Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерна інженерія та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи та засоби автоматизації процесів обробки великих даних
(тема)

Виконав:
студент II курсу, групи СПм-22-4
Залеський В.Д.
(прізвище, ініціали)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: доц. Федорченко В.М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерна інженерія та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Залеському Валентину Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи та засоби автоматизації процесів обробки великих даних

затверджена наказом по університету від " 01 " квітня 2024 р. № 257Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 червня 2024 р.

3. Вхідні дані до роботи
база даних
сховище даних
ELT/ETL процеси
інструменти оркестрування
Spark
DBT

4. Перелік питань, що потрібно опрацювати у роботі
Методи автоматизації процесів обробки великих даних
Експериментальні дослідження

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 14 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання на аналіз джерел за темою	01.04.2024-24.04.2024	
	роботи		
2	Аналіз існуючих методів	25.04.2024-06.05.2024	
3	Розробка нового або модифікація існуючого	01.05.2024-11.05.2024	
	методу		
4	Програмна реалізація розробленого або	01.05.2024-24.05.2024	
	модифікованого методу		
5	Аналіз отриманих результатів	24.05.2024-01.06.2024	
6	Оформлення пояснювальної записки та	01.06.2024-12.06.2024	
	Документів до неї		

Дата видачі завдання 01 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Федорченко В.М.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 68 с., 44 рис., 2 табл., 1 дод., 26 джерел.

ОРКЕСТРУВАННЯ ДАНИХ, КОНВЕЄР ОБРОБКИ ДАНИХ, СХОВИЩЕ ДАНИХ, ETL, ELT, DAG, DBT, SPARK.

Метою кваліфікаційної роботи є оптимізація конвеєрів обробки великих даних.

У ході виконання кваліфікаційної роботи були розглянуті сучасні інструменти оркестрування даних і побудудован автоматизований процесс обробки даних для експериментальної аналітичної системи з використанням деяких з розглянутих інструментів.

Об’єктом дослідження є інженерія великих даних.

Предметом дослідження є оркестрування даних.

Завдання:

- визначення методів підвищення продуктивності обробки великих даних;

- визначення засобів покращення оркестрування великих даних;

- розробка експериментальної аналітичної системи з оптимізованими методами обробки і засобами оркестрування даних.

ABSTRACT

Master’s thesis: 68 pages, 44 figures, 2 tables, 1 appendices, 26 sources.

DATA ORCHESTRATION, DATA PIPELINE, DATA WAREHOUSE, ETL, ELT, DAG, DBT, SPARK

The major goal of this thesis was an optimization of automated data pipelines.

In order to achieve this goal, the modern data orchestration tools were both overview and applied on built of experimental analytical system.

The object of research is data engineering.

The subject of research is data orchestration.

Research goals:

- big data processing performance improvement;
- data orchestration improvement;
- design of experimental data analytical system with optimized both data processing and data orchestration.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ETL – Extract, Transform, Load

ELT – Extract, Load, Transform

DAG – directed acyclic graph

CI/CD – continuous integration and continuous delivery

OLTP – Online Transaction Processing

ERD - Entity-Relationship Diagram

КІ – користувацький інтерфейс

ПЗ - програмне забезпечення

ВСТУП

В даний час спостерігається значне зростання обсягів інформації, що надходить з найбільш різних джерел. Це можуть бути мобільні пристрої, програмні логи, інформація про погоду, сейсмічні дані, інформація про проведені фінансові операції, активності користувачів соціальних мереж і багато іншого. Інформація представлена широким спектром типів даних, починаючи зі звичних баз даних з чіткою структурою, закінчуючи зображеннями і текстами на природній мові. Обсяг, накопиченої інформації, настільки великий, а типи даних настільки різноманітні, що її обробка стандартними методами не є можливим. Термін Big Data служить для позначення подібних даних, обробка яких стандартними методами за розумний час неможлива.

Зі збільшенням кількості даних збільшується потреба в управлінні, синхронізації розкладів та вирішеня проблем обробки. Виникає необхідність зламати бар'єри між джерелами даних та сховищами, щоб по-справжньому використовувати всю інформацію, яка збираються. Оркестрування даних дозволяє автоматизувати та оптимізувати дані, перетворюючи їх на оперативні активи, щоб цінну інформацію можна було використовувати для прийняття бізнес-рішень у режимі реального часу. За деякими оцінками, 55% роботи, пов'язаної з аналізом даних, зводиться до збирання та підготовки даних, що означає, що оркестрування даних може скоротити велику кількість часу на обробку та планування.

З неухильним зростанням кількісті інструментів для обробки та оркестрування даних виникає необхідність їх систематизації для кращого розуміння коли і який інструмент підходить під вирішення конкретних задач в конкретному випадку.

Метою кваліфікаційної роботи є оптимізація конвеєрів обробки великих даних.

Об’єкт дослідження є інженерія великих даних.

Предметом дослідження є оркестрування даних.

Завдання:

- підвищення продуктивності обробки великих даних;

- покращення оркестрування великих даних;

- розробка експериментальної системи з оптимізованими засобами обробки і оркестрування даних.

1 АНАЛІЗ МЕТОДОЛОГІЙ ОБРОБКИ ВЕЛИКИХ ДАНИХ

1.1 Огляд еволюції методів обробки великих даних

Великі дані, і зокрема екосистема Hadoop, як засіб зберігання та обробки петабайтів даних, народилися трохи більше 15 років. Навколо нього сформувалася широка та динамічна екосистема із сотнями проектів, і він досі використовується багатьма великими компаніями.

Еволюцію екосистеми Hadoop, можно умовно поділити на окремі періоди.

На початку 2000-х років компанія Google зіткнулася з двома основними проблемами під час спроби індексувати величезний обсяг інформації в Інтернеті:

- зберіганя сотні терабайт даних на тисячах дисків на більш ніж тисячі машин без простоїв та втрат даних;

- ефективне розподільне обчислення та обробка всіх цих даних.

Як результат вирішення цих проблем у період з 2003 по 2006 рік Google було опублікувано три дослідницькі статті, які пояснювали нову архітектуру обробки даних:

- The Google File System (2003) [2] – описувала розподілену файлову систему;

- MapReduce: Simplified Data Processing on Large Clusters (2004) [3]– детально розглядала метод розподіленої обробки даних;

- Bigtable: A Distributed Storage System for Structured Data (2006) [4] – представляла нову систему зберігання даних.

Саме ці роботи слугували базою для створення Hadoop. У 2005 році у рамках Yahoo! був створений проект Hadoop, що включав Hadoop Distributed File System (HDFS) та реалізацію MapReduce. На відміну від Google, Yahoo! було вирішено зробити Hadoop відкритим проектом, що значно сприяло його

популяризації та розвитку.

Оскільки написання джоб обробки даних на MapReduce виявилось доволі складною задачею, то як результат виникло ряд інструментів покликаних для спрощення цього процесу, таких як Apache Pig, Apache Hive і тд., суть яких була у перетворенні SQL або власної мови (як Apache Pig з мовою Pig Latin) у кроки MapReduce.

Іншою складною проблемою використання Hadoop були великі зусилля, необхідні для налаштування, моніторингу та обслуговування кластера Hadoop.

Першими хто спробував вирішити цю проблему були Amazon. Вони запустили AWS хмарні сервіси, серед яких був Elastic MapReduce сервіс, який дозволяє розгортати та запускати завдання MapReduce, без проблем з керуванням кластером Hadoop.

У 2008 році був заснований перший постачальник Hadoop – Clouder. Cloudera запропонувала готовий дистрибутив Hadoop під назвою CDH разом із інтерфейсом моніторингу кластера Cloudera Manager, який полегшив установку та підтримку кластера Hadoop разом із супутнім програмним забезпеченням, таким як Hive і HBase. Згодом з цією ж метою були засновані Hortonworks і MapR.

Наступною проблемою була необхідність у інструменті, який добре справлявся би з інтерактивною аналітикою та BI, адже хоч Hive і був чудовим інструментом SQL він погано підходив для цих задач.

І знову Google опосередковано вплинуло на світ великих даних, опублікувавши в 2010 році четверту дослідницьку статтю під назвою “Dremel: Interactive Analysis of Web-Scale Datasets”[5].

У цьому документі описуються дві основні інновації: розподілена інтерактивна архітектура запитів, яка надихне більшість інтерактивного SQL, і формат зберігання, орієнтований на стовпці, який надихне кілька нових форматів зберігання даних, таких як Apache Parquet, розроблений спільно Cloudera і Twitter, а також Apache ORC, розроблені спільно Hortonworks і

Facebook.

Натхненний Dremel, намагаючись вирішити проблему високої затримки Hive, Cloudera у 2012 році запустила новий механізм SQL з відкритим кодом для інтерактивних запитів під назвою Apache Impala.

Подібним чином MapR запустив власний інтерактивний механізм SQL з відкритим вихідним кодом під назвою Apache Drill, тоді як у Hortonworks вирішили, що краще зробити Hive швидшим, ніж створювати новий механізм з нуля, і запустили Apache Tez і адаптували Hive для виконання на Tez замість MapReduce.

Під час оновлення Hadoop 2.0 було представлено ключовий компонент, YARN (Yet Another Resource Manager) як офіційного менеджера ресурсів, роль, яку раніше виконував MapReduce.

В цей же час проект Apache Spark почав набирати популярності і стало зрозуміло, що Spark стане заміною MapReduce, оскільки він мав кращі можливості, простіший синтаксис і в багатьох випадках був набагато швидшим за MapReduce, особливо завдяки його здатності кешувати дані в оперативній пам’яті.

Він також мав чудову взаємодію з Hive, оскільки SparkSQL базувався на синтаксисі Hive, що полегшувало міграцію з Hive на SparkSQL. Він також отримав велику популярність у світі машинного навчання, оскільки попередні спроби написати алгоритми машинного навчання на MapReduce, як-от Apache Mahout, були швидко перевершені реалізаціями Spark.

У 2013 році творцями Spark було заснувано Databricks. Перевага Databricks полягала у тому що вони запропонували прості та багатофункціональні API багатьма мовами (Java, Scala, Python, R, SQL, і навіть .NET) і власні конектори для багатьох джерел даних і форматів (csv, json, parquet, jdbc, avro тощо) для обробки даних. Слід зазаначити, що у Databricks було обрано іншу ринкову стратегію від своїх попередників: замість локального розгортання Spark (який був доданим Cloudera та Hortonworks до своєї власної платформи), Databricks запропонував

розгортання в хмарній платформі побудованій зпочатку у AWS а згодом і в Azure та GCP.

У 2011 році з’явилися проекти для роботи з подіями в реальному часі: Apache Kafka – розподілена черга повідомлень, створена LinkedIn, і Apache Storm, механізм розподілених обчислень реального часу, створений Twitter.

До 2014 року кількість проектів в екосистемі Hadoop значно зросла. З’явилися нові інструменти для обробки даних, спрямовані на надання уніфікованого інтерфейсу для обробки як пакетної, так і потокової обробки на основі різних розподілених обчислень: Apache Beam, Apache Flink або DataFlow від Google.

Приблизно у 2015 році на ринку з’являються хороші інструменти оркестрування з відкритим кодом: Apache Airflow від Airbnb, Luigi від Spotify – які швидко отримали широке поширення в інших компаніях. Зокрема, Airflow тепер доступний у режимі SaaS на Google Cloud Platform і Amazon Web Services.

Що стосується SQL, з’явилося кілька інших розподілених сховищ даних, які мали на меті забезпечити швидші можливості інтерактивного запиту, аніж Apache Hive. У 2013 році Facebook зробив відкритим вихідний код, який згодом був перейменований Amazon на сервіс під назвою Athena. Цей період також був ознаменований розвитком нових пропреєтарних розподілених сховищ: BigQuery від Google у 2011 році, Redshift від Amazon у 2012 році та Snowflake, заснований у 2012 році.

У міру того як розвивалась індустрія обробки даних спостерігалось впровадження інструментів керування метаданими та каталогів таких як наприклад Apache Atlas, започаткований Hortonworks у 2015 році.

Наразі частиною екосистеми Hadoop є понад 150 проектів [6].

Спрощене зображення екосистеми Hadoop зображене на рисунку 1.1.

Рисунок 1.1 – Спрощене зображення екосистеми Hadoop

Наступні роки можно охарактеризувати наступними тендеціями.

Першою тенденцією була масова міграція інфраструктур даних у хмару, де HDFS було замінено хмарними сховищами, такими як Amazon S3, Google Storage або Azure Blob Storage.

Оскільки Hive і Spark спочатку створювалися для HDFS, то деякі характеристики продуктивності, гарантовані HDFS, були втрачені під час міграції до хмарних сховищ як S3, що спричинило появу нових форматів зберігання даних, в яких цю пробему було виправлено.

До нових форматів зберігання даних з відкритим кодом можно віднесстри: Apache Hudi розроблений Uber у 2016 році, Apache Iceberg запущений Netflix у 2017 році, і Delta Lake, який був створений Databricks у 2019 році.

Другою тенденцією стала контейнеризація з використанням Docker і Kubernetes. Ці інструменти дозволили компаніям розгортати нові типи розподілених архітектур, більш стабільні та масштабовані, для багатьох випадків використання, включаючи обробку даних в реальному. Hadoop

знадобився деякий час, щоб наздогнати Docker, оскільки підтримка запуску контейнерів Docker у Hadoop надійшла з версією 3.0 у 2018 році.

Третьою тенденцією було зростання повністю керованих масових паралельних сховищ даних SQL для аналітики, зростання «Modern Data Stack» і dbt, який вперше був відкритий у 2016 році.

Четвертою тенденцією, яка вплинула на Hadoop, стала поява глибокого навчання. Оскільки Hadoop і Machine Learning були дуже різними напрямками, і їх було важко поєднати разом - викликало потребу в нових підходах до великих даних і довело, що Hadoop не є правильним інструментом для всього.

Складність полягала у тому що для работи глибоким навчанням, потрібні були дві речі, які Hadoop на той час не міг забезпечити. Були потрібні графічні процесори, яких зазвичай не було у вузлах кластера Hadoop, і потрібно було встановити останню версію своїх бібліотек глибокого навчання, таких як Tensorflow або Keras, що було важко зробити на цілому кластері, особливо коли кілька користувачів запитували різні версії однієї бібліотеки.

Ця конкретна проблема вирішена у 2017 році, коли Cloudera випустила Data Science Workbench, який базувалося не на Hadoop чи YARN, а на контейнеризації за допомогою Docker і Kubernetes і дозволяляє розгортати моделі у власному середовищі як контейнерну програму, не ризикуючи проблемами безпеки чи стабільності.

Щодо сьогодення то можно виділити наступні тенденції.

Розгортання Hadoop у хмарі здебільшого замінено програмами Apache Spark або Apache Beam (здебільшого на GCP), на користь Databricks, Amazon Elastic Map Reduce (EMR), Google Dataproc/Dataflow або Azure Synapse.

Крім того, багато компаній націлюються на підхід “Modern Data Stack”, побудований на аналітичному сховищі SQL таких як: BigQuery, Databricks-SQL, Athena або Snowflake, що обробляють дані на стороні сховища та орнанізовані за допомогою dbt, який не потребує інструментів

розподіленого обчислення, таких як Spark.

Розповсюджується концепції Lakehouse – платформа, яка поєднує в собі переваги озера даних і сховища даних. Що дозволяє спеціалістам з обробки даних і користувачам BI працювати на одній платформі даних, що полегшує керування, безпеку та обмін знаннями. Databricks були першими, хто ввів цей термін і позиціонувався в цій пропозиції продуктів завдяки універсальності Spark. За ними послідували Snowflake зі Snowpark, Azure Synapse і нещодавно Google з BigLake. Що стосується відкритого коду, Dremio пропонує архітектуру Lakehouse з 2017 року.

У міру того як індустрія обробки даних продовжувала розвиватися, набувають популярності ряд нових технологіях оркестрування, як-от Prefect, Dagster, Mage, Kestra чиї репозиторії з відкритим кодом, які приходять на зміну до цього безальтернативного Airflow.

1.2 Методологія обробки великих даних поза сховищем даних

Під обробкою даних поза сховищем даних мається на увазі використання інструменетів розподіленої обробки великих даних, які імпліментують трансформаціїї в рамках ETL [9] процесів (рисунок 1.2) на базі обчислювальних потужностей на окремовідведених для цього кластерах. В цій царині безумовним лідером серед інструментів є Apache Spark [10], тож нижче буде наведен детальний аналіз цього інструменту.

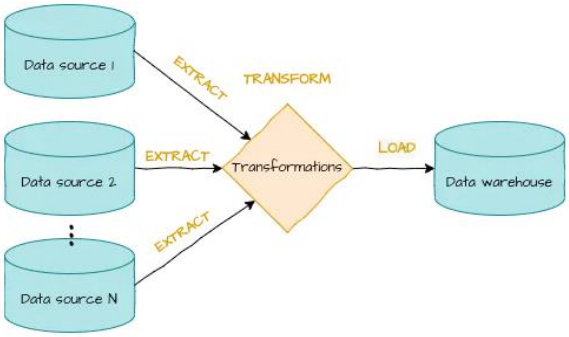


Рисунок 1.2 – Діаграма ETL процесу

Apache Spark [11] – це уніфікований механізм, призначений для розподіленої обробки великих даних, як у локальних дата-центрах, так і в хмарі. Spark забезпечує зберігання в пам'яті для проміжних обчислень, що робить його набагато швидшим за MapReduce.

Він включає в себе бібліотеки (рисунок 1.3) зі складовими API для машинного навчання (MLlib), SQL для інтерактивних запитів (Spark SQL), потокової обробки (Structured Streaming) для (Structured Streaming) для взаємодії з даними в реальному часі та обробки графів (GraphX).

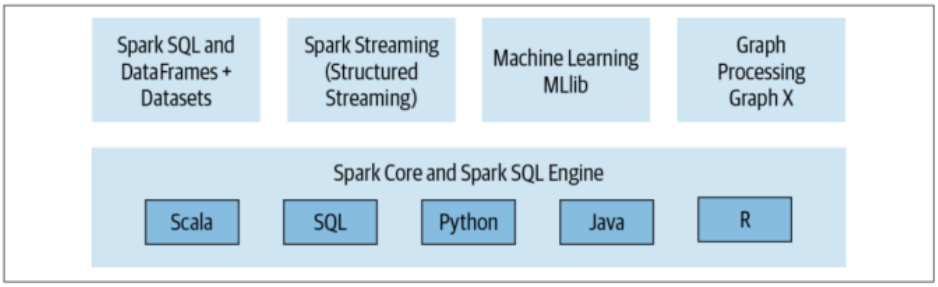


Рисунок 1.3 – Бібліотеки Spark

Філософія розробки Spark базується на чотирьох ключових характеристиках:

- швидкість;
- простота використання;
- модульність;
- розширюваність.

На рисунку1.4 наведена архітектура компонентів Spark.

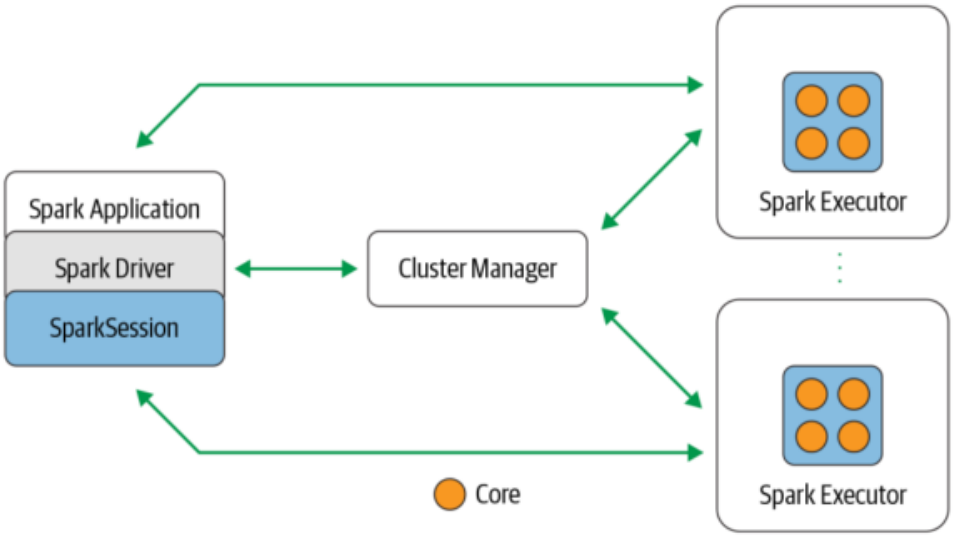


Рисунок 1.4 – Компоненти Spark

Роль драйвера Spark в застосунку Spark полягає в тому, що він відповідає за створення SparkSession. Драйвер Spark виконує кілька функцій:

- взаємодіє з диспетчером кластера;

- запитує ресурси (центральний процесор, пам'ять тощо) у диспетчера кластера для виконавців Spark (JVM);

- трансформує всі операції Spark в обчислення DAG, планує їх і розподіляє їх виконання як завдання між виконавцями Spark;

- після виділення ресурсів він напряму зв'язується з виконавцями.

SparkSession забезпечує єдину уніфіковану точку входу до всієї функціональності Spark. SparkSession дозволяє створювати параметри виконання JVM, визначати DataFrames і Datasets, читати різноманітні джерела даних, отримувати доступ до метаданих каталогу і виконувати Spark SQL запити.

Менеджер кластера відповідає за управління та розподіл ресурсів для вузлів кластеру, на яких працює застосунок Spark. Наразі Spark підтримує чотири кластерні менеджери: вбудований автономний кластерний менеджер, Apache Hadoop YARN, Apache Mesos та Kubernetes.

На кожному робочому вузлі кластера працює як мінімум один виконавець. Виконавці взаємодіють з програмою-драйвером і відповідають за виконання окремих завдань.

При виконанні застосунку Spark драйвер перетворює програму на одну або декілька завданнь(job). Потім він розбиває кожнt завдання на DAG (Directed Acyclic Graph – спрямований ациклічний граф). Це, по суті, план виконання Spark, де кожен вузол в DAG може бути одним або декількома етапами Spark (stages).

Етапи створюються всередині DAG на основі того, які операції можна виконувати послідовно або паралельно. Не всі операції Spark можуть відбуватися на одному етапі, тому їх можна розділити на кілька етапів.

Кожен етап складається із задач Spark (task), які потім розподіляються між кожним виконавцем Spark, як наведено на рисунку 1.5.

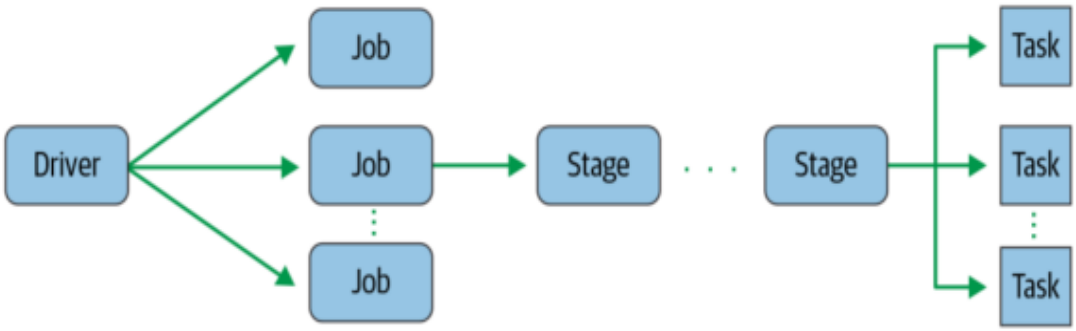


Рисунок 1.5 – Модель виконання застосунку Spark

Слід зазначити що дані при обробці фізично розподілені сховищем поміж декількох розділів (партицій), які зберігаються або в HDFS, або в хмарному сховищі. Хоча фізично дані розділені на кластері, Spark працює з кожним розіділом як з логічною одиницею високого рівня – DataFrame. За можливості, кожному виконавцю Spark призначається завдання обробити розділ, який знаходиться найближче до нього в мережі, дотримуючись локальності даних.

Розбиття даних на розділи дозволяє виконавцям Spark обробляти лише дані, розташовані поруч із ними, мінімізуючи використання мережевого трафіку. Тобто, кожному ядру виконавця призначається своя партиція даних на обробку, як наведено на рисунку 1.6.

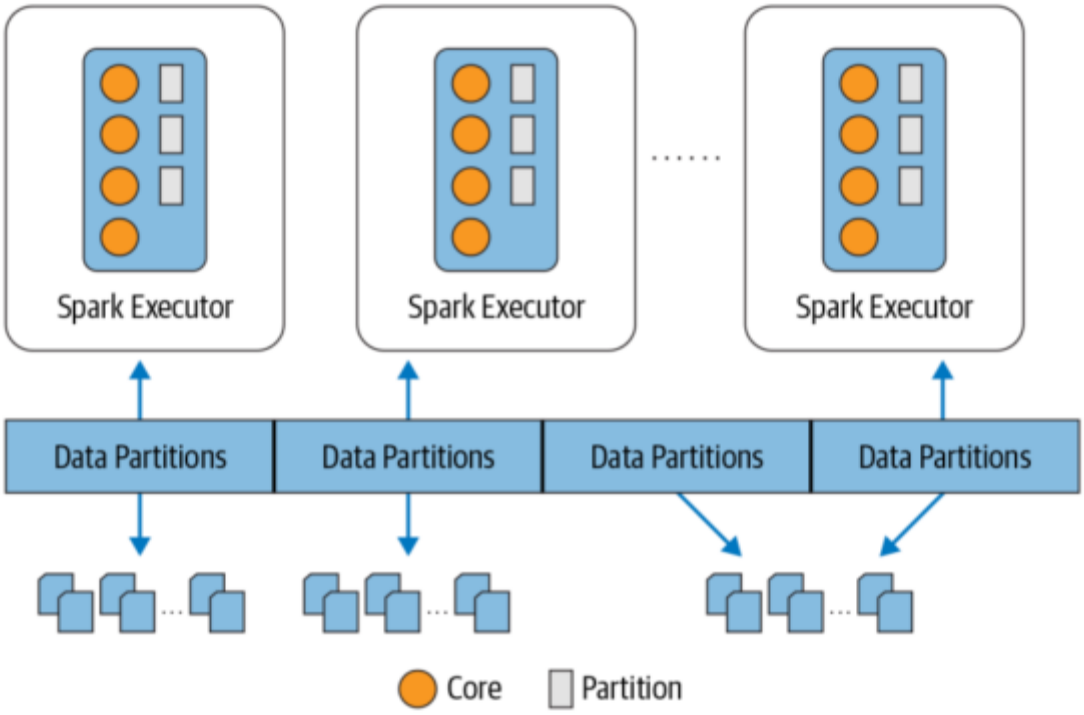


Рисунок 1.6 – Розбиття даних на розділи під час обробки у Spark

1.3 Методологія обробки великих даних безпосередньо у сховищі даних

Обробка даних на стороні сховища даних , тобто передбачає обробку даних безпосередньо у сховищі даних за рахунок його обчислювальних потужностей.

Тож виходячи з цієї необхідності був створений аналітичний інструмент з відкритим кодом – DBT (data build tool) [17]. Основна мета DBT - допомогти перетворити дані сховища даних у простий спосіб – за рахунок виконання SQL-запитів. Слід зазначити що DBT відповідає діяльності на етапі трансформації у ELT процесі (рисунок 1.7), коли дані спочатку витягуються з різних джерел і завантажуються в сховище даних, а потім до сирих даних застосовуються трансформації в самому сховищі даних.

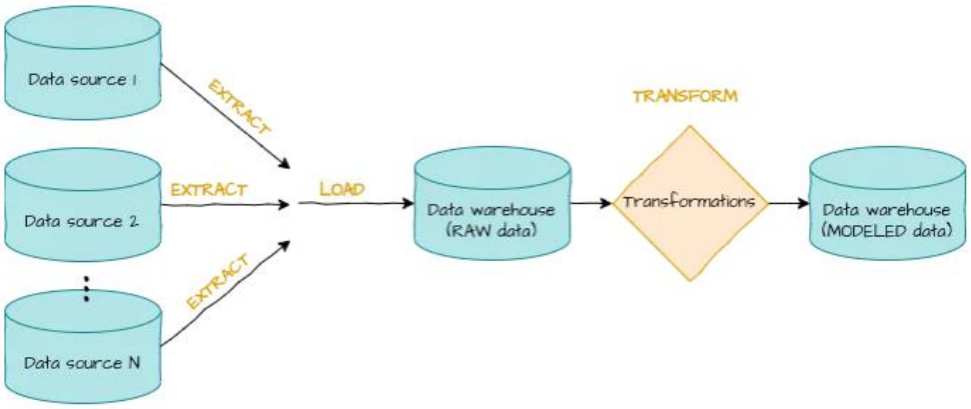


Рисунок 1.7 – Діаграма ELT процесу

Зокрема цього DBT також забезбечує контроль версій, створення документації, тести та автоматизоване розгортання та сумісний з BI інструментами (див рис.1.8).

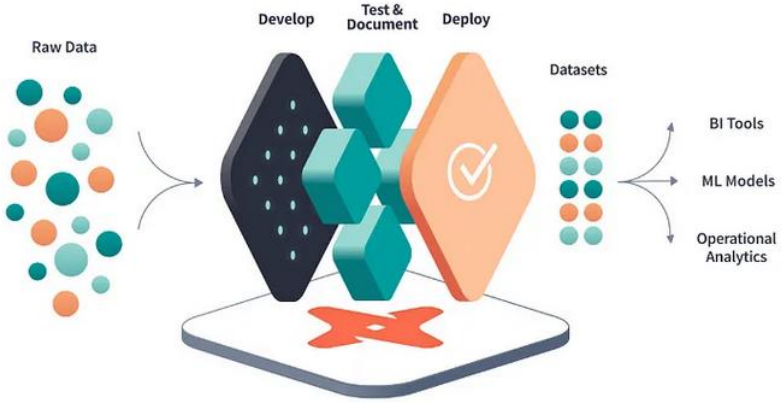


Рисунок 1.8 – Діаграма процесу розробки у DBT [16]

DBT пропонує два способи роботи: у хмарі DBT Cloud та локально за допомогою CLI (Command Line Interface – інтерфейс командного рядка). До особливостей архітектурних рішень dbt можно віднести наступні:

- централізоване керування: DBT в основному керується через інтерфейс командного рядка, що дозволяє користувачам запускати команди для перетворень , тестування та генерації документації;
- динамічна генерація SQL: поєднуючи SQL з механізмом шаблонів

Jinja2 (циклів, умовної логіки і макросів), DBT дозволяє динамічно генерувати SQL;

- DBT будує DAG залежностей моделей, що визначають порядок виконання перетворень під час запуску;

- DBT використовує адаптери для підключення та взаємодії з різними платформами даних, такими як Snowflake, BigQuery та Redshift, що перекладають загальний SQL DBT у специфічний SQL для конкретної бази даних;

- DBT підтримує як вбудовані тести (наприклад, unique або not_null), так і кастомні тести, визначені в SQL, що забезпечують якість даних і відповідність бізнес-правилам;

- інтеграція з Gitom: проекти DBT зазвичай зберігаються в репозиторіях Git, що дозволяє співпрацювати, керувати версіями та розгалуженнями;

- DBT автоматично генерує веб-портал документації, який візуалізує метадані моделі, граф походження та описи;

- DBT дозволяє розширення, тобто створювати різноманітні плагіни, що додають функціональності та сумісності з іншими інструментами.

На відміну від інструментів ETL, які можуть мати власні обчислювальні механізми, DBT компілює і виконує SQL безпосередньо у цільовому сховищі даних, використовуючи його обчислювальну потужність для перетворень.

Найважливіші команди, які можна використовувати в обох випадках, наведені нижче:

- dbt deps: завантажує залежності для проекту;
- dbt run: запускає моделі у проекті;
- dbt test: виконує тести, визначені у проекті;
- dbt docs generate: генерує документацію про проект.

На рисунку 1.9 показано в якій послідовності вони зазвичай виконуються

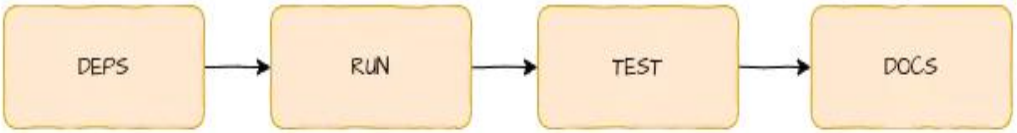


Рисунок 1.9 – Послідовність виконання команд при розробці у DBT

Спрощена структура DBT проекта наведена на рисунку1.10 і включає в себе ряд компонентів, які розглянуті нижче.

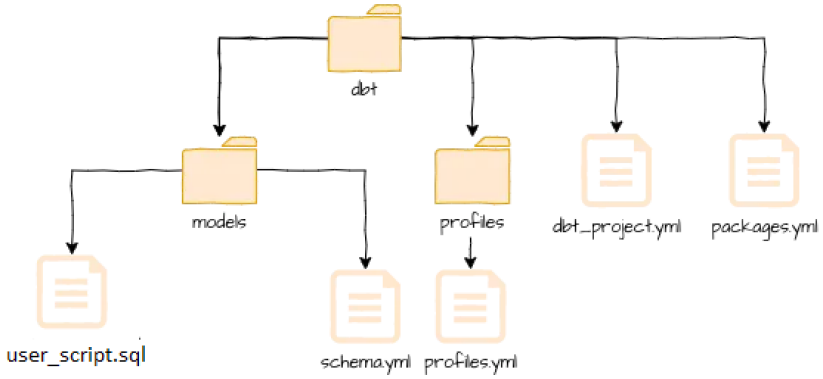


Рисунок 1.10 – Спрощена структура DBT проекту

Модель – це SQL-файл, який містить інструкцію select, що перетворює вихідні дані. У DBT назва моделі вказує на назву майбутньої таблиці.

Схема – це YAML-файл з усією інформацією про джерела та моделі. Назви джерел повинні збігатися з назвами файлів моделей у коді Jinja.

При використанні CLI необхідно визначити деталі підключення до нашої бази даних. Файл profiles.yml містить всю цю інформацію, і може зберігати стільки профілів, скільки потрібно, зазвичай по одному профілю для кожного сховища.

Файл dbt_project.yml є ядром будь-якого проекту DBT, він містить важливу інформацію, яка вказує DBT, як працювати.

Під пакетами мається на увазі бібліотеки, які є ніщо інше як інші проекти з моделями та макросами, які вирішують певну проблему.

2 ОГЛЯД ЗАСОБІВ ОРКЕСТРУВАННЯ ОБРОБКИ ВЕЛИКИХ ДАНИХ

2.1 Оркестрування обробки великих даних

Оркестрування даних [12] відіграє головну роль у автоматизації, кординації і управлінні робочими процесами при побудові конвеєрів обробки даних.

Центральну роль в оркеструванні даних відіграє ETL процес , він розшифровується як “вилучення (extract), перетворення (transform), завантаження (load)”.

Етап вилучення передбачає підготовку перевірку цілісності та правильності даних, додавання міток та позначень або збагачення нових сторонніх даних наявними наборами даних.

Етап перетворення передбачає виявленні та виправленні (або видаленні) пошкоджених, неточних, дублюючих або аномальних даних та перетворення їх у стандартний формат.

Після того, як необроблені дані видобуто та адаптовано, вони завантажуються в цільову систему, що забезпечує синхронізацію - безперервний процес оновлення даних між джерелами та призначеннями для забезпечення їх узгодженості.

Оркестрування даних часто плутають з оркеструванням робочих процесів. Оркестрування робочих процесів – це процес запуску та моніторингу стану завдань. Її сутність реалізувати аналог event-driven систем на базі пакетної обробки.

Оркестрування даних є підмножиною оркестрування робочих процесів і забезпечує надійну та ефективного синхронізацію даних у продакшен середовищі. На ряду з елементами ETL вона може також включати: елементи управління середовищем (CI/CD для даних або GitOps для даних або безперервна інтеграція та доставка даних), контроль доступу на основі ролей

("RBAC"), оповіщення та моніторинг стану даних.

Інструменти оркестрування робочих процесів існували вже протягом тривалого часу, але лише нещодавно їх почали називати інструментами оркестрування даних.

По суті, всі вони дуже схожі, пропонуючи фреймворк для написання коду та виконання орієнтованих ациклічних графів (DAG) у контейнеризованому середовищі.

Зазвичай логіка DAG описується на Python, що забезпечує гнучкість та інтеграцію з різноманітними бібліотеками Python для обробки даних та інших завдань.

Інструменти дозволяють запускати DAG вручну або автоматично на основі певних подій (тригерів) або за деяким розкладом. Вони також надають функції моніторингу для відстеження виконання DAG та оповіщення про помилки або успішне завершення.

На рисунку 2.1 наведена узагальнена архітектура інтрументів оркестрування даних.

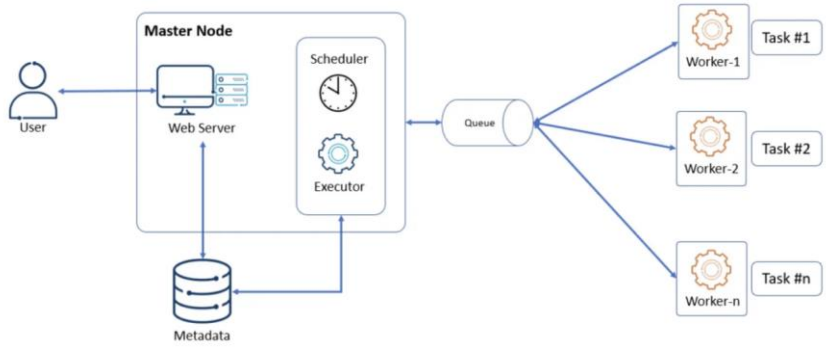


Рисунок 2.1 – Узагальнена архітектура інструментів оркестрування

Отже згідно з наведеної архітектури можна навести 4 загальні компоненти інструментів оркестрування [13].

Планувальник є критичним компонентом. Його основна функція

полягає в безперервному скануванні DAGs для виявлення та планування завдань на основі їх залежностей та заданих часових інтервалів. Планувальник відповідає за визначення, які завдання виконувати і коли. Він взаємодіє з метаданими бази даних для зберігання та отримання інформації про стан завдань та їх виконання.

Інструменти зазвичай використовують базу даних метаданих, таку як PostgreSQL або MySQL, для зберігання всіх деталей конфігурації, станів завдань та метаданих виконання. База даних метаданих забезпечує збереження і гарантує, відновлення після збоїв і продовження виконання завдань з останнього відомого стану. База даних також служить центральним репозиторієм для управління і моніторингу виконання завдань.

Компонент веб-сервера надає інтерфейс користувача для взаємодії з інструментом. Він дозволяє користувачам контролювати виконання завдань, переглядати стан DAGs та отримувати доступ до журналів і іншої операційної інформації. Веб-сервер взаємодіє з базою даних метаданих для отримання відповідної інформації і представляє її у зручному для користувача вигляді. Користувачі можуть запускати завдання вручну, відстежувати прогрес виконання завдань та налаштовувати параметри через інтерфейс веб-сервера.

Виконавець відповідає за виділення ресурсів і виконання завдань на зазначених робочих вузлах.

2.2 Огляд сучасних засобів оркестрування даних

Під засобами в даному випадку мається на увазі інструменти оркестрування. Перейдемо до безпосереднього розгляду інструментів.

Prefect [20] був заснований у 2018 році і позиціонується як "новий стандарт автоматизації потоків даних". Зовнішній вигляд інтерфейсу користувача Prefect наведено на рисунку 2.2.

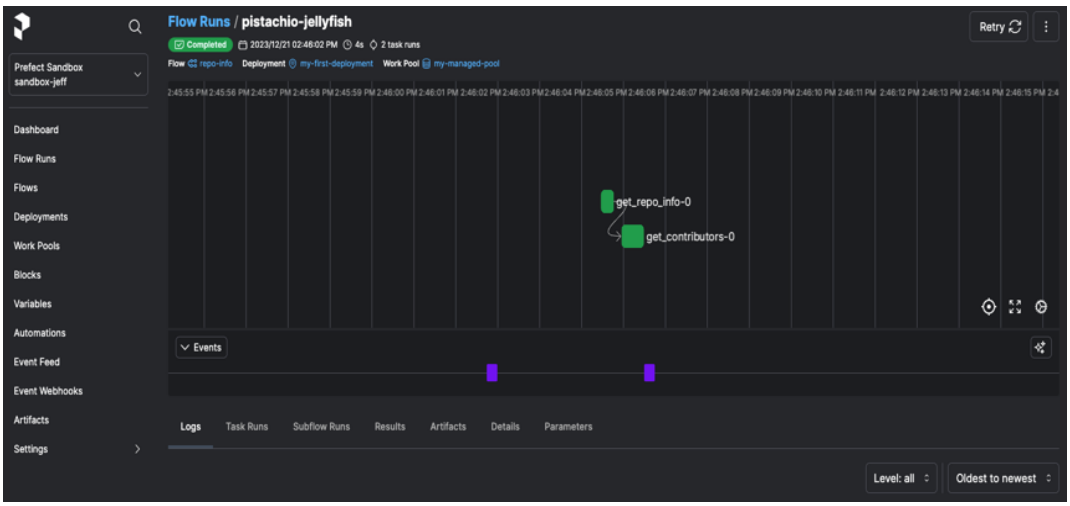


Рисунок 2.2 – Зовнішній вигляд КІ Prefect

Prefect має зручний КІ і спеціально розроблений для роботи з даними. Він пропонує простий і централізований спосіб керування та моніторингу всього стеку обробки даних. За рахунок додавання кількох декораторів до Python-коду, забезпечується перетворення Python функції на завдання Prefect, що дозволяє інтегрувати їх у робочі процеси. Prefect доступний розробникам з базовими знаннями Python, що полегшує початок оркестрації конвеєрів обробки даних.

Prefect пропонує унікальну "гібридну модель" для оркестрації робочих процесів, яка вирішує критично важливу проблему, яка постає при використані хмарних рішень: довіра. Традиційні служби оркестрації вимагають запуску коду клієнта на інфраструктурі провайдера. Це створює проблему довіри: клієнти повинні довіряти постачальнику свій конфіденційний код, а провайдери повинні гарантувати, що код не є зловмисним. Локальні рішення стикаються з аналогічною проблемою довіри, хоча й у протилежному напрямку (клієнти не довіряють програмному забезпеченню постачальника).

Prefect Cloud – сервіс, зосереджений виключно на оркестрації, а не на виконанні коду. Це усуває необхідність для клієнтів довіряти Prefect свій код. Клієнти зберігають повний контроль і безпеку: їх код залишається на їх

приватній інфраструктурі.

Отже типовий процес розробки виглядає наступним чином. Клієнти проектують і тестують робочі процеси за допомогою Prefect Core на власній інфраструктурі. Після завершення робочий процес реєструється в Prefect Cloud. При цьому передаються метадані про робочий процес (завдання, залежності, розклад тощо) - не сам код. Prefect Cloud використовує отримані метадані для оркестрації робочого процесу без необхідності в коді. Він керує плануванням розкладу, контролює стани виконання (запуск, успіх, помилка, повторний запуск) і координує кілька паралельних виконань. Далі Prefect Agent, що працює на інфраструктурі клієнта, виконує завдання локально або у віддаленому кластері. та передає стан виконання назад до Prefect Cloud.

У 2021 році був створений Mage [21] – інструмент оркестрації, спеціально розроблений для ETL-конвеєрів даних. Зовнішній вигляд інтерфейсу користувача Mage наведено на рисунку 2.3.

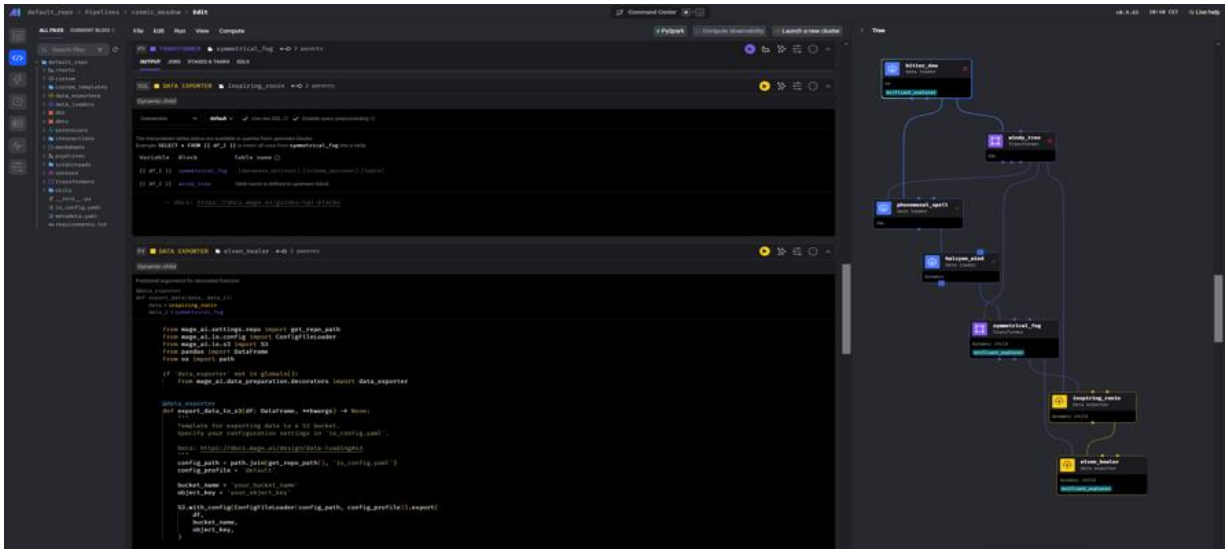


Рисунок 2.3 – Зовнішній вигляд КІ Mage

Він працює за допомогою блоків, які можуть мати декілька залежностей від попередніх або наступних блоків. Кожен блок має свій ізольований код, призначений для легкого повторного використання в різних конвеєрах.

Mage пропонує зручний КІ у вигляді інтерактивного блокнота для легкої розробки. Блокноти підтримують розробку кода на Python, R та SQL і дозволяють не турбуватися надто про обробку винятків, оскільки Mage бере це на себе. За допомогою Mage можна встановлювати зв'язки між різними блоками конвеєра безпосередньо через інтерфейс, що усуває потребу в додатковому коді.

Одна з переваг використання Mage – це можливість переглядати та візуалізувати результати конвеєра даних в режимі реального часу. Це дозволяє перевіряти та аналізувати дані без очікування розгортання в шарі оркестрації.

На додаток до розробки конвеєрів для пакетної обробки, Mage також дозволяє створювати конвеєри для стрімінгу даних. Наразі Mage підтримує Kafka, Azure Event Hub, Google Cloud PubSub, Kinesis та RabbitMQ сервіси для роботи зі стрімінгом даних, що дозволяє обробляти та аналізувати дані в режимі реального часу.

Слід зазначити що, для запуску повного набору інструментів оркестрації іншим аналогам може знадобитися кілька Docker-контейнерів. Це ускладнює роботу на окремому без серверному Docker-екземплярі, такому як Google Cloud Run або аналогічному Azure Container Instances. Mage робить акцент на простоті, він працює на одному Docker-контейнері, що значно полегшує його розгортання.

У 2022 році був розроблена Kestra [22] з метою полегшення масштабування оркестрації робочих процесів. Вона здатна обробляти збільшення робочого навантаження, дозволяючи розподіляти завдання між декількома вузлами кластеру. Зовнішній вигляд інтерфейсу користувача Kestra наведений на рисунку 2.4.

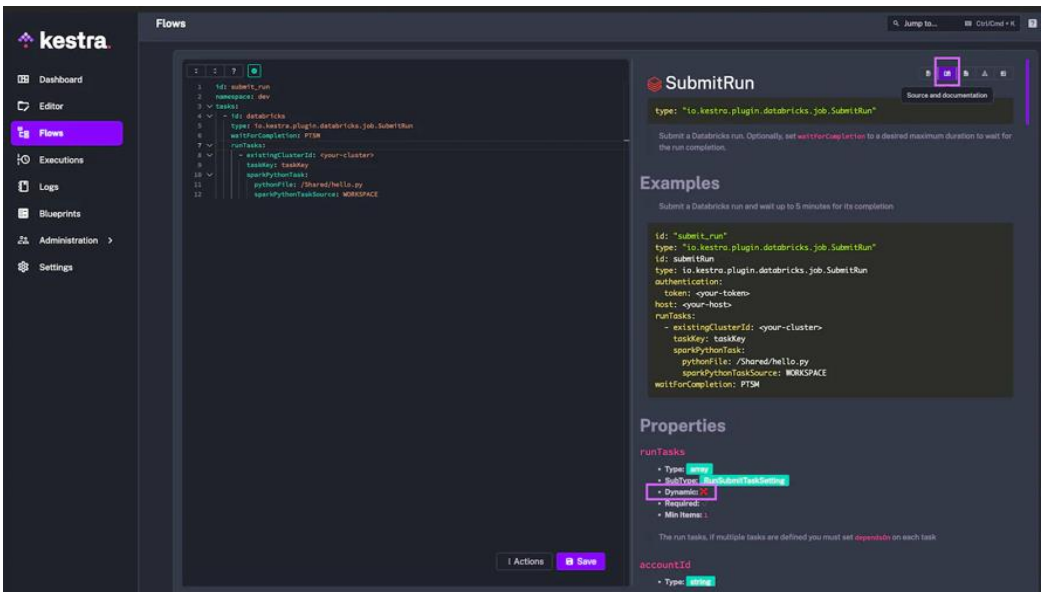


Рисунок 2.4 – Зовнішній вигляд КІ Kestra

Останнім часом порівнюють Kestra з Terraform, оскільки обидва інструменти використовують DSL для керування та автоматизації робочих процесів. Terraform справив значний вплив на DevOps, встановивши стандарт IaC. Kestra прагне запровадити схожі принципи в сфері оркестрації та автоматизації, хоча й з деякими відмінностями в підході.

Цей декларативний підхід означає, що визначається бажаний кінцевий стан, а не кроки для досягнення цього стану. За допомогою визначення робочих процесів у конфігураційному файлі YAML, Kestra абстрагується від складнощів процедурного коду, що робить створення, розуміння та підтримку робочих процесів легшими.

Хоча Kestra використовує YAML для визначення робочих процесів, вона пропонує свободу використання будь-якої мови програмування для написання скриптів всередині робочих процесів. Ця незалежність від конкретної мови дозволяє розробникам використовувати свої наявні навички, чи то Python, R, або інші мови.

Дизайн Terraform зосереджений на модульних конструкціях, які дозволяють розділяти складні ресурси на повторно використовувані компоненти. Kestra підтримує цю філософію модульності. За допомогою

таких функцій, як креслення (blueprints) та підпроцеси (subflows), вона пропонує структурований підхід до створення робочих процесів, що покращує їх повторне використання та обслуговування. Особливо виділяються підпроцеси Kestra, оскільки вони дозволяють повторно використовувати частини робочих процесів у різних операціях, полегшуючи оновлення та зміни в єдиному централізованому компоненті.

У 2022 році був заснований Dagster [23], що моделює конвеєри даних з точки зору активів даних, які вони виробляють та споживають. Зовнішній вигляд користувача Dagster наведено на рисунку 2.5.

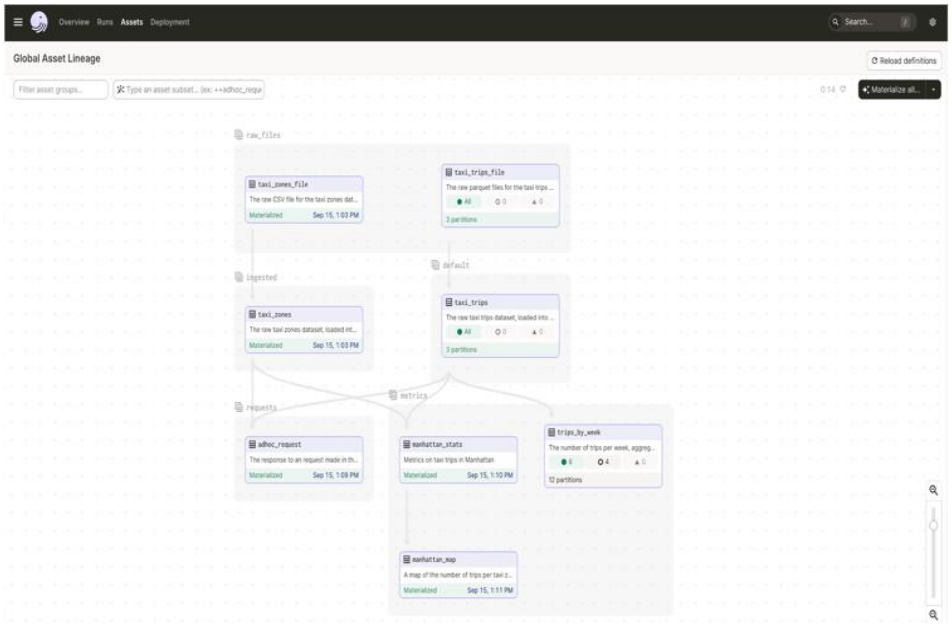


Рисунок 2.5 – Зовнішній вигляд КІ Dagster

Актив - це об'єкт у постійному сховищі, наприклад, моделі dbt, таблиці Snowflake або навіть CSV-файли. Програмно-визначений актив - це опис в коді, який повинен існувати і як створювати та оновлювати цей актив. Програмно-визначені активи дозволяють використовувати декларативний підхід до управління даними, в якому код є джерелом істини щодо того, які активи даних повинні існувати і як вони обчислюються.

Dagster вважає, що оркестратор повинен бути обізнаним щодо активів.

Ця інформація використовується для побудови Каталогу активів Dagster. Це принципово нового погляду на роботу оркестратора, який пов'язує активи з обчисленнями, що їх генерують. Користувачі можуть переходити до оркестратора, шукаючи створені активи, а не конвеєри, які їх створили.

Для визначення робочого процесу у Dagster користувач пише декоровані функції Python, що визначають обчислення та структуру графа. Подібно до Airflow, реалізація окремого вузла графа може робити все, що може Python. Проте, Dagster вважає, що функції повинні офіційно декларувати свої вхідні та вихідні дані, надавати гарантії типізації для цих даних, визначати необхідну конфігурацію тощо.

Dagster API дозволяє розділити обов'язки між обчисленнями та вводом/виводом. Це необхідна умова для тестування програм для роботи з даними, що забезпечують швидкий зворотній зв'язок для розробників. Наприклад, у режимі "тестування" ресурс може зберігати фрейм даних у файлову систему, а ресурс у режимі "виробництво" може зберігати фрейм даних у сховищі об'єктів, такому як S3

Dagster є достатньо гнучким для виконання обчислень без необхідності спеціальної інфраструктури. Не потрібна інфраструктура, розклад або реєстрація стану для виконання конвеєрів. Структура графа та розклад виконання є розділеними концепціями: розклад та сенсори визначаються незалежно від конвеєра.

2.3 Порівняння засобів оркестрування даних

Порівняльна характеристика інструментів наведена у таблиці 2.1, проаналізувавши яку, були зроблені висновки що кожного окремого інструменту.

Prefect добре підходить , коли не було попереднього досвіду в побудові DAG-ів і потрібний простий в освоєнні інструмент з великою спільнотою, а також гібридна модель.

Таблиця 2.1 – Порівняльна характеристика інструментів

Критерій	Prefect	Mage	Kestra	Dagster
Модель виконання	Гібридна (локальна/хмарна)	Контейнеризована	Універсальна	Універсальна
Підхід до опису робочих процесів	Декоратори Python	Python	YAML	Python
Мови програмування	Python	Python	Python, R, Julia, Node.js, etc.	Python
Спільнота	Велика	Зростаюча	Зростаюча	Зростаюча
Гнучкість	Висока	Середня	Висока	Висока
Масштабованість	Висока	Висока	Висока	Висока
Складність використання	Середня	Низька	Низька	Середня
Підтримка SaaS та хмарних сервісів	Висока	Висока	Середня	Низька
Відображення залежностей	Динамічне	Статичне	Статичне	Статичне
Підтримка контейнеризації	Висока	Висока	Середня	Низька

Mage підходить для тих випадків, коли потрібна обробка даних в реальному часі та розгортання в контейнеризованих середовищах.

Kestra чудовий вибір для команд, які не мали попереднього досвіду в побудові DAG-ів але мають гарний досвід з розгортанням інфраструктури з застосування принципів IaC.

Dagster добре підходить, коли потрібен інструмент для оркестрації dbt процесів та коли надається пріорітет наглядності та простоті у розумінні робочого процесу.

3 ПОБУДОВА ЕСПЕРИМЕНТАЛЬНОЇ СИСТЕМИ

3.1 Загальна постановка задачі та огляд експериментальної системи

В рамках експерименту необхідно реалізувати ПЗ експериментальної системи для автоматизації обробки великих даних та аналітичного аналізу.

Почнемо розгляд системи з розуміння джерел даних. В даному випадку джерелом даних є E-commerce системи компанії виробника велосипедів AdventureWorks. Більшість даних перебуває у OLTP базі даних – MS SQL Server, але значний об’єм даних приходить у json файлах.

Взаємозв’язок джерел даних зображений на рисунку 3.2, де SalesOrderHearder SalesOrderDetail джерела приходять у файлах, в той час як усі інші джерела походять з бази даних.

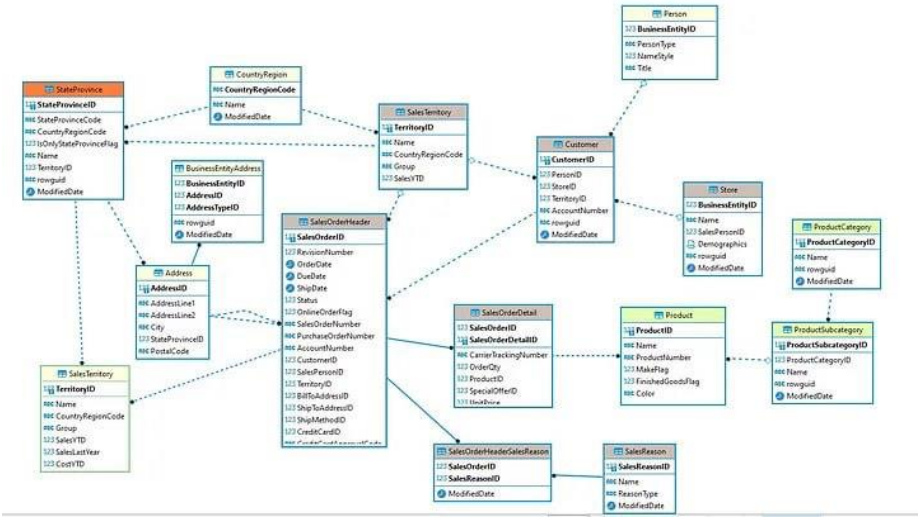


Рисунок 3.1 – Взаємозв’язок джерел даних

Перейдемо до основних мотивів побудови системи. Отже, підприємство хоче відстежувати свої продажі та продуктивність продукції в різних географічних місцях і в різні часи, що передбачає побудову аналітичних панелей з метою наглядності у відображенні ключових аспектів

проведеної аналітики.

Складність цієї задачі полягає в тому що наразі нема централізованого сховища даних, яке б слугувало джерелом правди, а також яке б мало стандартизований формат даних і яке було заточене на виконання складних аналітичних запитів.

Специфіка реалізації полягає у тому що не можливо обрати один конкретний процес: ELT чи ETL, адже не зважаючи на те що більшість обробки даних буде відбуватися на стороні сховища даних, є необхідність попередньо обробки даних, що приходять з файлів. Це призводить у необхідності реалізації як ELT так і ETL, що в свою чергу призводить до обчислень як на стороні сховища даних так і поза його межами, що потребує використання як різних фреймворків так і фахівців з різною експертизою.

Спрощена діаграма архітектури експериментальної системи наведена на рисунку 3.1.

Тож в реальному житті за частини 1, 2, 3 архітектури на рис.3.1 відповідали б відповідно окремі люди/команди. Тож перша команда відповідальна за оркестрування та моніторинг попередньої обробки даних з джерел і відвантаження їх до сховища даних. Друга команда залучена у оркестрування та моніторингу побудови та оновлення сховища даних. В той час як третя проводить аналітичну діяльність з метою відповісти на ключові питання бізнесу і результатом чого є побудовані аналітичні панелі.

При розробці експериментальної системи були обрані настуапні фреймворки: для трансформації даних PySpark і DBT в рамках ELT/ETL процесів, а також Prefect та Dagster для автоматизації та моніторингу цих процесів. Візуалізація даних [24] для проведення аналітичного аналізу повинна відбуватися в Power BI [26]. В якості сховища даних треба використовувати Postgresql.

Вхідними даними є json файли з необробленими даними, а також таблиці у MS SQL Server.

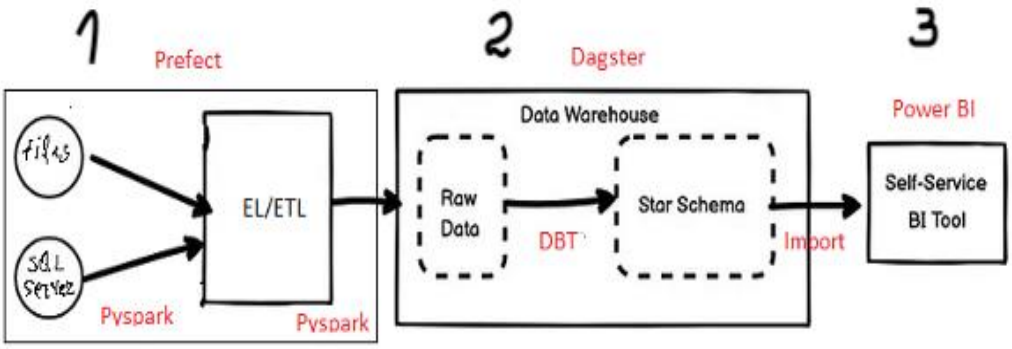


Рисунок 3.2 – Спрощена діаграма архітектури експериментальної системи

ПЗ експериментальної системи повинно зберігатися як сукупність файлів на накопичувачі або жорсткому диску.

Результатом роботи будуть оброблені дані, які зберігаються у Postgresql сховищев якості таблиць та аналітичні панелі у Power BI.

При написанні програми необхідно використати операційну систему Windows 10, мову програмування Python, середовище програмування PyCharm .

Вимоги до апаратного забезпечення при розробці ПЗ: ПК на базі процесора Intel з частотою не менше, ніж 1.5Ггц, з ОЗУ не менше ніж 8гб, операційна система Windows 10.

3.2 Реалізація оркестрування процесу обробки даних поза сховищем даних

Розглянемо реалізацію першу частину системи, де більше сконцентруємось на оркеструванні аніж трансформаціях.

Оркестрування ETL процесів за додомогою Prefect виявляється доволі не складно, адже для того щоб перетворити код у конвеєр обробки даних достатньо встановити додатковий пакет в пайтон проекті та обернути відповідні функції відповідними декораторами. Ця простота також досягається за рахунок того, що Prefect сповідує гібридну модель і йому все

одно де виконується код.

Перейдемо до розгляду коду, для простоти код був виконаний локально, тобто на власній інфраструктурі. Код конвеєру наведений у таблиці 3.1. Попередньо зазначимо, що трансформації написані на Spark фреймворці.

Таблиця 3.1 - Код Prefect конвеєру обробки даних

№	Код

1	`from pathlib import Path`
	`import os`
	`from prefect import flow, task`
	`from pyspark.sql import SparkSession`
	`from pyspark.sql import DataFrame`
	`import pyspark.sql.functions as F`
	`import pyspark.sql.types as T`

2	`spark = SparkSession.builder \`
	`    .appName("PySpark SQL Server Connection") \`
	`    .config("spark.jars", "../jars/postgresql-42.6.0.jar").getOrCreate()`

3	`@task(retries=2)`
	`def extract_json(table_name: str) -> DataFrame:`
	`    path = str(Path(__file__).parents[2] / "PrefectProject" / "data" / f"{table_name}.json")`
	`    df = spark.read.json(path)`
	`    return df`

4	`@task(retries=2)`
	`def transform(df: DataFrame, dt_columns: list, dollar_columns: list, debug: bool) -> DataFrame:`
	`    m_df = df.filter(F.col("rowguid").isNotNull())`
	`    for col in dt_columns:`
	`        m_df = m_df.withColumn(col, F.to_date(col, "M/d/y hh:mm:ss a"))`
	`    for col in dollar_columns:`
	`        m_df = m_df.withColumn(col,`
	
	`F.regexp_replace(F.expr(f"substr({col}, 2, length({col}))"), ",", "").cast(T.DoubleType()))`
	`    if debug:`
	`        cols = ["rowguid"]`
	`        cols.extend(dt_columns)`
	`        cols.extend(dollar_columns)`
	`        m_df.show(n=5, truncate=False)`
	`        print("Records cnt: ", m_df.count())`
	`    return m_df`

Продовження таблиці 3.1

№	Код

5	`@task(retries=2)`
	`def load_postgres(df, table_name: str):`
	`  url = "jdbc:postgresql://localhost:5432/adventureworks?schema=source"`
	`  user = "postgres"`
	`  password = os.environ['password']`
	`  driver = "org.postgresql.Driver"`
	`  df.write \`
	`  .format("jdbc") \`
	`  .option("url", url).option("driver", driver) \`
	`  .option("dbtable", table_name) \`
	`  .option("user", user).option("password", password) \`
	`  .mode("overwrite").save()`

6	`@flow(log_prints=True)`
	`def etl_process(src_table_name, dt_cols, dollar_cols, tgt_table_name, debug=False):`
	`  """`
	`  etl process`
	`  """`
	`  df = extract_json(src_table_name)`
	`  transformed_df = transform(df, dt_cols, dollar_cols, debug)`
	`  load_postgres(transformed_df, tgt_table_name)`

7	`if __name__ == "__main__":`
	`  etl_process.serve(`
	`  name="facts_ETL_process",`
	`  tags=["fact_related_data"],`
	`  parameters={"src_table_name": "dbo.vw_salesorderheader",`
	`  "dt_cols": ['DueDate', 'OrderDate', 'ShipDate', 'ModifiedDate'],`
	`  "dollar_cols": ['Freight', 'SubTotal', 'TaxAmt', 'TotalDue'],`
	`  "tgt_table_name": "source.SalesOrderHeader"}`

Нижче наведений короткий опис коду конвеєра:

1) підключення зовнішніх бібліотек;

2) створення Spark сесії, як вхідної точки в Spark застосунках;

3) код вилучення ETL процесу (E), який перетворений у задачу

конвеєру за допомогою декоратора;

4) безпосередньо код трансформацій:

- відфільтрувати рядки з нулями;
- приведення таймстемпу до дати;
- видалення знаку долара та приведення до коректного типу.

5) код завантаження ETL процесу (L), який перетворений у задачу конвеєру за допомогою декоратора;

6) перетворення головної функції у конвеєр за допомогою декоратора;

7) розгортання конвеєра у Prefect Cloud.

Схожа програма була розроблена для EL процесів. Має сенс зазначити, що дані які приходять в файлах будуть використовуватися для побудови fact таблиці і будуть оброблятися у ETL процесах, а дані які знаходяться в базі даних будуть оброблятися у EL процесах та будуть використанні для побудови dimensional таблиць.

Аби розгорнути конвеєр у Prefect Cloud треба спочатку пройти аутентифікацію виконавши команду у середовищі розробки: `prefect cloud login -` а потім виконати код. Після цього на веб сторінці Prefect Cloud у вкладці Deployments можна побачити наші конвеєри готові до використання (наведено на рисунку 3.3).

Рисунок 3.3 – Deployments Prefect Cloud

Далі можна виконувати конвеєри в ручному режимі з параметрами за замовчуванням (наведені в 7 блоці таблички 3.1), або задавати кастомні параметри, або створювати розклад виконання. Розглянемо більш детально на прикладах.

Аби виконати конвеєри вручну з кастомними параметрами як на рисунку 3.4 (для EL процесів наведено лише один випадок), треба перейти до відповідного деплойменту та та нажати на трикутник(custom run).

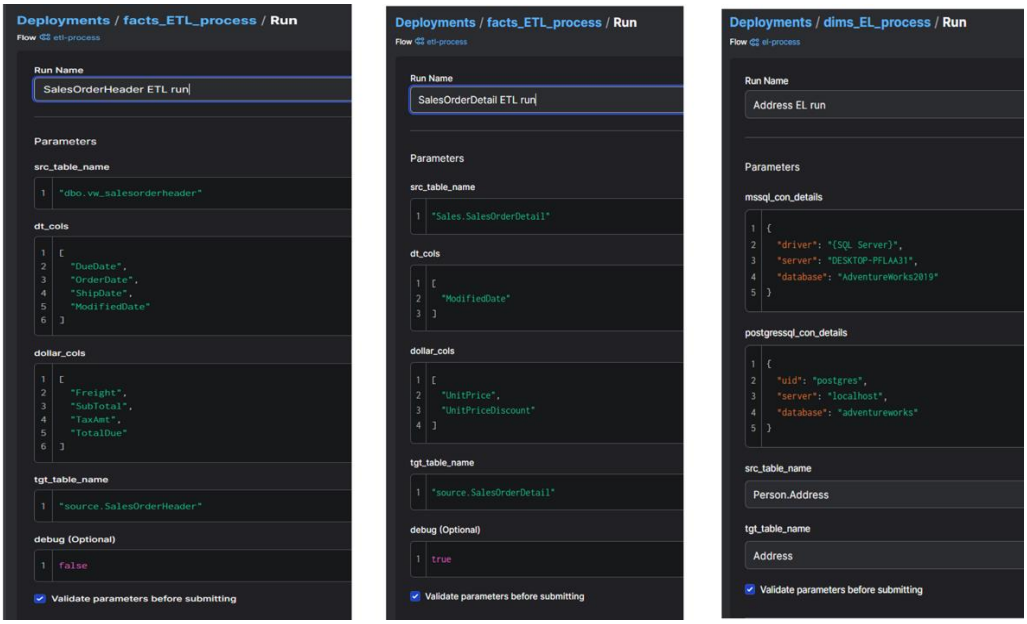


Рисунок 3.4 – виконання конвеєрів вручну у Prefect Cloud веб-застосунку

Якщо відкрити конкретний деплоймент (рисунок 3.5), то можно побачити найсвіжіші запуски та відповідну статистику

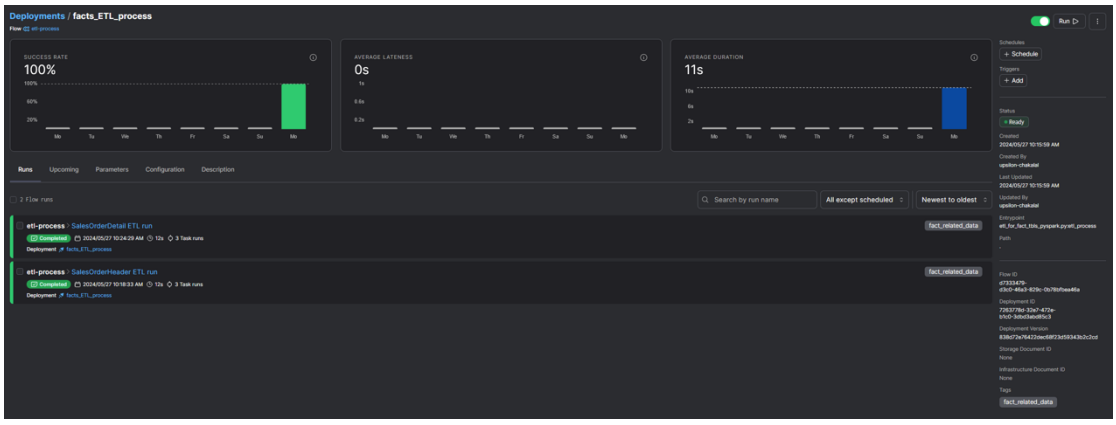


Рисунок 3.5 – Відображення конкретного деплоймента конвеєра у Prefect Cloud

У КІ також можно побачити інформацію щодо виконань конвеєрів у

вкладці Runs, але там є також можливість відфільтрувати виконання по назві деплойменту, конвеєра, тегах. А також побачити скільки їх було і як довго вони тривали (рисунок 3.6)

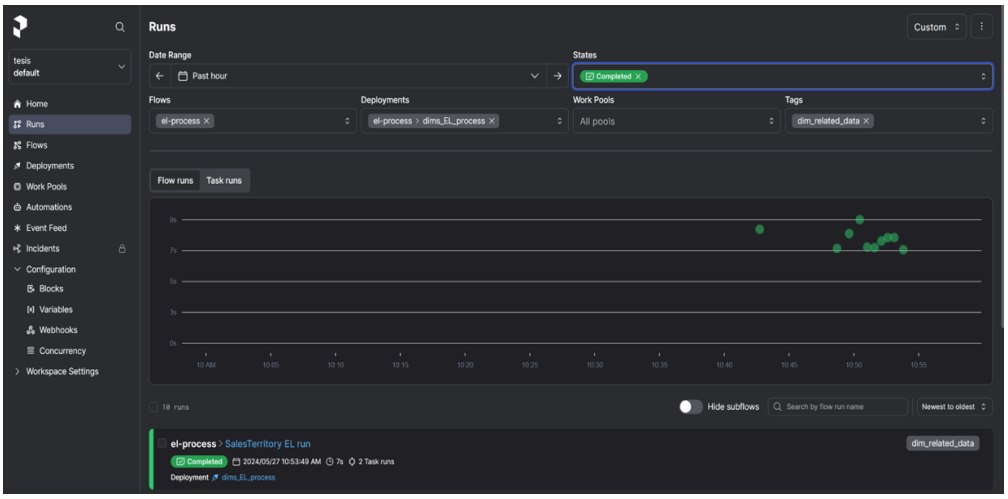


Рисунок 3.6– Відображення виконаних конвеєрів у Prefect Cloud веб-застосунку

Якщо розглянути якесь конкретне виконання конвеєру, то там можна побачити залежність підзадач конвеєра у вигляді графа на рисунку 3.7 (відповідають блокам кода 3, 4, 4 у таблиці 3.1), параметри з якими запускався конвеєр (ті які ми задавали, коли запускали конвеєр), та безпосередньо логи підзадач (рисунок 3.8). На логах зокрема службових записів можна побачити stdout блоку кода 4 таблиці 3.1, де виведено 20 записів та загальну кількість рекордів датасету.

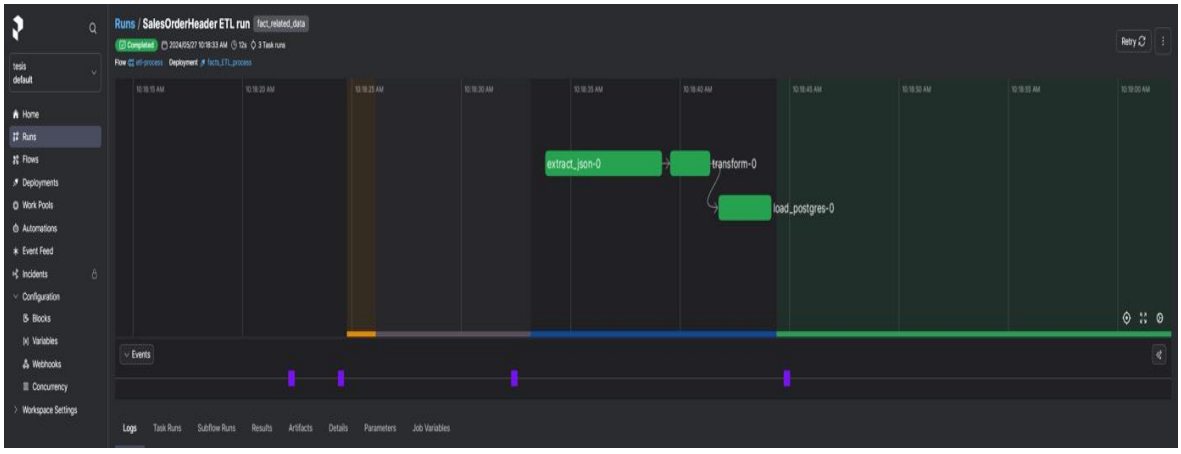


Рисунок 3.7 – Графічне відображення конвеєра у Prefect Cloud веб-застосунку

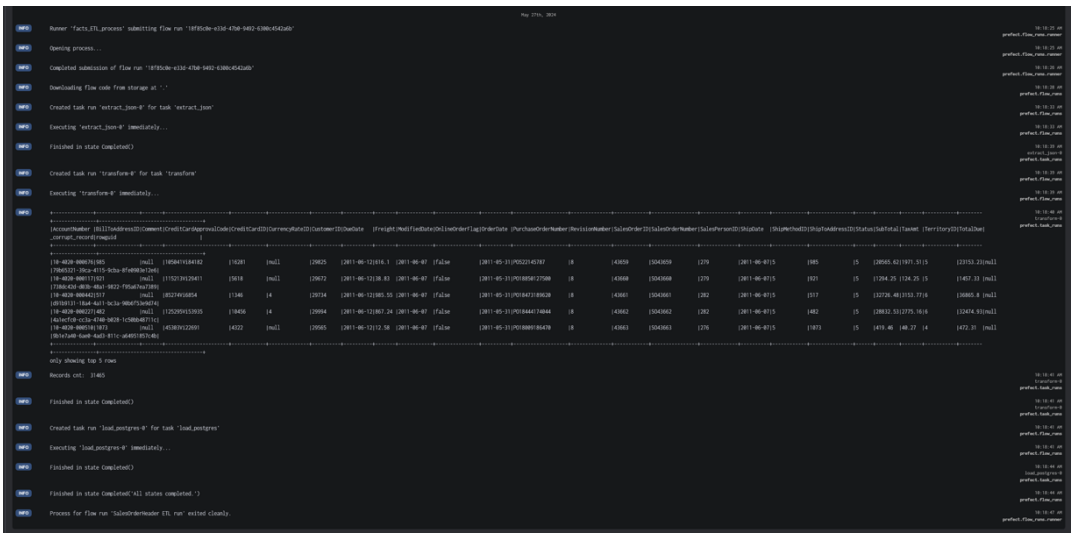


Рисунок 3.8 – Логи конкретного виконання конвеєра у Prefect Cloud веб-застосунку

Після того як є впевність що все працює локально, можно створити work pool, аби конвеєр виконувався на інфраструктурі відмінній від локальної, для цього достатньо виконати наступну команду: `prefect work-pool create my-managed-pool --type prefect:managed`.

Наступним логічним кроком є створити розклад для виконання конвеєрів, це можно зробити у вкладці Deployments, де запропоновано 3 варіанти. На рисунку 3.9 зображен варіант задання за допомогою крон виразу, який відповідаю в даному випадку означає виконання кожен день у десятій годині ранку.

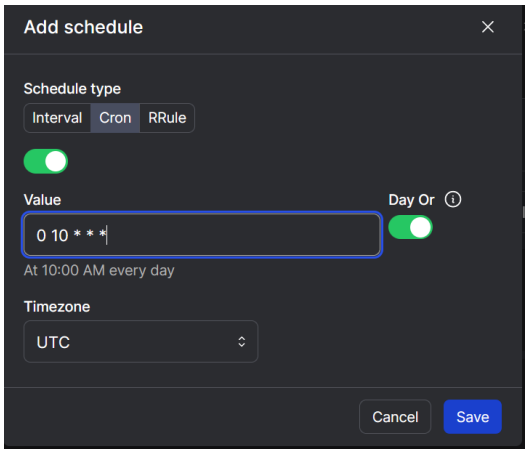


Рисунок 3.9 – Задання розкладу виконання конвеєра у Prefect Cloud

3.3 Реалізація оркестрування процесу обробки даних на стороні сховища

При побудові сховища даних було обрана модель даних запроваджена Кімбалом, метою якої є перетворення необроблених даних у таблиці фактів і вимірів. Цей вибір пояснюється рядом причин:

- адже модель добре підходить для побудови аналітичних моделей, що є взагалі нашим випадком;
- ця модель дуже ефективна у разі відслідковування змін та трендів даних;
- ця модель значно спрощує запити, зменшує кількість джоїнів та покращую читабельність запитів;
- попередній пункт також призводить до меншої тривалості виконання запитів.

В цій секції основний фокус на трансформаціях, адже значна частина їх відбувається на стороні сховища даних. Для розробки був обраний dbt фреймворк, адже він створювався саме для виконання задач по трансформації в рамках ELT процесів.

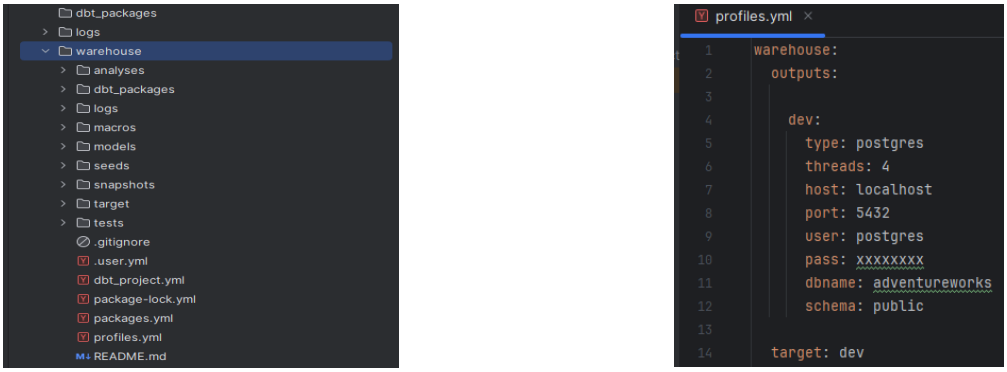
Він дозволяє створювати об’єкти бази даних, такі як таблиці, представлення, і перетворювати наші дані просто за допомогою операторів SQL select, автоматизувати перевірку якості даних і надавати надійні дані з автоматично створеною документацією разом із кодом.

Розпочати роботу з dbt доволі не складно, бо це не що інше як пайтон фреймворк. Тож для початку роботи потрібно лише встановити пакет відповідний до сховища даних, в даному випадку PostgreSQL: `pip install dbt-postgres` та виконати наступну команду аби створити скелет проекту (рисунок 3.10а): `dbt init warehouse`.

Аби почати виконувати запити у PostgreSQL треба додати конфігурацію з’єднання у profiles.yml, як наведено на рисунку 3.10б.

Однією з найпотужніших функцій dbt є те, що можно змінити тип

об’єкту в базі, просто змінивши значення конфігурації.



а) б)

Рисунок 3.10 – dbt проект: а) структура проекту б) конфігурацію з’єднання зі сховищем у dbt проекту

Також можна переключатися між таблицями та представленнями(views), просто змінивши ключове слово, без необхідності написання DDL запитів.

Ця конфігурація представлена на рисунку 3.11 і в той самий час вона є головною в проекті. Більшість її атрибутів згенеровано по замовчуванню, але атрібут models був модифікований аби створити потрібну структуру сховища даних. До речі, основним типом об’єктів сховища даних обрана табличка, що зазначено в атрібуті materialized. Також в конфігурації зазначено, що проміжні таблички будуть створюватися в stagging схемі, а факти та виміри у sales схемі сховища.

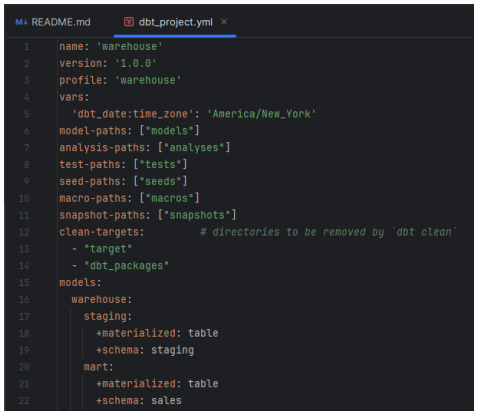


Рисунок 3.11 – Головна конфігурацію dbt проекту

Трохи забігаючи наперед продемонструємо граф походження табличок, згенерований dbt, зазначений на рисунку 3.12. Щодо створення проміжних табличок в stagging схемі(відповідає частині під цифрою 1) все доволі просто: відповідні запити просто роблять select необхідних для подальших трансформацій колонок. Ці запити розміщають в папці stagging у models (рисунок 3.13), де назва файлів з запитами відповідає назві проміжних табличок, які в результаті виконання запиту матеріалізується у сховищі.

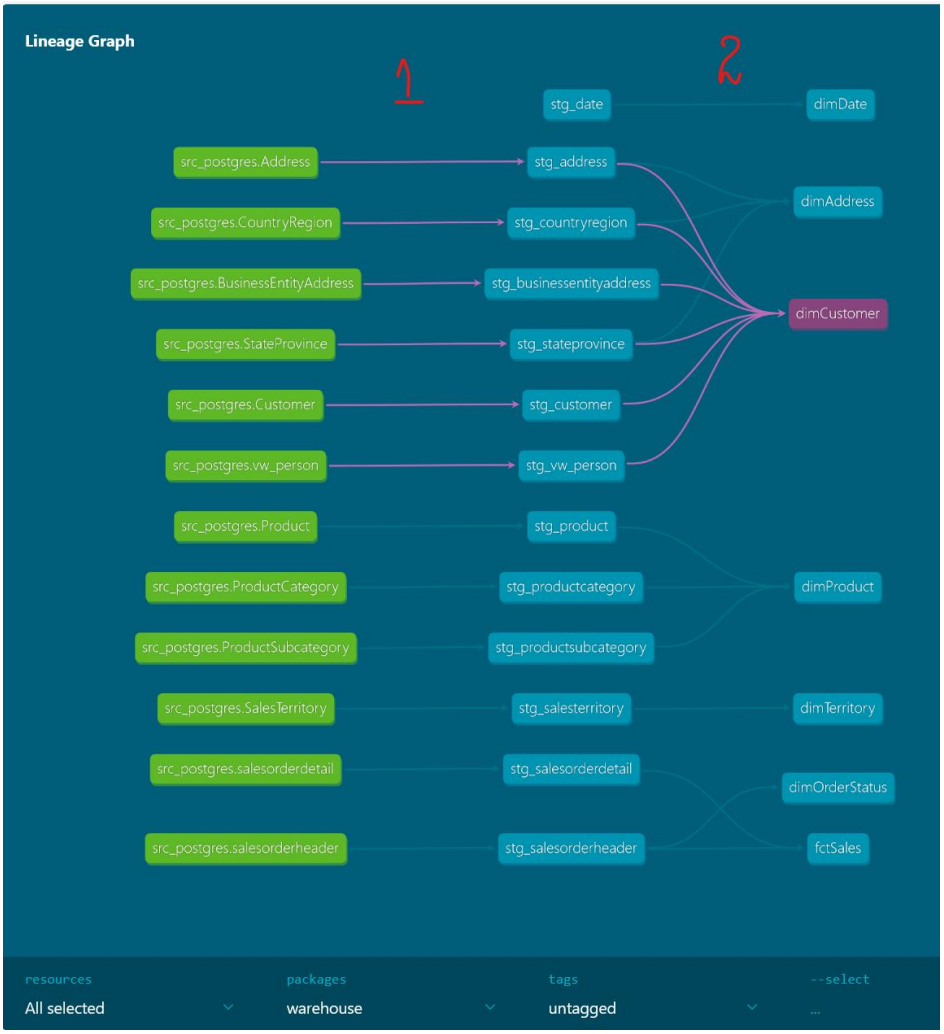


Рисунок 3.12 – Граф походження табличок dbt проекту

Також на рисунок 3.13 можна побачити конфігурацію джерел для проміжних табличок, де зазначені таблички які слугують джерелами, назва

бази даних, схема та її псевдонім, який ми бачимо на рисунку 3.12, де джерела відображені зеленим кольором.

Перейдемо до розгляду створення табличок фактів та вимірів, що можна побачити на частині рисунку 3.12 під цифрою 2. Усі відповідні запити були додані в файлах в папці mart з назвами, які відповідають кінцевим табличкам (рисунок 3.13).

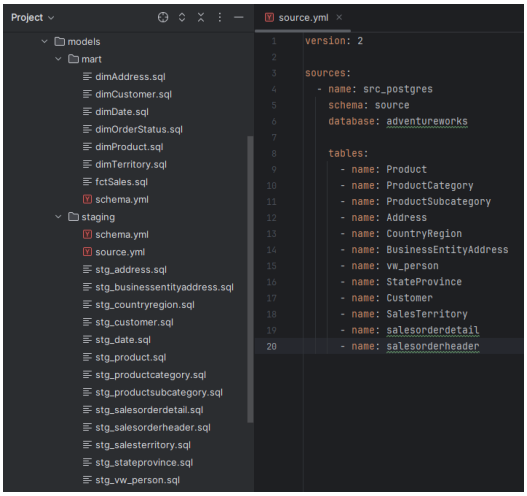


Рисунок 3.13 – Конфігурація джерел даних для stagging моделей dbt проекту

Після завершення написання усіх необхідних запитів і конфігурацій залишається ще один крок перед тим як матеріалізувати таблички – встановити пакети для коректної роботи макросів у запитах.

Макроси – це фрагменти коду SQL для багаторазового використання, які дозволяють інкапсулювати логіку та зменшити кількість повторюваних фрагментів коду у запитах. Вони схожі на функції в мовах програмування і можуть використовуватися для спрощення складних операцій SQL.

Для того щоб встановити пакет треба виконати команду `dbt deps`, після макроси підтягнуться до папок відповідних до пакетів всередині папки dbt_packages, як зазначено на рисунку 3.14.

Після цього можна матеріалізувати таблички, виконавши наступну

команду: `dbt run`, що в свою чергу скомпілює код запитів, тобто підмінить макроси і jinja template на відповідний код, готовий до виконання у сховищі і покладе це у папку target (рисунок 3.10а).

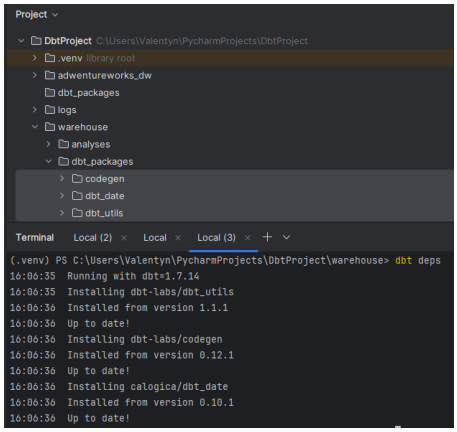


Рисунок 3.14 – Встановлення макетів з макросами у dbt проекту

Логи виконання матеріалізації можна побачити на рисунку 3.15.

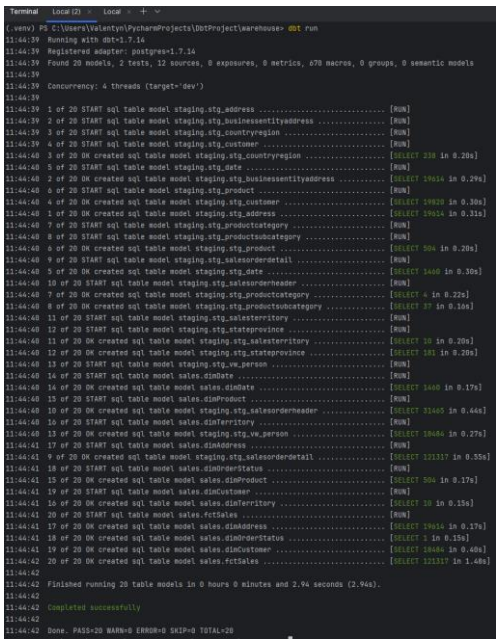


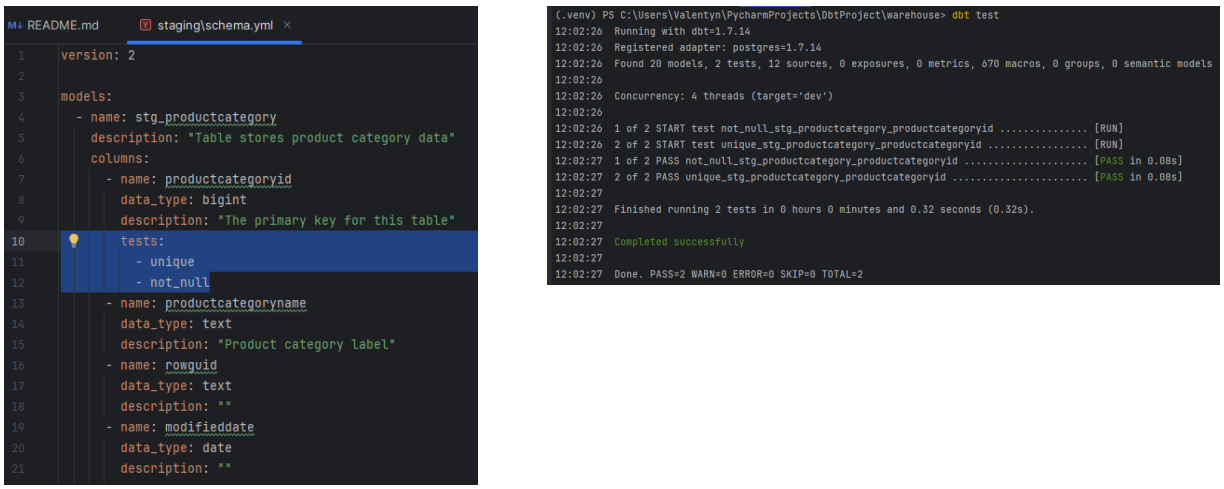
Рисунок 3.15 – Логи виконання матеріалізації у dbt проекту

Але замість матеріалізації всіх табличок, можна матеріалізувати окремі таблички, що вимагає невеликої модифікації команди, я наприклад: `dbt run`

`--select factSales`, де ми просто конкретизуємо яку саме табличку матеріалізувати. Трохи забігаючи наперед, зазначмо що коли ми в Dagster матеріалізуємо всі табличко або окремі, то під капотом виконуються зазначені вище команди.

У випадках коли треба почистити папки з артіфактами(target, dbt_packages) - можна виконати наступну команду: `dbt clean`.

dbt також надає можливість проводити валідацію моделей виконуючи тести. Це досягається додаванням атрібуту tests з відповідними значеннями у конфігурації schema.yml. dbt з коробки пропонує наступні тести: *unique*, *not_null*, *accepted_values*, *relationships*. Зокрема є можливість розробляти та інтегрувати свої тести. Логи виконання тестів та конфігурація наведені на рисунку 3.16.



а) б)

Рисунок 3.16 – Додавання тестів у dbt проекту: а) конфігурація тестів; б) логи виконання тестів

Перевагою dbt є також можливість автоматизувати процес документування. Аби згенерувати всі необхідні артіфакти необхідно виконати наступну команду: `dbt docs generate` – це відповідно створить ряд json файлів у target папці.

Після того, виконавши команду `dbt docs serve` і перейшовши до dbt

веб-застосунку за постланням `http://localhost:8080`, розглянемо детально згенеровану документацію разом з запитами для отримання табличок фактів та вимірів. Вигляд головної сторінки dbt застосунку зображен на рисунку 3.17.

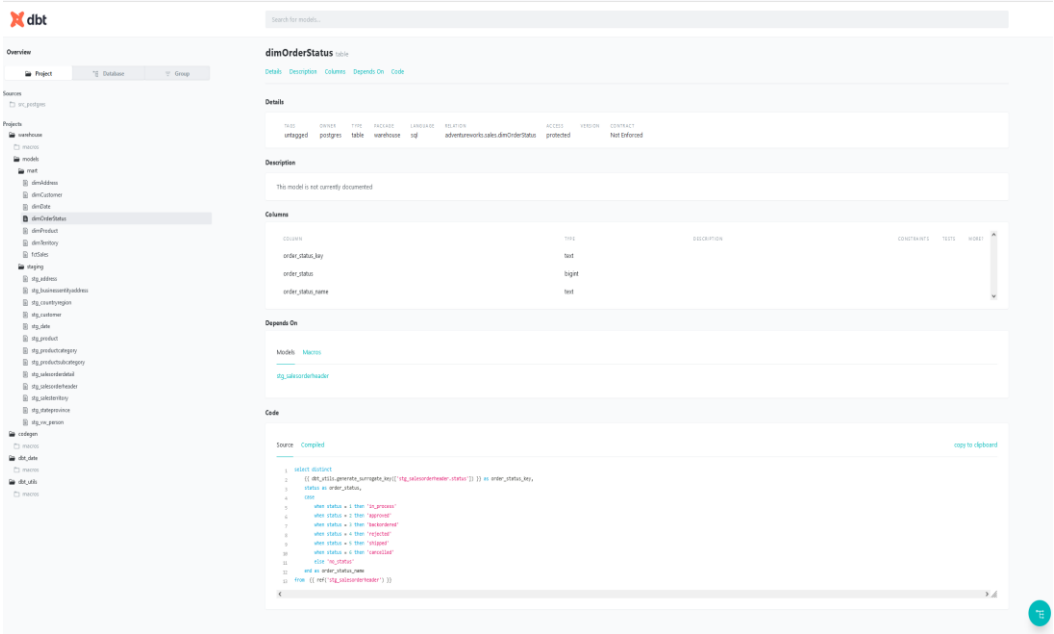


Рисунок 3.17 – Зовнішній вигляд документації у dbt веб-застосунку

Отже в лівій частині можна побачити панель навігації з трьома вкладками. Перша вкладку відкриває доступ до структури проекту, де ми можемо знайти як моделі так і макроси з підключених пакетів. У другій представлена безпосередньо структура сховища даних. Нагорі є поле для пошуку моделі, а правому нижньому куті посилання на граф походження табличок (рисунке 3.12).

Перейдемо до детального опису моделі виміру dimOrderStatus, який розділений на п’ять секцій. В першій секції Details представлена загальна інформація про табличку. В секції Discription може бути наданий опис з точки зору бізнесу. Далі в секції Columns наведений опис схеми таблички. В наступній секції зазначені залежності від інших табличок (можна також вивести графічне зображення натиснувши на кнопку в правому куті) та

макроси які використовувалися. А в останній наведений код з jinja template та кінцевий, який буде виконуватися у сховищі.

На рисунку 3.18 для порівняння наведено код до компіляції та після, де можна побачити, що макрос був підмінений відповідним кодом, як і jinja template відповідними значенням.

На рисунках 3.19-3.24 буду наведені запити решти табличок вимірів та фактів.

На рисунку 3.25 наведено ERD діаграму сховища побудовану у Power BI по моделі Кімбала по схемі зірочка.

Рисунок 3.18 – Запит моделі dimOrderStatus виміру: а) до компіляції б) після компіляції

Рисунок 3.19 – Запит моделі dimCustomer виміру

Source Compiled

```
select
     {{ dbt_utils.generate_surrogate_key(['stg_salesterritory.territoryid']) }} as territory_key,
    territoryid,
    salesterritoryname,
    "group" as territory_group,
    countryregioncode,
    costytd,
    salesytd,
    costlastyear,
    saleslastyear,
    modifieddate
from {{ ref('stg_salesterritory') }}
```

Рисунок 3.20 – Запит моделі dimTerritory виміру

Source Compiled copy to clipboard

```
select
    {{ dbt_utils.generate_surrogate_key(['stg_product.productid']) }} as product_key,
    stg_product.productid,
    stg_product.productname as product_name,
    stg_product.productnumber,
    stg_product.color,
    stg_product.daystomanufacture,
    stg_product.safetystocklevel,
    stg_product.standardcost,
    stg_productsubcategory.productsubcategory as product_subcategory_name,
    stg_productcategory.productcategory as product_category_name,
    stg_product.sellstartdate,
    stg_product.sellenddate
from  {{ ref('stg_product') }}
left join  {{ ref('stg_productsubcategory') }} on stg_product.productsubcategoryid = stg_productsubcategory.productsubcategoryid
left join  {{ ref('stg_productcategory') }} on stg_productsubcategory.productcategoryid = stg_productcategory.productcategoryid
```

Рисунок 3.21 – Запит моделі dimProduct виміру

Source Compiled

```
select
    {{ dbt_utils.generate_surrogate_key(['stg_address.addressid']) }} as address_key,
    stg_address.addressid,
    stg_address.city as city_name,
    stg_address.postalcode,
    stg_address.addressline1 || ' '|| coalesce(stg_address.addressline2, '') as Addressline,
    stg_stateprovince.statprovincename as state_name,
    stg_countryregion.countryregionname as country_name
from {{ ref('stg_address') }}
left join {{ ref('stg_stateprovince') }} on stg_address.stateprovinceid = stg_stateprovince.stateprovinceid
left join {{ ref('stg_countryregion') }} on stg_stateprovince.countryregioncode = stg_countryregion.countryregioncode
```

Рисунок 3.22 – Запит моделі dimAdress виміру

Source Compiled

```
with date_dimension as (
    select * from {{ ref('stg_date') }}
),
full_dt as (
    {{ dbt_date.get_base_dates(start_date="2011-01-01", end_date="2014-12-31") }}
),
full_dt_tr as (
select
    d.*,
    f.date_day as fulldt,
    {{ dbt_date.convert_timezone("f.date_day", "America/New_York", "UTC") }} as dulldtz,
    {{ dbt_date.convert_timezone("f.date_day", "America/New_York", source_tz="UTC") }} as dulldtzt,
    {{ dbt_date.convert_timezone("f.date_day", "America/New_York") }} as test,
    f.date_day::timestamp "direct_dts",
    f.date_day::timestamp AT TIME ZONE 'UTC' AS "ts_utc"
from
    date_dimension d
    left join full_dt f on d.date_day = cast(f.date_day as date)
)
select
        {{ dbt_utils.generate_surrogate_key(['direct_dts']) }} as date_key,
    *
From full_dt_tr
```

Рисунок 3.23 – Запит моделі dimDate виміру

Source Compiled

```
select
    {{ dbt_utils.generate_surrogate_key(['stg_salesorderdetail.salesorderid', 'salesorderdetailid']) }} as sales_key,
    {{ dbt_utils.generate_surrogate_key(['productid']) }} as product_key,
    {{ dbt_utils.generate_surrogate_key(['customerid']) }} as customer_key,
    {{ dbt_utils.generate_surrogate_key(['creditcardid']) }} as creditcard_key,
    {{ dbt_utils.generate_surrogate_key(['shiptoaddressid']) }} as ship_address_key,
    {{ dbt_utils.generate_surrogate_key(['status']) }} as order_status_key,
    {{ dbt_utils.generate_surrogate_key(['orderdate']) }} as order_date_key,
    {{ dbt_utils.generate_surrogate_key(['shipdate']) }} as ship_date_key,
    {{ dbt_utils.generate_surrogate_key(['duedate']) }} as due_date_key,
    {{ dbt_utils.generate_surrogate_key(['territoryid']) }} as territory_key,
    {{ dbt_utils.ge}}
    orderdate,
    onlineorderflag,
    stg_salesorderdetail.unitpricediscount as unitpricediscount,
    stg_salesorderheader.salesordernumber,
    stg_salesorderdetail.salesorderid,
    stg_salesorderdetail.salesorderdetailid,
    stg_salesorderdetail.unitprice,
    stg_salesorderdetail.orderqty,
    stg_salesorderdetail.linetotal as revenue,
    stg_salesorderdetail.linetotal as salesamount,
    case
        when stg_salesorderdetail.unitpricediscount > 0 then stg_salesorderdetail.linetotal * stg_salesorderdetail.unitpricediscount
        else stg_salesorderdetail.linetotal
    end as totaldiscount,
    stg_salesorderheader.taxamt
from {{ ref('stg_salesorderdetail') }}
inner join  {{ ref('stg_salesorderheader') }} on stg_salesorderdetail.salesorderid = stg_salesorderheader.salesorderid
```

Рисунок 3.24 – Запит моделі dimDate виміру

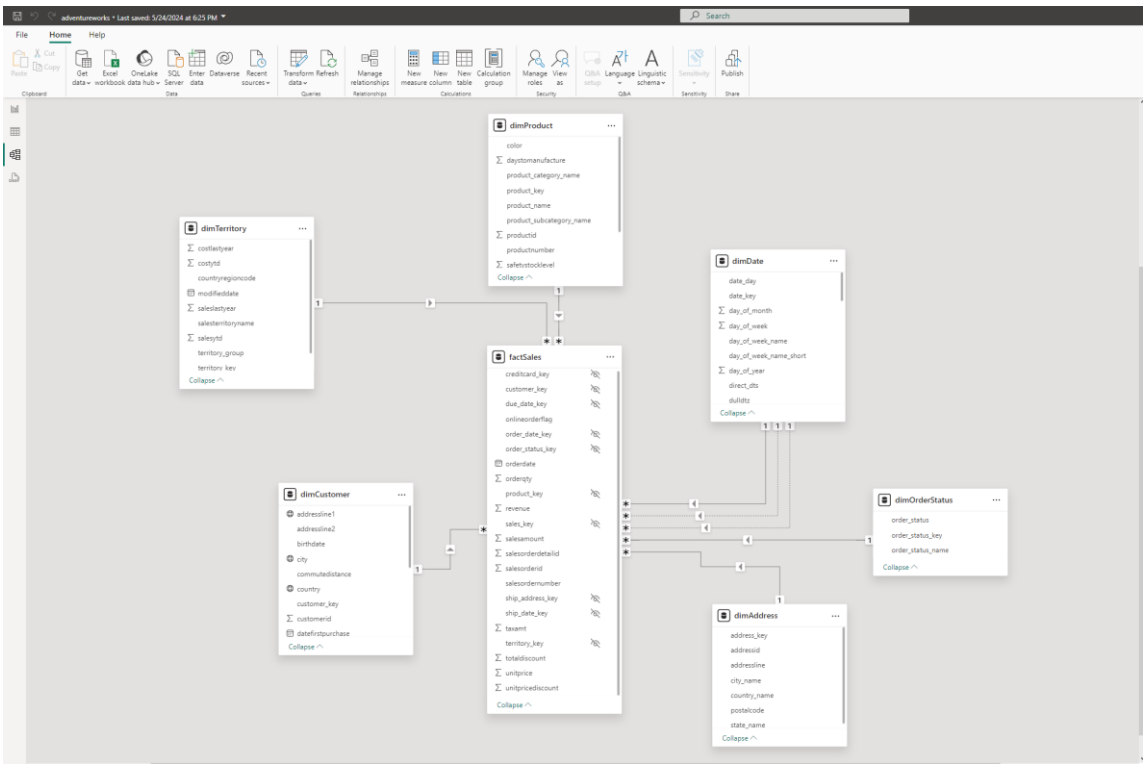


Рисунок 3.25 – ERD діаграму сховища

Тепер коли зміни успішно внесені в сховище, виникає необхідність автоматизувати процес виконання DBT команд. Тут на допомогу приходить Dagster. Задаючи розклад у Dagster, ми можемо делегувати виконання dbt команд Dagster.

Крім того Dagster забезпечує більш деталізований погляд на походження табличок і їх взаємозв'язок. Тож якщо під час виконання виникне помилка, Dagster може допомогти зрозуміти кожен крок процесу, який ми зробили в DBT. Це в свою чергу заощадить час, щоб відстежити помилку.

До того ж історія виконання dbt команд, на відмінну від ручного виконання буде збережена у логах. Що значно спрощує процес документації бага при вирішені помилки і значно зменшує його повторну появу.

Щоб почати роботу з Dagster, знаходячись у корні dbt проекту, у віртуальному середовищі потрібно встановити `dagster-dbt` і `dagster-webserver`.

Потім створити Dagster проект за допомогою команди:

```
dagster-dbt project scaffold --project-name adventureworks_dw
```

Після цього треба перейти до папки adwentureworks_dw і запустити `dagster-dev`. Це дасть доступ до веб-застосунку Dagster за посиланням «`http://127.0.0.1:3000/`».

Щоб задати розклад виконання в Dagster, треба перейти до файлу schedule.py у папці проекту Dagster та модифікувати код який був згенерований з атрібутами за замовчуванням (рисунок 3.26): задати відповідні значення для атрибутів job_name та cron_schedule.

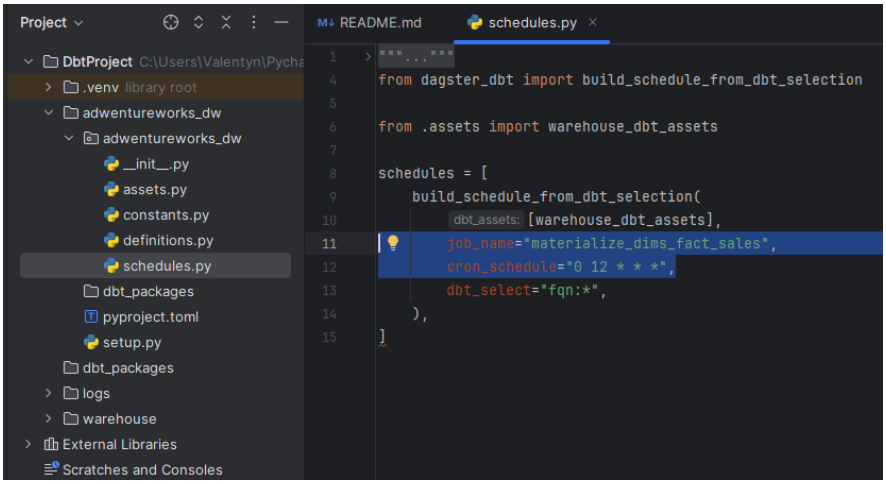


Рисунок 3.26 – Код файлу schedule.py

Розглянемо більш детально веб-застосунок Dagster. При відвідувані зазначеного посилання ми попадаємо одразу у вкладку Deployments навігації застосунка (рисунок 3.27), у шапці сторінки можемо побачити загальну інформацію: назву, джоби, останнє виконання джоби і скільки моделей матеріалізувалось. Всередині зображен граф походження і взаємозалежносте моделей. Зліва перелік моделей, а з права детальна інформація щодо конкретної моделі: запит, логи останнього виконання та інше.

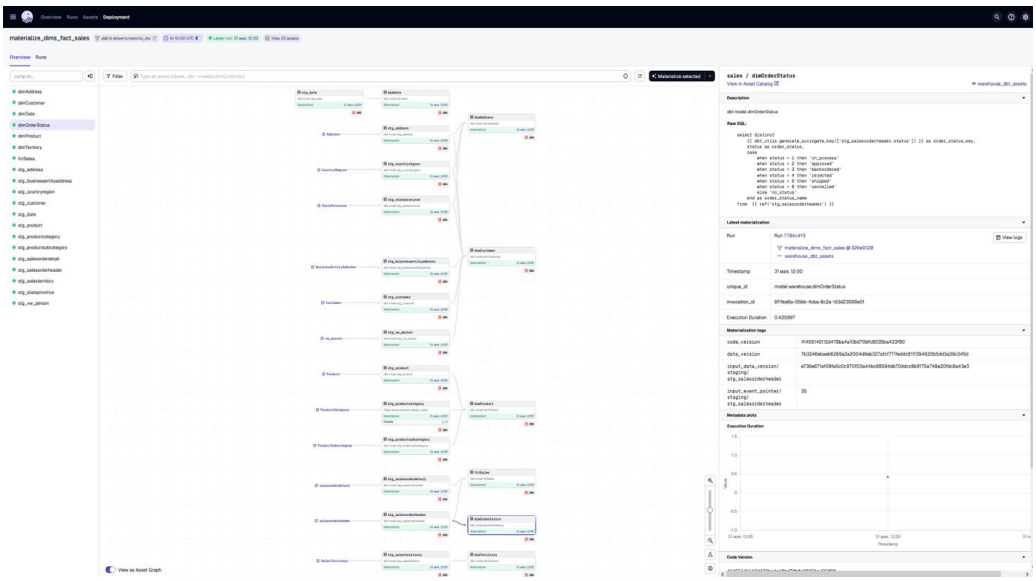


Рисунок 3.27 – Вкладка навігації Deployments застосунка у Dagster

У наступній вкладці Assets наведена вся інформація яка стосується самих моделей (рисунок 3.28): назва схеми та таблички, приналежність до відповідної групи моделей та папки в проекті, та статус і дата матеріалізації.

У слідуючій вкладці Runs наведена вся інформація виконання джоб по матеріалізації(рисунок 3.29): назва джоби, статус виконання, тривалість.

У наступній вкладці Overview наведена вся інформація про джоби (рисунок 3.30): назва джоби, розклад, сенсори, дата останнього виконання.

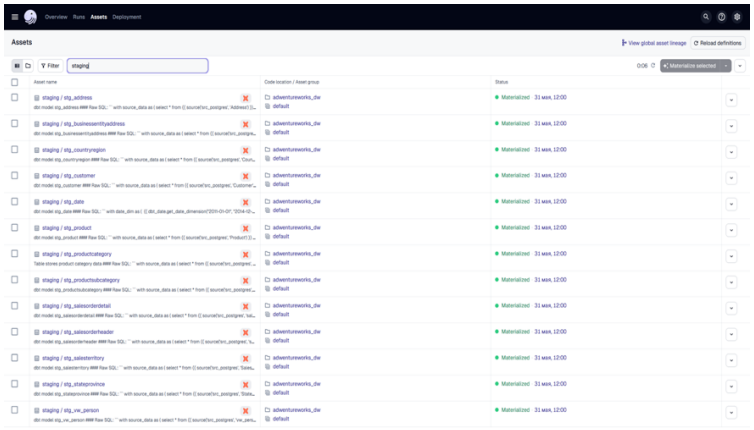


Рисунок 3.28 – Вкладка навігації Assets застосунка у Dagster

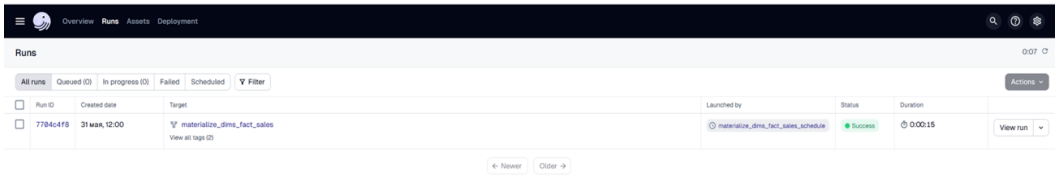


Рисунок 3.29 – Вкладка навігації Runs застосунка у Dagster

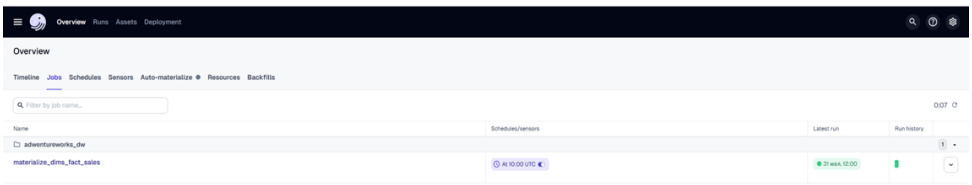


Рисунок 3.30 – Вкладка навігації Overview застосунка у Dagster

3.4 Реалізація побудови аналітичної панелі у Power BI

Перед тим як почати будувати панелі у Power BI треба визначитися як саме дані необхідні для панелей будуть підтягуватися у Power BI. Існує два шляхи: імпортувати їх або ж встановити пряме з’єднання між Power BI та джерелом даних.

В даному випадку я обрав імпорт адже він пропонує ряд переваг на прямим з’єднанням:

- відтворення панелей відбувається значно швидше, адже необхідні дані вже на стороні Power BI і нема необхідності виконувати запити до джерела даних;
- відсутність залежності від з’єднання, що дозволяє взаємодію з панелями навіть коли нема з’єднання з джерелом даних;
- нівелюється ризики створення додаткового навантаження на джерело даних;
- заплановані оновлення (наприклад, щодня, щогодини) забезпечують актуальність даних без постійних запитів;
- забезпечується управління та контроль даних на етапі імпорту, що дозволяє застосовувати узгоджені набори даних та заходи безпеки.

Далі бізнес вимоги до аналітичних панелей були перекладені на функціональні.

1) Показати наступні показники ефективності продажів:

- загальний обсяг продажів;

- кількість замовлень;

- загальна кількість проданих продуктів.

2) Показати 10 найпопулярніших продуктів на основі загальної вартості продажів.

3) Візуалізувати продажі за роками та місяцями.

4) Візуалізувати розподіл продажів за різними територіями та категоріями продуктів.

5) Візуалізувати порівняння продажів поточного року з попереднім роком.

6) Створити фільтри, які дозволяють користувачам розділяти дані для панелей за різними територіями та роками для більш детального аналізу.

Побудована аналітична панель на основі функціональних вимог наведена на рисунку 3.31.

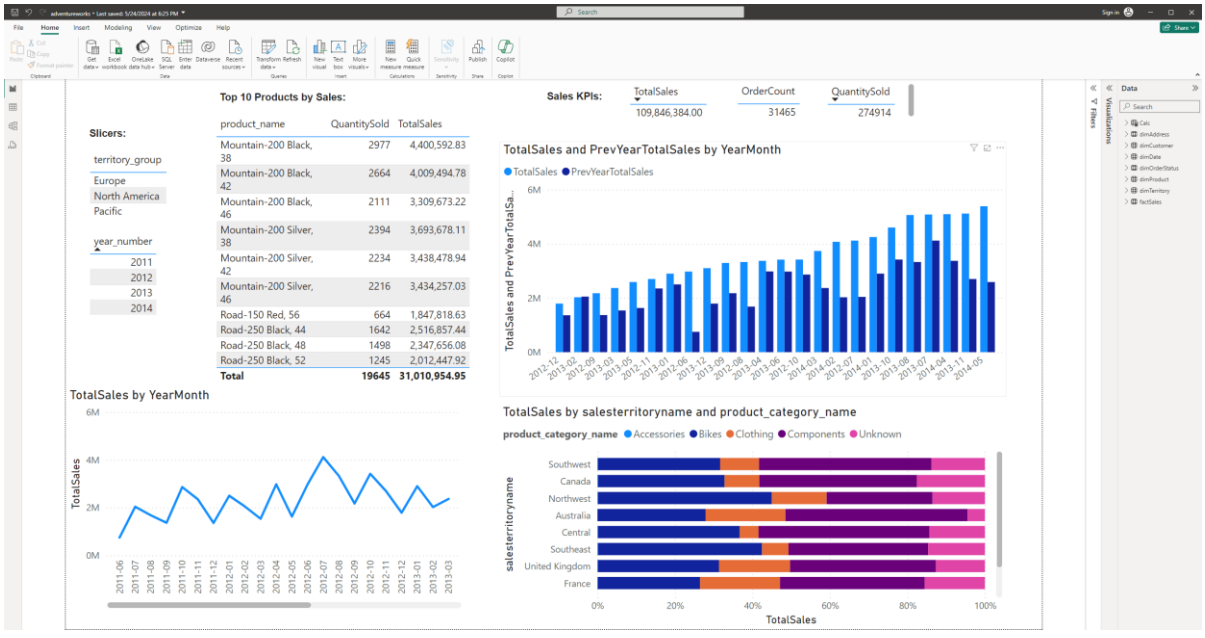


Рисунок 3.31 – Аналітична панель у Power BI

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено дослідження тенденцій методів обробки великих даних та аналіз засобів оркестрування обробки великих даних.

Також була розроблена експериментальна аналітична система з оптимізованими засобами оркестрування та продуктивними методами обробки даних. Ця система складається з трьох компонентів:

- оркестрований процес обробки поза сховищем даних;
- оркестрований процес обробки даних на стороні сховища;
- аналітичні панелі на базі даних зі сховища.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. В.Д. Залеський, П.С. Івановський, В. М. Федорченко Сучасні інструменти оркестрації даних для побудови конвеєрів автоматичної обробки даних // Системи управління, навігації та зв'язку. Збірник наукових праць. Харків, 2024. - с.95-98

2. Google файлова система [Electronic resource] – URL: https://dl.acm.org/doi/abs/10.1145/945445.945450

3. MapReduce спрощена обробка даних на великих кластерах [Electronic resource] – URL: https://dl.acm.org/doi/abs/10.1145/1327452.1327492

4. Bigtable: розподілена система зберігання структурованих даних [Electronic resource] – URL: https://dl.acm.org/doi/abs/10.1145/1365815.1365816

5. Dremel: інтерактивний аналіз наборів даних веб-масштабу [Electronic resource] – URL: https://research.google/pubs/pub36632/

6. Екосистема Hadoop [Electronic resource] – URL: https://hadoopecosystemtable.github.io/

7. Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale. Authors: Wes McKinney – 2015. – 756p.

8. Python for Data Analysis 3e: Data Wrangling with pandas, NumPy, and Jupyter. Authors: Tom White – 2022. – 550p.

9. Data Pipelines Pocket Reference: Moving and Processing Data for Analytics. Authors: James Densmore – 2021. – 274 p.

10. Офіційний сайт Spark [Electronic resource] – URL: https://spark.apache.org/

11. Learning Spark, Authors: Jules S. Damji, Brooke Wenig, Tathagata Das, and Denny Lee, July 2020. - 399p.

12. Big Data, Black Book: Covers Hadoop 2, MapReduce, Hive, YARN, Pig, R and Data Visualization 1st Edition. Authors: Editorial Services – 2016. – 300 p.

13. Big Data:Principles and best practices of scalable realtime data systems Paperback. Authors: ames Warren, Nathan Marz – 2015. – 328 p.

14. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Authors: Martin Kleppmann – 2017. – 590 p.

15. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. Authors: Ralph Kimball, Margy Ross– 2013. – 608 p.

16. Офіційний сайт DBT [Electronic resource] – URL: https://docs.getdbt.com/

17. Analytics Engineering with SQL and dbt, Authors: Rui Machado and Hélder Russa, December 2023. -324p.

18. Fundamentals of Data Engineering. Authors: Joseph Reis and Matthew Housley -2022. – 447 p.

19. Офіційний сайт Airflow [Electronic resource] – URL: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html

20. Офіційний сайт Prefect [Electronic resource] – URL: https://docs.prefect.io/latest/

21. Офіційний сайт Mage [Electronic resource] – URL: https://docs.mage.ai/introduction/overview

22. Офіційний сайт Kestra [Electronic resource] – URL: https://kestra.io/docs

23. Офіційний сайт Dagster [Electronic resource] – URL: https://docs.dagster.io/getting-started/what-why-dagster

24. Storytelling with Data: A Data Visualization Guide for Business Professionals 1st Edition. Authors: Cole Nussbaumer -2015. – 288 p.

25. The Big Book of Dashboards: Visualizing Your Data Using Real-World Business Scenarios. Authors: Steve Wexler, Jeffrey Shaffer, Andy Cotgreave- 2017. – 448p.

26. Офіційний сайт Power BI [Electronic resource] – URL: https://www.microsoft.com/en-us/power-platform/products/power-bi