



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)

---

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.04.01 Информатика и вычислительная техника

МАГИСТЕРСКАЯ ПРОГРАММА 09.04.01/07 Интеллектуальные системы анализа,  
обработки и интерпретации больших данных

## О Т Ч Е Т

по рубежному контролю № 1

**Название:** Разработка современных витрин данных

**Дисциплина:** Искусство аналитической работы с большими данными

Студент

ИУ6-22М  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

С.В. Астахов  
(И.О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Д.В. Березкин  
(И.О. Фамилия)

Москва, 2024

## Содержание

Введение.....	3
1. Проблематика построения и понятие витрин данных.....	4
2. Версионность витрин данных.....	7
3. Apache Airflow.....	12
Заключение.....	17
Список использованных источников.....	18

## **Введение**

В настоящее время в контексте словосочетания “большие данные” наиболее популярны темы, связанные с машинным обучением, нейронными сетями и искусственным интеллектом.

Действительно, искать скрытые закономерности, визуализировать заранее собранные датасеты, применять к ним методы машинного обучения и математической статистики может быть довольно интересно (и, конечно, же полезно для бизнеса).

Однако, зачастую, даже люди, профессионально занимающиеся информационными технологиями, не осознают всей технической сложности, скрывающейся за «расчетом» обычных витрин данных.

В данном реферате освещается основная проблематика расчета витрин данных в современных реалиях, а также использование ПО Apache Airflow для оркестрации расчета витрин данных и ETL-процессов.

## 1. Проблематика построения и понятие витрин данных

Витрина данных (Data Mart) — подмножество (срез) хранилища данных, представляющее собой массив тематической, узконаправленной информации, ориентированной, например, на пользователей одной рабочей группы или подразделения.

Обычно это данные по определенной теме или задаче в компании. Например, витрина с данными о заказчиках для отдела маркетинга может содержать подробные данные по договорам, истории заказов и поставок, оплатах, звонках и адресах доставки[1]. Ничего лишнего, только нужные и актуальные очищенные данные, полученные из других ИС предприятия. Таких витрин даже на одном предприятии может быть множество (рисунок 1).

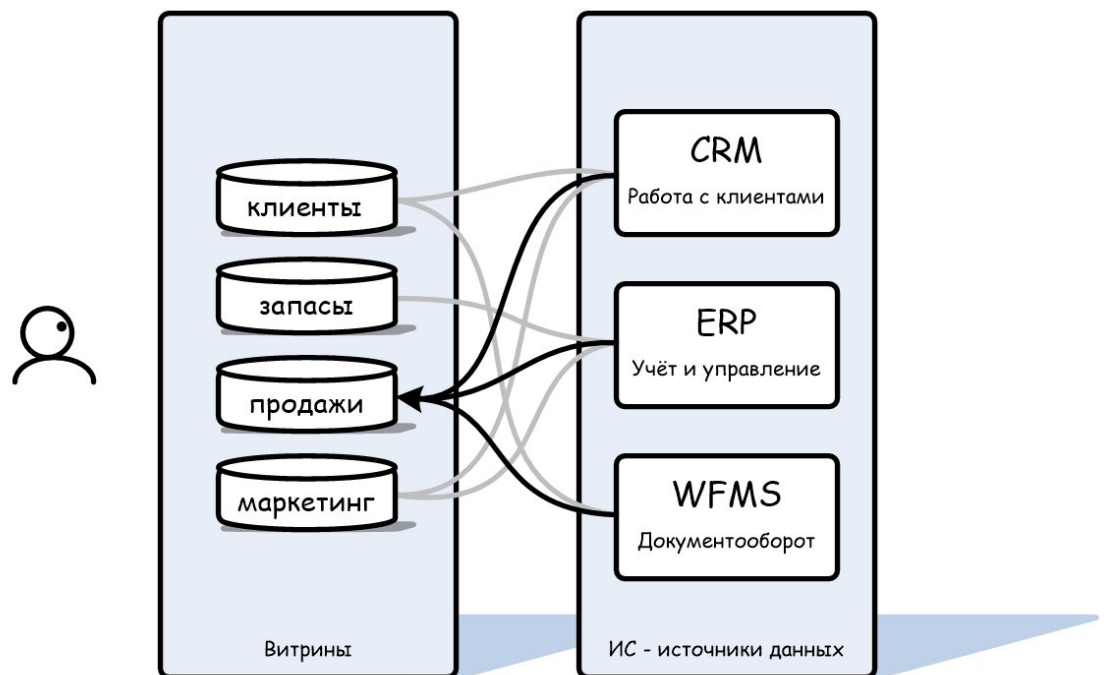


Рисунок 1 — источники и витрины данных

Теперь, попытаемся понять, что из себя представляет витрина данных на более прикладном уровне, для этого пойдем от обратного и перечислим все, что внешне напоминает или ассоциируется с витринами данных, но не является по определению или не удовлетворяет требованиям к эффективности и надежности.

Представления (view). Многим, кто имел дело с СУБД на уровне простых запросов, курсовых и pet-проектов, а затем лишь в виде готовых оберток в приложениях на работе может показаться, что запрос на создание описанных выше сущностей может быть реализован с помощью простых SQL-представлений. Действительно, представление это очень понятная программная реализация данных, однако в реальных условиях она не применима. Основных причин (как минимум) две — невозможность выдержать высокую нагрузку и сложность проектирования. Реальные базы данных, например, такие, как база данных системы управления контентом Drupal (рисунок 2) [2], включают сотни таблиц, а процесс построения витрины на практике захватывает до нескольких десятков таблиц, что затруднительно обработать в рамках одного запроса хотя бы с точки зрения его написания.

Иерархия представлений. Чтобы облегчить составление витрины, можно представить ее как иерархию представлений. Такой подход уже лучше описывает декомпозицию процесса обработки данных в ETL. Однако, в этом случае запрос к витрине все еще будет обрабатываться оптимизатором как единый и будет (вероятно) неоптимален, так как даже современные оптимизаторы имеют довольно ограниченные возможности по сравнению со сложностью ETL-процессов, особенно если по какой-то из таблиц-источников не собрана статистика. Но, даже, если мы построим оптимальный запрос, единовременная загрузка данных из всех источников будет создавать слишком большую нагрузку на оперативную память.

Иерархия материализованных представлений. Итак, мы пришли к тому, что следует «приземлять» результаты вычислений на диск. Но, материализованное представление будет пересчитываться каждый раз, когда изменяются данные хоть в одном источнике. Имея витрину как каскад таких представлений (а сами витрины тоже могут быть зависимы) получим почти постоянные перестроения. Кроме того, объем данных в реальных процессах обычно слишком велик, чтоб пересчитывать всю витрину целиком было в принципе возможно.

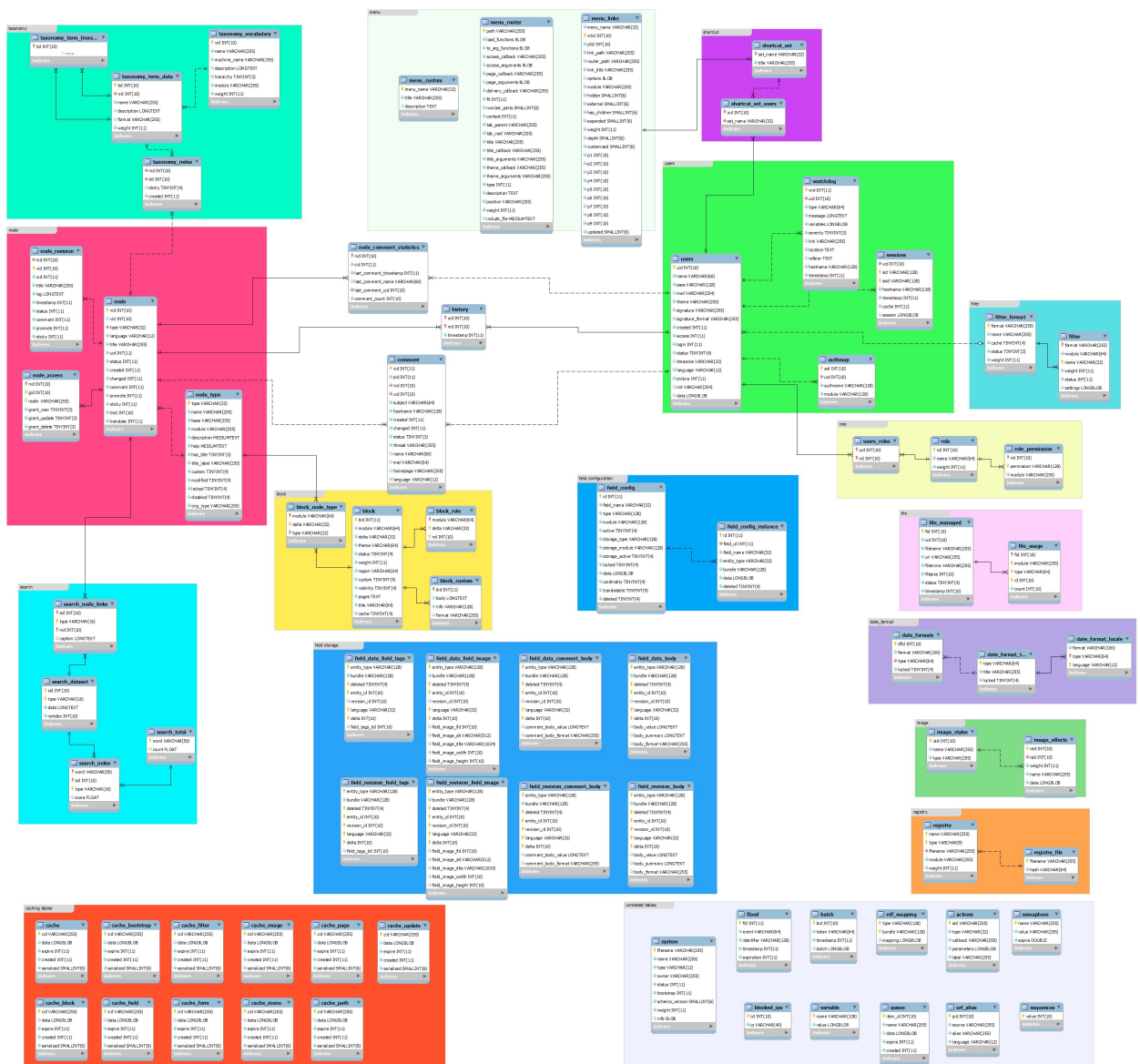


Рисунок 2 — структура БД системы Drupal

Как правило, заказчику не нужно, чтобы витрина постоянно содержала полностью актуальные данные, следовательно мы можем пересчитывать ее по определенному расписанию (в профессиональном сообществе оно называется “регламент”). Кроме того, чтобы минимизировать объем данных в расчетах, как правило, витрина либо строится за один раз лишь на одну отчетную дату (или диапазон дат), например, витрина закупок на определенный день – такие витрины называют “срезными” (они строятся в “разрезе” определенной даты), результирующая таблица наращивается простой дозаписью новых данных. Если бизнес-логика не позволяет организовать такую постройку витрины, например, нам нужна витрина закупок по отделам и типам товаров за все время

наблюдений, то строится т.н. “инкрементальная” витрина. Такая витрина сравнивает версии последних использованных при расчете данных источников с актуальными версиями и производит расчет на основе изменений.

Такой подход (с использованием расписания) позволяет разрабатывать ETL-процессы в виде набора sql-функций или например, spark-скриптов, оркестрируемых каким-либо управляющим механизмом. В теории это могут быть триггеры БД или самостоятельно разрабатываемые скрипты на высокоуровневом языке программирования. Однако, такие инструменты не дают желаемого уровня контроля за успешностью ETL-процессов, их надежностью и т.д. Поэтому, используются либо дорогостоящие BI-инструменты, такие как SAS или Informatica BI. Либо, в сочетании со Spark, СУБД Greenplum и т.п. используется Apache Airflow — открытое программное обеспечение для создания, выполнения, мониторинга и оркестровки потоков операций по обработке данных (см. главу).

## **2. Версионность витрин данных**

Подняв тему инкрементальных витрин, мы затронули такой вопрос как версионность данных. Зачастую, клиенту необходимо, чтобы витрина хранила не только актуальные данные, но и историю их изменений (например, для срезных или т.н. “полносрезных” витрин, которые строятся за один запуск ETL-процесса, но данные в них могут меняться из-за изменения источников).

Для хранения истории изменений существует несколько подходов, разработанных в рамках парадигмы SCD (slowly changing dimensions — редко изменяющиеся измерения, то есть измерения, не ключевые атрибуты которых имеют тенденцию со временем изменяться) [3].

Тип 0. Заключается в том, что данные после первого попадания в таблицу далее никогда не изменяются. Этот метод практически никем не используется, т.к. он не поддерживает версионности. Он нужен лишь как нулевая точка отсчета для методологии SCD.

Тип 1. Это обычная перезапись старых данных новыми. В чистом виде этот метод тоже не содержит версионности и используется лишь там, где история фактически не нужна. Тем не менее, в некоторых СУБД для этого типа возможно добавить ограниченную поддержку версионности средствами самой СУБД (например, Flashback query в Oracle) или отслеживанием изменений через триггеры.

Достоинства:

- не добавляется избыточность;
- очень простая структура.

Недостатки:

- не хранит истории.

Тип 2. Данный метод заключается в создании для каждой версии отдельной записи в таблице с добавлением поля-ключевого атрибута данной версии, например: номер версии, дата изменения или дата начала и конца периода существования версии (рисунок 3).

ID	NAME	POSITION_ID	DEPT	DATE_START	DATE_END
1	Коля	21	2	11.08.2010 10:42:25	01.01.9999
2	Денис	23	3	11.08.2010 10:42:25	01.01.9999
3	Борис	26	2	11.08.2010 10:42:25	01.01.9999
4	Шелдон	22	3	11.08.2010 10:42:25	01.01.9999
5	Пенни	25	2	11.08.2010 10:42:25	01.01.9999

Рисунок 3 — таблица с версионностью SCD-2

В этом примере в качестве даты конца версии по умолчанию стоит '01.01.9999', вместо которой можно было бы указать, скажем, null, но тогда возникла бы проблема с созданием первичного ключа из ID, DATE\_START и DATE\_END, и, кроме того, так упрощается условие выборки для определенной даты (*"where snapshot\_date between DATE\_START and DATE\_END"* вместо *"where snapshot\_date > DATE\_START and (snapshot\_date < DATE\_END or DATE\_END is null)"*).



При такой реализации при увольнении сотрудника можно будет просто изменить дату конца текущей версии на дату увольнения вместо удаления записей о работнике.

Достоинства:

- хранит полную и неограниченную историю версий;
- удобный и простой доступ к данным необходимого периода;

Недостатки:

- провоцирует на избыточность или заведение дополнительных таблиц для хранения изменяемых атрибутов измерения;
- усложняет структуру или добавляет избыточность в случаях, если для аналитики потребуется согласование данных в таблице фактов с конкретными версиями измерения и при этом факт может быть не согласован с текущей для данного факта версией измерения. (Например, у клиента изменились ревизиты или адрес, а нужно провести операцию/доставку по старым значениям);

Тип 3. В самой записи содержатся дополнительные поля для предыдущих значений атрибута. При получении новых данных, старые данные перезаписываются текущими значениями (рисунок 4).

	ID	UPDATE_TIME	LAST_STATE	CURRENT_STATE
1	1	11.08.2010 12:58:48	0	1
2	2	11.08.2010 12:29:16	1	1

Рисунок 4 — таблица с версионностью SCD-3

Достоинства:

- небольшой объем данных;
- простой и быстрый доступ к истории.

Недостатки:

- ограниченная история.

Тип 4. История изменений содержится в отдельной таблице (рисунки 5-6): основная таблица всегда перезаписывается текущими данными

с перенесением старых данных в другую таблицу. Обычно этот тип используют для аудита изменений или создания архивных таблиц (как я уже говорил, в Oracle этот же 4-й тип можно получить из 1-го используя flashback archive).

ID	NAME	POSITION_ID	DEPT
1	Коля	21	2
2	Денис	23	3
3	Борис	26	2
4	Шелдон	22	3
5	Пенни	25	2

Рисунок 5 — таблица с версионностью SCD-4

ID	NAME	POSITION_ID	DEPT	DATE
1	Коля	21	1	11.08.2010 14:12:13
2	Денис	23	2	11.08.2010 14:12:13
3	Борис	26	1	11.08.2010 14:12:13
4	Шелдон	22	2	11.08.2010 14:12:13

Рисунок 6 — таблица с версионностью SCD-4

Достоинства:

- быстрая работа с текущими версиями.

Недостатки:

- разделение единой сущности на разные таблицы.

Гибридный тип/Тип 6(1+2+3). Тип 6 был придуман Ральфом Кимболлом как комбинация вышеназванных методов и предназначен для ситуаций, которые они не учитывают или для большего удобства работы с данными (рисунок 7). Он заключается во внесении дополнительной избыточности: берется за основу тип 2, добавляется суррогатный атрибут для альтернативного обзора версий(тип 3), и перезаписываются одна или все предыдущие версии(тип 1).

VERSION	ID	NAME	POSITION_ID	DEPT	DATE_START	DATE_END	CURRENT
1	1	Коля	21	2	11.08.2010 10:42:25	01.01.9999	1
1	2	Денис	23	3	11.08.2010 10:42:25	01.01.9999	1
1	3	Борис	26	2	11.08.2010 10:42:25	11.08.2010 11:42:25	0
2	3	Борис	26	2	11.08.2010 11:42:26	01.01.9999	1

Рисунок 7 — таблица с версионностью SCD-6

Примечание. На практике, конкретно в витринах данных используются 1, 2 и 6 типы. В хранилище, с которым работаю я, тип 1 реализован на встроенном уровне для любой витрины, т.к. мы работаем с append-only таблицами. Для срезных объектов для каждого РК выбирается строка с максимальным version\_id, для удаленных строк она содержит специальный маркер удаления. Для инкрементальных таблиц поверх этого реализуется тип 3, при этом в приемник обычно складываются записи с effective\_to = 9999, а сбор консистентной истории изменений реализуется на уровне представления на основе комбинации effective\_to, lead(effective\_from) и сортировки по version\_id (таблицы 1-2).

Таблица 1 — исходные данные в приемнике витрины

user_id	zarplata_amt	eff_from	eff_to	version_id	deleted_flg
Vasya	45.000	2001	2999	1	[]
Vasya	60.000	2005	2999	2	[]
Vasya	null	2007	2999	3	[v]

Таблица 2 —результатирующие данные в представлении витрины

user_id	zarplata_amt	eff_from	eff_to
Vasya	45.000	2001	2004
Vasya	60.000	2005	2006

### 3. Apache Airflow

В главе 1 было указано, что одним из удобных способов оркестрации ETL-процессов является использование Apache Airflow. Рассмотрим этот инструмент.

Apache Airflow — открытое программное обеспечение для создания, выполнения, мониторинга и оркестровки потоков операций по обработке данных.

По сути, Apache Airflow является набором из python-библиотеки, веб-сервисов и веб-интерфейса для разработки, исполнения и мониторинга сценария выполнения задач. Зависимости между задачами описываются в виде направленного ациклического графа (DAG — directed acyclic graph), задача может представлять из себя, например, bash-скрипт, python-скрипт, обращение к БД на языке SQL, spark-скрипт или пустой оператор (используется для более удобной визуализации сложных зависимостей). Набор операторов может быть расширен пользователем, Airflow очень универсален.

Рассмотрим пример. Допустим, наш пользователь хочет получать отчет и фото с новых запусках космических ракет, используя API thespacedevs.com. DAG из задач, необходимых для реализации такого сценария представлен на рисунке 8 [4].

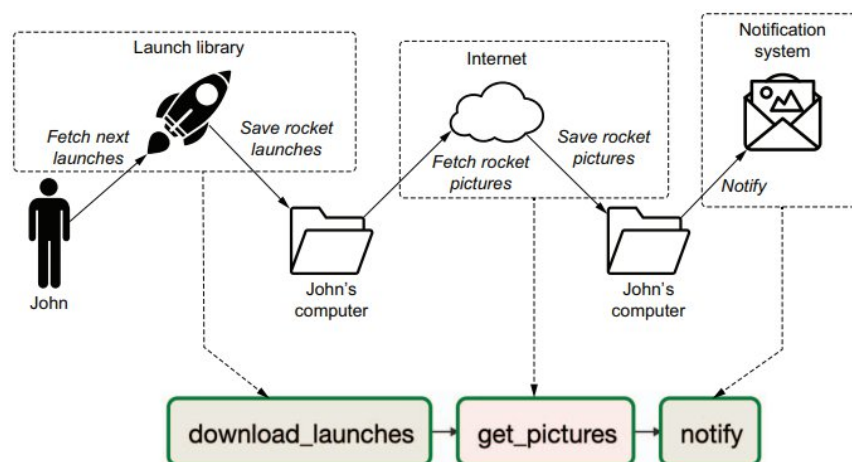


Рисунок 8 — Airflow DAG

Исходный код подобного DAG приведен в листинге 1.

## Листинг 1 — исходный код DAG

```
import json
import pathlib
import requests
import requests.exceptions as req_exec
from datetime import datetime

from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.python import PythonOperator

# отчетная дата
default_args = {
    'start_date': datetime(2021, 7, 10)
}

# функция загрузки картинок
def _get_pictures():
    #ensure directory exists
    pathlib.Path('/tmp/images').mkdir(parents=True, exist_ok=True)
    #download all pictures in launches.json
    with open("/tmp/launches.json") as f:
        launches = json.load(f)
        image_urls = [launch["image" ] for launch in launches["results"]]

    for image_url in image_urls:
        try:
            response = requests.get(image_url)
            image_filename = image_url.split("/") [-1]
            target_file = f"/tmp/images/{image_filename}"
            with open(target_file, "wb") as f:
                f.write(response.content)
            print(f"Downloaded {image_url} to {target_file}")
        except req_exec.MissingSchema:
            print(f"{image_url} appears to be invalid URL.")
        except req_exec.ConnectionError:
            print(f"Could not connect to {image_url}")

with
DAG('download_rocket_launches', schedule_interval=None, default_args=default_args) as dag:

    # шаг 1. Загрузка списка запусков
    download_launches=BashOperator(
        task_id='download_launches',
        bash_command="curl -o /tmp/launches.json -L
'https://11.thespacedevs.com/2.0.0/launch/upcoming/'",
    )

    # шаг 2. Загрузка фото запусков
```

```

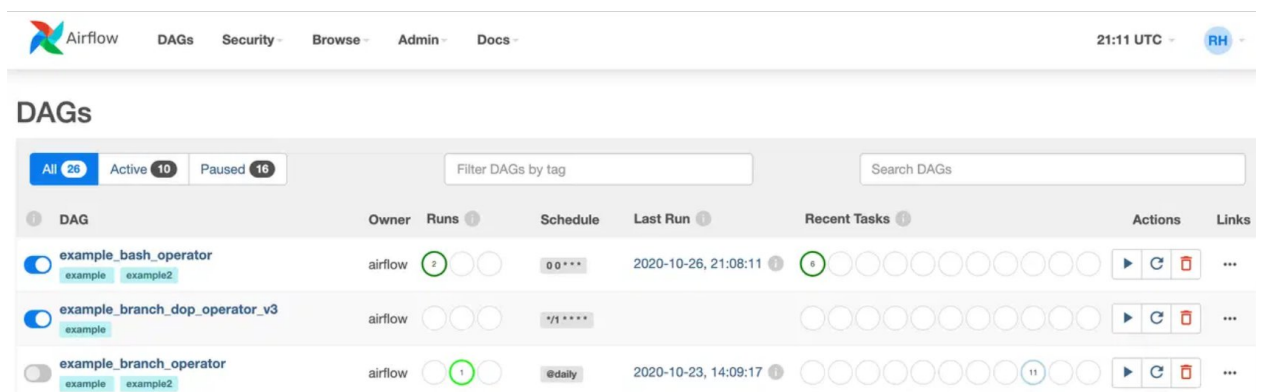
get_pictures=PythonOperator(
    task_id='get_pictures',
    python_callable=_get_pictures
)

# шаг 3. Сообщение в консоль
notify=BashOperator(
    task_id='notify',
    bash_command = 'echo "There are now $(ls /tmp/images/ | wc -l) images."'
)

# ЗАВИСИМОСТИ
download_launches >> get_pictures >> notify

```

Примеры графического интерфейса Airflow приведены на рисунках 9-12 [5].



DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
example_bash_operator	airflow	2	00***	2020-10-26, 21:08:11	6	[Play, Refresh, Stop]	[More]
example_branch_dop_operator_v3	airflow	0	*/****		0	[Play, Refresh, Stop]	[More]
example_branch_operator	airflow	1	@daily	2020-10-23, 14:09:17	11	[Play, Refresh, Stop]	[More]

Рисунок 9 — список DAG

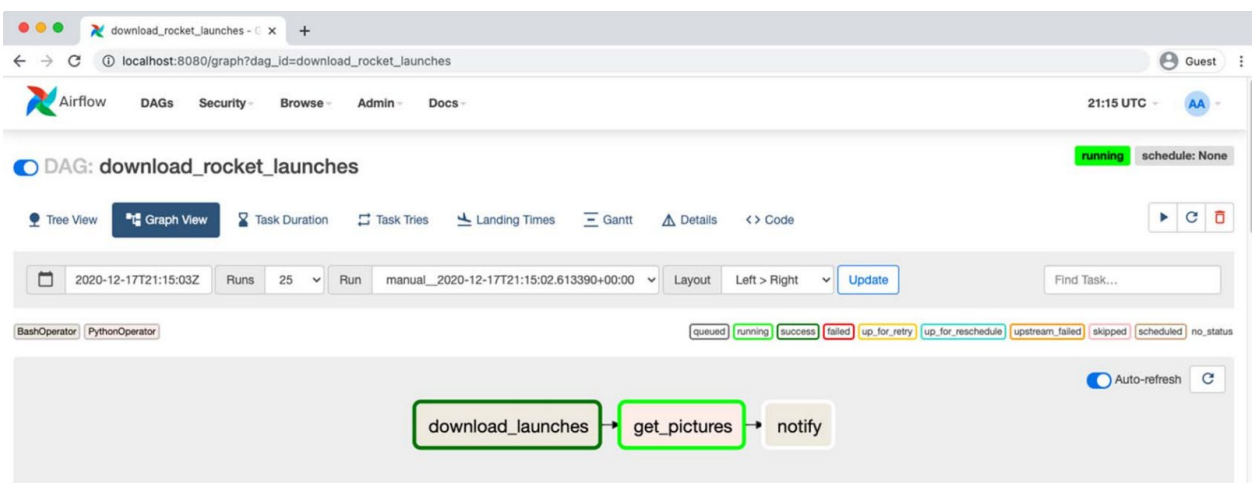


Рисунок 10 — визуализация DAG

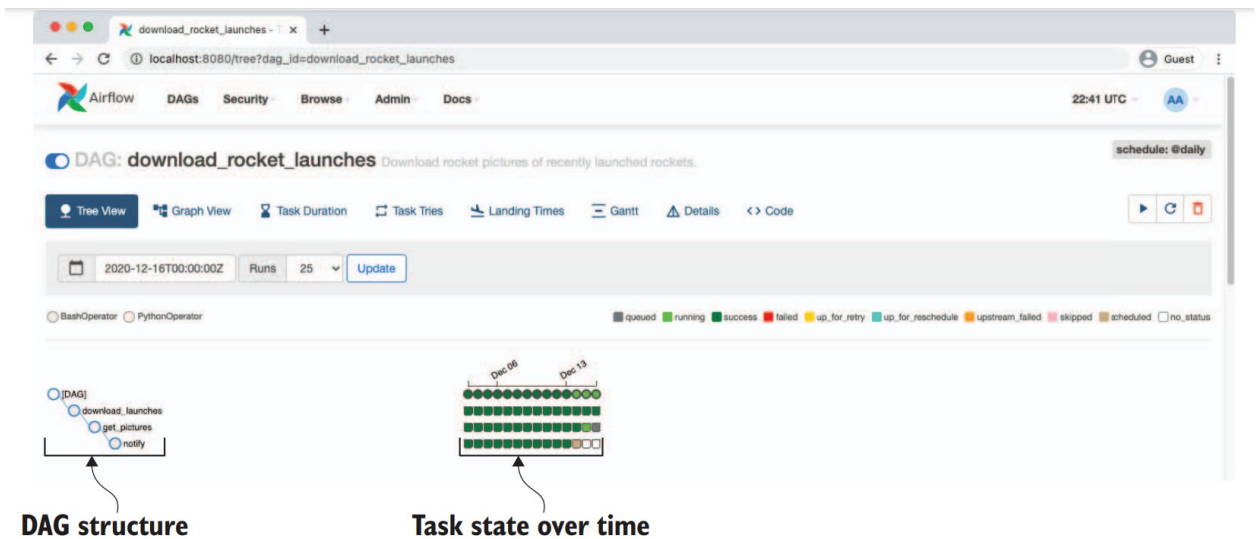


Рисунок 11 — история запусков DAG

```

*** Reading local file: /opt/airflow/logs/download_rocket_launches/notify/2020-12-17T21:15:02.613390+00:00/1.log
[2020-12-17 21:15:30,917] {taskinstance.py:826} INFO - Dependencies all met for <TaskInstance: download_rocket_launches.notify 2020-12-17T21:15:30,917>
[2020-12-17 21:15:30,923] {taskinstance.py:826} INFO - Dependencies all met for <TaskInstance: download_rocket_launches.notify 2020-12-17T21:15:30,923>
[2020-12-17 21:15:30,923] {taskinstance.py:1017} INFO -

[2020-12-17 21:15:30,923] {taskinstance.py:1018} INFO - Starting attempt 1 of 1
[2020-12-17 21:15:30,923] {taskinstance.py:1019} INFO -

[2020-12-17 21:15:30,931] {taskinstance.py:1038} INFO - Executing <Task(BashOperator): notify> on 2020-12-17T21:15:02.613390+00:00
[2020-12-17 21:15:30,933] {standard_task_runner.py:51} INFO - Started process 1483 to run task
[2020-12-17 21:15:30,937] {standard_task_runner.py:75} INFO - Running: ['airflow', 'tasks', 'run', 'download_rocket_launches', 'notify', '2020-12-17 21:15:30,937']
[2020-12-17 21:15:30,938] {standard_task_runner.py:76} INFO - Job 6: Subtask notify
[2020-12-17 21:15:30,969] {logging_mixin.py:103} INFO - Running <TaskInstance: download_rocket_launches.notify 2020-12-17T21:15:02.613390+00:00>
[2020-12-17 21:15:30,993] {taskinstance.py:1230} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=download_rocket_launches
AIRFLOW_CTX_TASK_ID=notify
AIRFLOW_CTX_EXECUTION_DATE=2020-12-17T21:15:02.613390+00:00
AIRFLOW_CTX_DAG_RUN_ID=manual__2020-12-17T21:15:02.613390+00:00
[2020-12-17 21:15:30,994] {bash.py:135} INFO - Tmp dir root location:
/tmp
[2020-12-17 21:15:30,994] {bash.py:158} INFO - Running command: echo "There are now $(ls /tmp/images/ | wc -l) images."
[2020-12-17 21:15:31,002] {bash.py:169} INFO - Output:
[2020-12-17 21:15:31,006] {bash.py:173} INFO - There are now 2 images.
[2020-12-17 21:15:31,006] {bash.py:177} INFO - Command exited with return code 0
[2020-12-17 21:15:31,021] {taskinstance.py:1135} INFO - Marking task as SUCCESS. dag_id=download_rocket_launches, task_id=notify, execution_date=2020-12-17 21:15:30,923
[2020-12-17 21:15:31,037] {taskinstance.py:1195} INFO - 0 downstream tasks scheduled from follow-on schedule check
[2020-12-17 21:15:31,070] {local_task_job.py:118} INFO - Task exited with return code 0
  
```

Рисунок 12 — логи DAG

Таким образом, на данном примере мы познакомились с основными сущностями Apache Airflow:

1. DAG'и (DAGs) — ключевая сущность Airflow. Это скрипты на Python, которые описывают логику выполнения задач: какие должны быть выполнены, в каком порядке и как часто.
2. Задача (Task) — описывает, что делать. Например, выборку данных, анализ, запуск других систем. Каждая задача — это экземпляр оператора с определенными параметрами. Допустим, есть DAG для загрузки данных из базы. Можно создать задачу для выполнения оператора, который отправит SQL-запрос для загрузки данных. Она

будет содержать информацию о том, какой SQL-запрос нужно выполнить, когда и в каком контексте.

3. Оператор (Operator) — класс Python, который определяет, что нужно сделать в рамках задачи. Есть операторы для выполнения скриптов Bash, кода Python, SQL-запросов. Например, чтобы выполнить скрипт Python для анализа данных, используют PythonOperator.



## **Заключение**

Несмотря на растущую популярность таких направлений обработки больших данных, как машинное обучение и нейронные сети, в настоящее время остается актуальной проблема проектирования высоконагруженных хранилищ данных, ETL-процессов и витрин данных.

В данном реферате были рассмотрены:

- определение витрины данных;
- актуальные технические проблемы построения высоконагруженных витрин данных;
- организация версионного хранения данных;
- использование Apache Airflow в построении ETL-процессов.

### Список использованных источников

1. Простор для данных // habr : сайт. – URL: <https://habr.com/ru/articles/650237/> (дата обращения: 15.03.2024)
2. Drupal : сайт. – URL: <https://www.drupal.org/node/2360815> (дата обращения: 18.03.2024)
3. Версионность и история данных // habr : сайт. – URL: <https://habr.com/ru/articles/101544/> (дата обращения: 21.03.2024)
4. Harenslak, B. Data Pipelines with Apache Airflow / B. Harenslak, J. de Ruitер. – Shelter Island : Manning Publications Co., 2021. – 482 с. – ISBN 9781617296901.
5. Как Apache Airflow помогает дирижировать данными компаний // Яндекс.Практикум : сайт. – URL: <https://practicum.yandex.ru/blog/apache-airflow/> (дата обращения: 26.03.2024)