fake.postcode()
        },
        "preferences": {
            "preferred_language": "ru",
            "preferred_payment_method": random.choice(["card", "cash"]),
            "receive_promotions": random.choice([True, False])
        }
    }
    customers.append(customer)
    with open(f"data/customers/{customer_id}.json", "w", encoding="utf-8")
as f:
        json.dump(customer, f, ensure_ascii=False, indent=2)

# === 4. Генерация покупок ===
for i in range(200):
    customer = random.choice(customers)
    store = random.choice(stores)
    items = random.sample(products, k=random.randint(1, 3))
    purchase_items = []
    total = 0
```

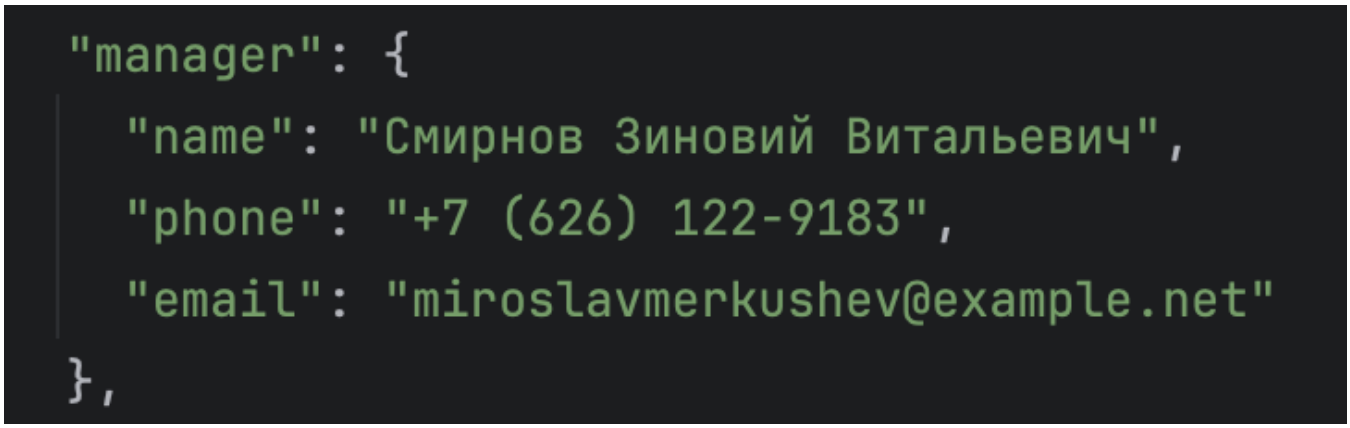
```
    for item in items:
        qty = random.randint(1, 5)
        total_price = round(item["price"] * qty, 2)
        total += total_price
        purchase_items.append({
            "product_id": item["id"],
            "name": item["name"],
            "category": item["group"],
            "quantity": qty,
            "unit": item["unit"],
            "price_per_unit": item["price"],
            "total_price": total_price,
            "kbju": item["kbju"],
            "manufacturer": item["manufacturer"]
        })
    purchase = {
        "purchase_id": f"ord-{i+1:05}",
        "customer": {
            "customer_id": customer["customer_id"],
            "first_name": customer["first_name"],
            "last_name": customer["last_name"]
        },
        "store": {
            "store_id": store["store_id"],
            "store_name": store["store_name"],
            "store_network": store["store_network"],
            "location": store["location"]
        },
        "items": purchase_items,
        "total_amount": round(total, 2),
        "payment_method": random.choice(["card", "cash"]),
        "is_delivery": random.choice([True, False]),
        "delivery_address": customer["delivery_address"],
        "purchase_datetime": (datetime.now() -
timedelta(days=random.randint(0, 90))).isoformat()
    }
    with open(f"data/purchases/{purchase['purchase_id']}.json", "w",
encoding="utf-8") as f:
        json.dump(purchase, f, ensure_ascii=False, indent=2)
```

Получится вот такая директория, но качество данных пострадает (ничего трогать не нужно ручками) -

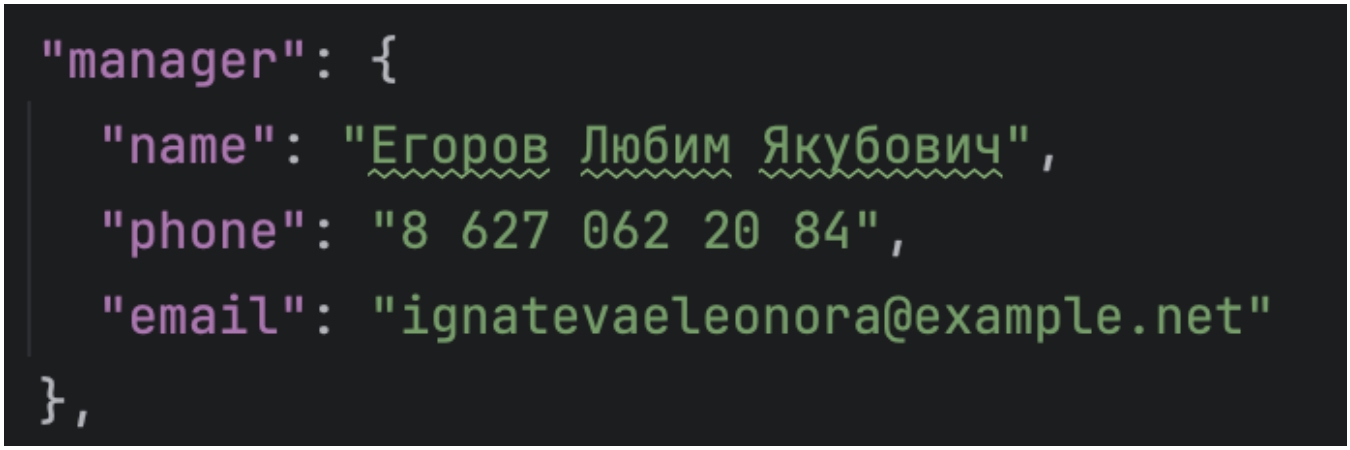


После того, как пункты выше будут выполнены необходимо реализовать следующую задачу -

1. Добавить все JSON файлы в NoSQL хранилище (Docker). Реализовать это необходимо при помощи скрипта на Python, который залезет в локальную директорию и заберет оттуда все JSON файлы для загрузки. Таким образом мы смоделируем хранилище заказчика.
2. Далее необходимо при помощи Kafka загрузить эти данные в RAW (сырое) хранилище. Необходимо взять ClickHouse. Важно, чтобы каждая таблица отвечала за что-то свое и могла джойнится с другой. Так, например, очевидно, что покупатели будут связаны с магазинами, покупками и товарами. **Важно, что в Clickhouse данные прилетают в том виде в котором они лежали у заказчика, так как это RAW хранилище.** А это значит, что номера -



и



Будут загружены, как STRING. Параметры Kafka - забрать только то, что лежит в хранилище.

Таким образом в ходе выполнения этой части задания мы получим данные от заказчика в своей системе. Но, кажется мы что-то забыли.

3. Персональная информация (телефон и почта) должны быть зашифрованы любым удобным способом до загрузки их в Clickhouse. Более того необходимо предусмотреть приведение только этих двух полей к нормальному единому виду.

Таким образом, мы получим РЕАЛЬНЫЙ дата инженерный опыт по загрузке данных из хранилища заказчика в наше хранилище посредством использования Kafka/Python/Clickhouse/NoSQL.

Ну и конечно же, чтобы было удобней проверить данное задание, необходимо выполнить 4 пункт.

4. Постройте дашборд в графане на основе которого можно будет сделать вывод о том, что количество покупок действительно 200, а магазинов 45.

Критерии проверки следующие -

1. Есть GIT репозиторий в котором описываются участники команды.
2. Есть генератор файлов JSON, есть загрузчик этих файлов JSON в NoSQL.
3. Есть docker compose с kafka, clickhouse, noSQL, grafana.
4. Есть дашборд в Grafana (нужен скриншот).

Clickhouse RAW | Clickhouse MART

После того, как выполнена первая часть задания и заказчик удостоверяется в том, что у нас действительно есть ресурсы для сохранения его данных , они лежат в нормальном виде и визуализируются, нам необходимо разработать внутренний ETL пайплайн, который будет предусматривать -

Взятие данных из Clickhouse, SQL будет очищать их и переносить в другую БД Clickhouse. Скрипт должен быть написан полностью на SQL и должен включаться в момент вставки данных в сырые таблицы. В него дефолтно будут входить -

- проверка дубликатов
- проверка пустых строк
- проверка NULL
- проверка адекватности значений - то есть дата покупки, дата рождения - не должны быть позже текущего дня.
- все данные должны быть приведены в нижний регистр

В случае превышения > 50 % дубликатов в исходных таблицах - алертить в телеграмм при помощи Grafana.

Критерии проверки следующие -

1. В гите указан телеграмм бот (название, скриншот его работы)
2. Приложен SQL скрипт, который выполняет очистку данных и делает это при помощи MV.

3. В Grafana реализован алертинг дубликатов. (скриншот)

Таким образом здесь мы научимся очищать первородные данные и готовить их к дальнейшей обработке.

Clickhouse MART | Традиционный ETL

Как только данные преобразованы в адекватный вид - самое время начать решать бизнес задачи заказчика. А условие будет следующее -

1. Необходимо построить ETL процесс на PySpark (локально).
2. Данные для построения витрины необходимо забирать из Clickhouse MART (который у нас в Docker)
3. В ходе создания витрины будут созданы несколько полей, а именно матрица признаков, которая позволит выявлять покупательские группы (кластеризация) -

№	Признак	Описание
1	`bought_milk_last_30d`	Покупал молочные продукты за последние 30 дней
2	`bought_fruits_last_14d`	Покупал фрукты и ягоды за последние 14 дней
3	`not_bought_veggies_14d`	Не покупал овощи и зелень за последние 14 дней
4	`recurrent_buyer`	Делал более 2 покупок за последние 30 дней
5	`inactive_14_30`	Не покупал 14–30 дней (ушедший клиент?)
6	`new_customer`	Покупатель зарегистрировался менее 30 дней назад
7	`delivery_user`	Пользовался доставкой хотя бы раз
8	`organic_preference`	Купил хотя бы 1 органический продукт
9	`bulk_buyer`	Средняя корзина > 1000₽
10	`low_cost_buyer`	Средняя корзина < 200₽
11	`buys_bakery`	Покупал хлеб/выпечку хотя бы раз
12	`loyal_customer`	Лояльный клиент (карта и ≥3 покупки)
13	`multicity_buyer`	Делал покупки в разных городах
14	`bought_meat_last_week`	Покупал мясо/рыбу/яйца за последнюю неделю
15	`night_shopper`	Делал покупки после 20:00
16	`morning_shopper`	Делал покупки до 10:00

17	`prefers_cash`	Оплачивал наличными ≥ 70% покупок
18	`prefers_card`	Оплачивал картой ≥ 70% покупок
19	`weekend_shopper`	Делал ≥ 60% покупок в выходные
20	`weekday_shopper`	Делал ≥ 60% покупок в будни
21	`single_item_buyer`	≥50% покупок — 1 товар в корзине
22	`varied_shopper`	Покупал ≥4 разных категорий продуктов
23	`store_loyal`	Ходит только в один магазин
24	`switching_store`	Ходит в разные магазины
25	`family_shopper`	Среднее кол-во позиций в корзине ≥4
26	`early_bird`	Покупка в промежутке между 12 и 15 часами дня
27	`no_purchases`	Не совершал ни одной покупки (только регистрация)
28	`recent_high_spender`	Купил на сумму >2000₽ за последние 7 дней
29	`fruit_lover`	≥3 покупок фруктов за 30 дней
30	`vegetarian_profile`	Не купил ни одного мясного продукта за 90 дней

И, как говорится - наша задача собрать и отдать заказчику. А его аналитики уже сделают какие-то выводы из этого. Пример строки в итоговой таблице (такой формат дан для удобства чтения). 0 или 1 заменяемы на True/False - на Ваше усмотрение -


```
{
  "customer_id": "cus-102345",
  "bought_milk_last_30d": 1,
  "bought_fruits_last_14d": 1,
  "not_bought_veggies_14d": 0,
  "recurrent_buyer": 1,
  "inactive_14_30": 0,
  "new_customer": 0,
  "delivery_user": 1,
  "organic_preference": 1,
  "bulk_buyer": 0,
  "low_cost_buyer": 1,
  "buys_bakery": 1,
  "loyal_customer": 1,
  "multicity_buyer": 0,
  "bought_meat_last_week": 0,
  "night_shopper": 1,
  "morning_shopper": 0,
  "prefers_cash": 0,
  "prefers_card": 1,
  "weekend_shopper": 1,
  "weekday_shopper": 0,
```

```
  "single_item_buyer": 0,
  "varied_shopper": 1,
  "store_loyal": 1,
  "switching_store": 0,
  "family_shopper": 1,
  "early_bird": 0,
  "no_purchases": 0,
  "recent_high_spender": 1,
  "fruit_lover": 1,
  "vegetarian_profile": 1
}
```


Задание со звездочкой предусматривает создание обертки для этого процесса в Airflow. Расписание 10:00. Данные всегда берем все! То есть если в данных заказчика были внесены изменения - мы всегда будем получать актуальную информацию!
И, наконец, данные необходимо куда-то отправить - пусть это будет S3 (можно использовать Minio в Docker), где будет лежать файл CSV в формате analytic_result_2025_08_01. Количество столбцов = количество полей указанных в ТЗ.

Критерии проверки следующие -

1. Есть ETL процесс на PySpark. Реализована логика хотя бы 10 полей из 30. (полный код в GIT)
2. Файл формируется и загружается в S3 (MINIO). (скриншоты)