АРХИТЕКТУРА РЕШЕНИЯ.

Исполнитель: Асет Иманкулов

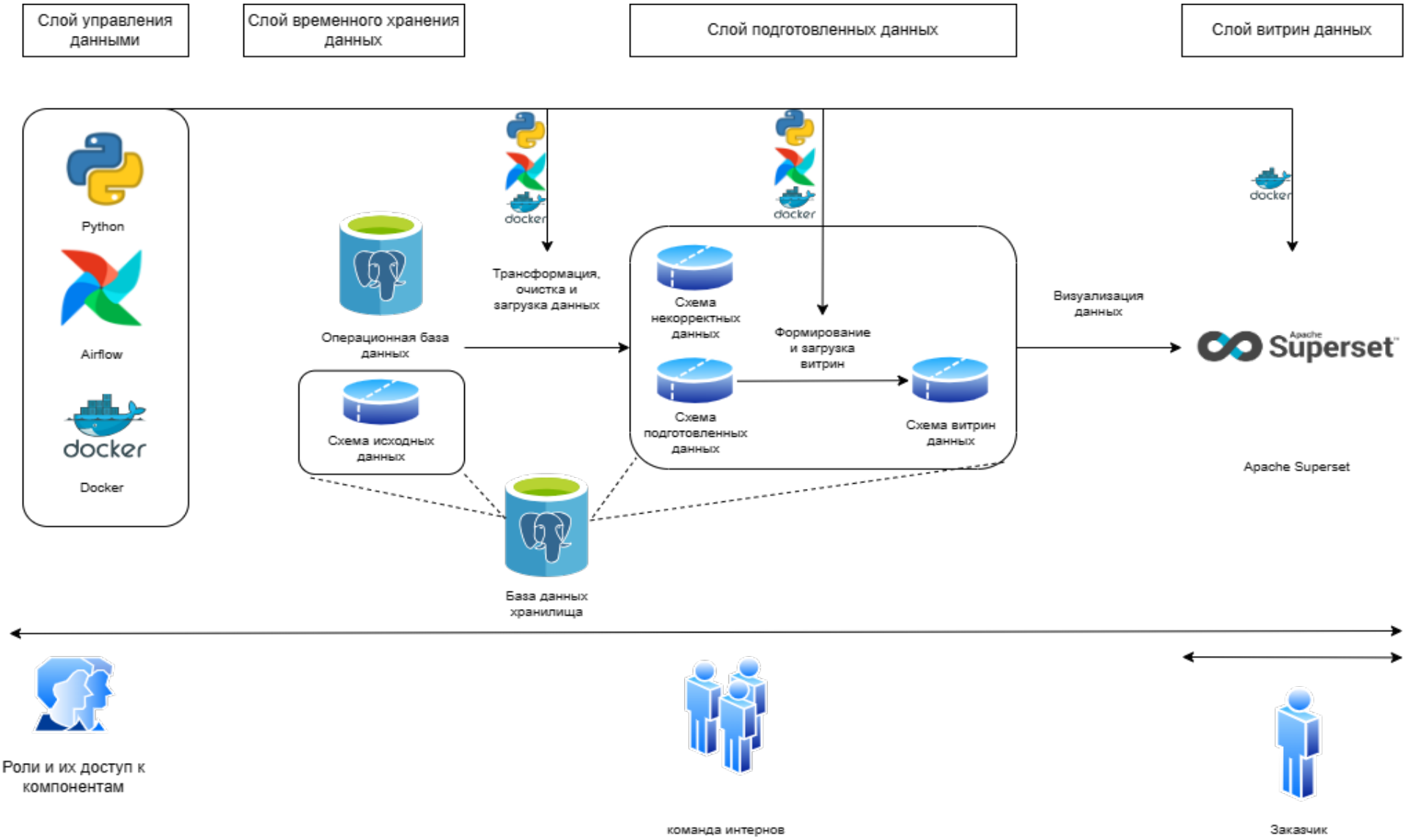
Должность: разработчик

Дата: 31.07.2023

Версия документа: 3.0

Оглавление

1. СХЕМА АРХИТЕКТУРНОГО РЕШЕНИЯ.



2. ОПИСАНИЕ КОМПОНЕНТОВ.

2.1. Наименование баз данных и их элементов в информационной системе

База данных	**Наименование**
Операционная база данных	internship_sources
База данных хранилища	internship_7_db

База данных	**Схема**	**Наименование**
Операционная база данных	Схема исходных данных	sources
База данных хранилища	Схема исходных данных	sources
	Схема подготовленных данных	dds
	Схема некорректных данных	data_quality
	Схема витрин данных	datamarts

2.2. Версии используемого ПО.

Программное обеспечение	**Версия программного обеспечения**
PostgreSQL	12.10
Python	3.10
Docker (клиент)	24.0.2
Docker (сервер)	24.0.2
Apache Airflow	2.6.2
Apache Superset	2.1.0

2.3. Параметры подключения к компонентам.

Подключение к базам данных:

База данных	**Хост**	**Порт**	**Пользователь**	**Схема**
internship_sources	10.1.108.29	5432	interns_7	sources
internship_7_db	10.1.108.29	5432	interns_7	sources
				dds
				data_quality
				datamarts

Подключение к Apache Airflow:

Параметр	**Значение**
URL вэб-интерфейса	http://localhost:8080/home
Логин	interns_7
Путь к ДАГам	../docker/airflow/dags/
Путь к исполняемому скрипту	../docker/airflow/docker-compose.yml

Подключение к Apache Superset:

Параметр	**Значение**
URL вэб-интерфейса	http://localhost:9000/superset/welcome/
Логин	interns_7
Путь к исполняемому скрипту	../docker/superset/docker-compose-non-dev.yml

2.4. Используемые сущности.

Слой	**База данных**	**Схема**	**Сущность**	**Тип**
Слой временного хранения данных	internship_sources	sources	brand	таблица
			category	таблица
			product	таблица
			stock	таблица
			transaction	таблица
	internship_7_db	sources	product_quantity	таблица
			stores	таблица
			stores_emails	таблица
			transaction_stores	таблица
Слой подготовленных данных	internship_7_db	dds	brand	таблица
			category	таблица
			product	таблица
			stock	таблица
			transaction	таблица
			product_quantity	таблица
			stores	таблица
			stores_emails	таблица
			transaction_stores	таблица
		data_quality	brand	таблица
			category	таблица
			product	таблица
			stock	таблица
			transaction	таблица
			product_quantity	таблица
			stores	таблица
			stores_emails	таблица
			transaction_stores	таблица
			upload_tables_tech	таблица
Слой витрин данных	internship_7_db	datamarts	orders_data	таблица
			stock_data	таблица
			stores_data	таблица
			orders_data_mart	представление
			stock_data_mart	представление
			stores_data_mart	представление

3. ОПИСАНИЕ СЛОЕВ.

Архитектурное решение представлено четырьмя слоями: слой управления данных, слой временного хранения данных, слой подготовленных данных, слой витрин данных.

В слое управления данным указаны технологии, осуществляющие обработку данных. Docker осуществляет контейнеризацию Airflow с Python. При этом, Airflow оркестрирует исполнение двух Python-скриптов, содержащих логику обработки данных, и двух python-скриптов, предназначенных для запуска DAG'ов. Первый скрипт предназначен для трансформации и очистки исходных данных и их последующую загрузку в схему подготовленных данных операционной базы данных. Данные, непрошедшие проверку качества попадают в таблицы логирования, размещенные на схеме некорректных данных.

Второй python-скрипт на основе полученных данных формирует и загружает витрины данных. Формирование витрин происходит в два этапа: сперва создаются таблицы соединения, а затем, на их основе, создаются представления. Образованные витрины данных визуализируются с помощью инструмента создания чартов и дашбордов Superset, который, в свою очередь, контейнеризирован с помощью Docker.

Слой временного хранения данных представлен операционной базой (internship_sources), в которой хранятся исходные «сырые» данные и схемой «sources», размещенной на базе данных хранилища.

Слой подготовленных данных содержит базу данных хранилища (internship_7_db) с тремя схемами – схема подготовленных данных (dds), схема некорректных данных (data_quality) и схема витрин данных (datamarts).

ETL-процесс

Содержание проекта

* Директории:

1. AIRFLOW - содержит информацию об используемых переменных, конфигурационные файлы запуска Apache Airflow с помощью Docker, директории с DAG'ами и скриптами обработки данных.

2. POSTGRES - содержит DDL-скрипты, предназначенный для формирования всех сущностей DDS и datamarts-слоев.

4. DOCUMENTATION - содержит документацию по проекту.

* Файлы:

1. AIRFLOW/.env - информация о переменных среды AIRFLOW_UID и PYTHONPATH.

2. AIRFLOW/docker-compose.yml - измененный скрипт запуска Apache Airflow с помощью Docker с примонтированной директорией, хранящей python-скрипты.

3. AIRFLOW/variables.txt - используемые переменные в Apache Airflow.

4. AIRFLOW/dags/sources_to_dds.py - python-скрипт с DAG-ом, запускающим процедуру проверки качества данных.

5. AIRFLOW/dags/dds_to_dm.py - python-скрипт с DAG-ом, запускающим процедуру создания витрин данных.

6. AIRFLOW/dags/select_data_dag.py - python-скрипт с DAG-ом, запускающим процедуру возвращения заданного количества строк из целевой таблицы.

8. AIRFLOW/scripts/data_quality.py - python-скрипт с классом для проверки качества данных и загрузки данных из одной базы данных в другую.

9. AIRFLOW/scripts/data_marts.py - python-скрипт с классом для создания таблиц и витрин данных.

10. AIRFLOW/scripts/data_quality_params.json - набор параметров, используемый при наполнении данными dds-слоя.

11. AIRFLOW/scripts/data_marts_params.json - набор параметров, используемый при наполнении данными datamarts-слоя.

12. POSTGRES/DDL.sql - SQL-скрипт, с процедурами создания сущностей на слое DDS

13. POSTGRES/DDL.sql - SQL-скрипт, с процедурами создания сущностей на слое datamarts

14. DOCUMENTATION/АРХИТЕКТУРА РЕШЕНИЯ.docx - информация об архитектуре решения.

Хранилище данных.

Хранилище данных представлено четырьмя слоями: слой управления данных, слой временного хранения данных,

слой подготовленных данных, слой витрин данных.

В слое управления данным указаны технологии, осуществляющие обработку данных. Docker осуществляет контейнеризацию Airflow с Python. При этом, Airflow оркестрирует исполнение двух Python-скриптов, содержащих логику обработки данных, и двух python-скриптов, предназначенных для запуска DAG’ов.

Первый скрипт предназначен для трансформации и очистки исходных данных и их последующую загрузку в схему подготовленных данных операционной базы данных. Данные, непрошедшие проверку качества попадают в таблицы логирования, размещенные на схеме некорректных данных.

Второй python-скрипт на основе полученных данных формирует и загружает витрины данных. Формирование витрин происходит в два этапа: сперва создаются таблицы соединения, а затем, на их основе, создаются представления. Образованные витрины данных визуализируются с помощью инструмента создания чартов и дашбордов Superset, который, в свою очередь, контейнеризирован с помощью Docker.

![Схема архитектурного решения](https://github.com/asetimankulov/internship/assets/98170451/52a84b9c-4a53-4255-94b0-24d0a8ed59cf)

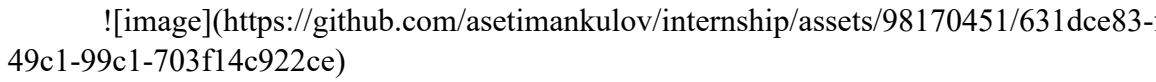
Трансформация, очистка и загрузка данных.

Первый набор python-скриптов, предназначенный для трансформации, очистки и загрузки исходных "сырых" данных, содержит DAG, запускающий процедуры

проверки качества загружаемых данных и их преобразования, и файл с классом проверки качества данных и процедурами загрузки и обработки данных.

DAG запускает исполнение 11 тасок. При этом, параллельное исполнение всех тасок нереализуемо в виду наличия ограничений на ссылочную целостность

связанных таблиц. Поэтому реализована следующая последовательность исполнения тасок:



где:

* end_step - оператор, исполняющийся в случае успешного завершения всех тасок.

* remove_all_data - оператор Postgres, очищающий все таблицы на слое DDS.

* brand_upload, category_upload, stores_upload, product_upload, transaction_stores_upload, product_quantity_upload, stock_upload, stores_emails_uload, transaction_upload - операторы Python, исполняющие процедуры проверки качества соответствующих таблиц.

Модуль data_quality содержит класс проверки данных таблицы по различным критериям качества данных.

Класс содержит следующие методы проверки качества данных:

* noise_restricts_check - логирование и устранение "шумов" в данных. С помощью регулярного выражения выявляется наличие/отсутствие "шумов"

и происходит их устранение.

* data_types_check - типы обрабатываемых данных проверяются на их соответствие полям таблицы загрузки. Некорректные данные заменяются

на пропуск и логируются.

* missing_values_check - проверка наличия/отсутствия пропусков и их обработка. Отсутствие данных по полям первичного ключа приводит к устранению данных, в остальных случаях - данные остаются без изменений или заменяются на указанное значение.

* duplicates_check - устранение дубликатов по всем полям или первичному ключу. Некорректные данные логируются.

* value_restrict_check - проверяет соответствие значения указанного поля таблицы некоторым ограничениям. Результат несоответствия логируется.

* len_restricts_check - проверяет длину указанного поля и логирует те строки, которые нарушают ограничение.

* ref_integrity - проверка, устраняющая из проверяемой таблицы те строки по указанному полю, которые отсутствуют в связанном наборе данных.

Для каждой таблицы сформированы свои требования к качеству данных. Хранение набора параметров для каждой таблицы реализовано следующим образом:

```
    "category": {
```

```
"missing": {
                "drop": ["category_id"],
                "fill": {"category_name": "Категория не определена"}
        },
"duplicate": {
                "drop": ["category_id"],
                "log": ["category_id"]
        },
"noise": {"category_name": {"regex": "_", "match_replace": {"_": " "}}},
"len_restrict": {"category_name": {"min": 2, "max": null}},
"data_types": {
                "category_id": "text",
                "category_name": "text"
        },
        "ref_integrity": null,
        "val_restrict": null
  }
```


В случае с таблицей "category", проверка качества реализована следующим образом:

* Обработка пропусков ("missing"): пропуск по "category_id" приводит к удалению строки. Пропуски по полю "category_name" заполняются значением "Категория не определена".

* Обработка дубликатов: дублем считается строка, совпадающая по полю "category_id". Они же логируются.

* Обработка шумов ("noise"): значения поля "category_name" проверяются регулярным выражением "_". Некорректный символ "_" заменяется на " ".

* Обработка длины значений ("len_restrict"): значение по полю "category_name" будет залоггировано в случае, если его длина < 2.

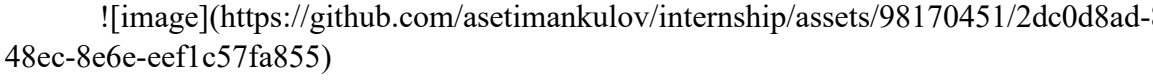
* Обработка типов данных ("data_types"): значения полей "category_id" и "category_name" проверяются на соответствие типу данных text.

* Обеспечение ссылочной целостности ("ref_integrity"): нет наборов данных (таблиц), на которые бы ссылался какой-либо внешний ключ.

Формирование и загрузка витрин.

Второй набор python-скриптов, предназначенный для формирования и загрузки витрин, содержит DAG, запускающий процедуры создания витрин, и файл с классом формирования таблиц, на основе которых будут созданы представления.

DAG запускает исполнение 7 тасок. При этом, осуществлено параллельное исполнение тасок, наполняющих таблицы, сформированных на основе данных из dds-слоя. Реализована следующая последовательность исполнения тасок:



где:

* remove_all_dm_data - оператор Postgres, очищающий все таблицы на слое datamarts.

* orders_data_upload, stock_data_upload, stores_data_upload - операторы Python, исполняющие процедуры наполнения соответствующих таблиц. При этом, формирование таблицы stores_data осуществляется на основе выборки набора полей из наполняемой таблицы stock_data_upload.

* create_orders_view, create_stock_view, create_stores_view - создание представления на основе наполняемых таблиц.

Модуль data_marts содержит класс формирования таблиц данных, на основе которых формируются витрины данных.

Класс содержит следующие методы формирования таблиц:

* create_mart - формирование набора данных и его загрузка в целевую таблицу

* orders_data - формирование таблицы orders_data

* stock_data - формирование таблицы stock_data

* stores_data - формирование таблицы stores_data

Для каждой таблицы сформированы свои параметры формирования. Хранение набора параметров для каждой таблицы реализовано следующим образом:

```
    "stock_data": {
  "conn_info": {
    "from": {"conn_id": "dds_id", "schema": "dds"},
    "to": {"conn_id": "dm_id", "schema": "datamarts"}
  },
  "join_info": {
  "source": {"table": "stock", "rename": true},
  "joined_tables": [
    {"table": "stores",
      "how": "inner",
      "left_on": ["stock_pos"],
      "right_on": ["stores_pos"],
      "cast": null,
      "rename": true},
    {"table": "product",
      "how": "inner",
      "left_on": ["stock_product_id"],
      "right_on": ["product_product_id"],
```

```
    "cast": null,
    "rename": true},
   {"table": "category",
    "how": "inner",
    "left_on": ["product_category_id"],
    "right_on": ["category_category_id"],
    "cast": null,
    "rename": true},
   {"table": "brand",
    "how": "inner",
    "left_on": ["product_brand_id"],
    "right_on": ["brand_brand_id"],
    "cast": null,
    "rename": true},
   {"table": "product_quantity",
    "how": "inner",
    "left_on": ["stock_product_id"],
    "right_on": ["product_quantity_product_id"],
    "cast": null,
    "rename": true},
   {"table": "stores_emails",
    "how": "inner",
    "left_on": ["stock_pos"],
    "right_on": ["stores_emails_pos"],
    "cast": null,
    "rename": true}
]
```

```
},
 "column_names": {
  "stock_available_on": "Дата наличия товара",
  "stores_pos": "ID магазина",
  "stores_pos_name": "Магазин",
  "stock_product_id": "ID товара",
  "product_name_short": "Товар",
  "stock_available_quantity": "Доступное количество товара, шт.",
  "stock_cost_per_item": "Закупочная цена товара, руб.",
  "available_amount": "Сумма доступного остатка, руб.",
  "category_category_id": "ID категории",
  "category_category_name": "Категория",
  "brand_brand": "Бренд",
  "update_date": "Дата последнего обновления",
  "load_id": "ID процесса загрузки"
 }
}
```


В случае с таблицей "stock_data", процедура формирования реализована следующим образом:

* Объединение исходных таблиц ("join_info"): таблицей-источником служит "stock"("source"["table"]), наименование полей которой будет изменено ("rename": true) добавлением названия таблицы в качестве префикса. Далее происходит присоединение нескольких таблиц через операцию join. Присоединение таблицы "stores" происходит через inner join, где ключ объединения слева - stock_pos, а справа - stores_pos. Для осуществления присоединения изменения типов какого-либо поля не требуется ("cast": null). Происходит переименование полей ("rename": true).

* Создание представления ("column_names"): из объединения таблиц выбирается набор полей с последующим переименованием.