Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра інтелектуальних програмних систем і технологій

Рекомендовано до захисту:
протокол засідання кафедри
№ ___ від ____.____.2025 р.
Завідувач кафедри _____
_____ О. І. Олешко
(підпис) (прізвище та ініціали)

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему «Розробка системи моніторингу ETL-процесів з використанням Java та BI додатку»

Захищено на засіданні ЕК № _____
протокол № ____ від ____.____.2025 р.
Оцінка _____ / _____
Голова ЕК
_____ _____
(підпис) (прізвище та ініціали)

Виконав:
студент 4 курсу, групи КС- 43
Спеціальності:
122 комп'ютерні науки_____
(шифр і назва спеціальності підготовки)
Гордієнко Олег Андрійович_____
(прізвище, ім'я та по батькові, підпис)
Керівник д-т кафедри, к-т економічних наук, Чуб Ольга Ігорівна_____
(ступень, звання, прізвище та ініціали, підпис)
Рецензент д-т ЗВО, к-т фізико-математичних наук, Споров Олександр Євгенович_____
(ступень, звання, прізвище та ініціали, підпис)

Харків – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

ННІ комп'ютерних наук та штучного інтелекту
Кафедра Інтелектуальних програмних систем і технологій
Рівень вищої освіти (освітньо-кваліфікаційний рівень) Бакалавр
Напрям підготовки 122 Комп'ютерні науки
Спеціальність 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О. І. Олешко

"1" березня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ (ПРОЕКТ)

Гордієнко Олег Андрійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи – Розробка системи моніторингу ETL-процесів з використанням Java та BI додатку.

керівник роботи Чуб Ольга Ігорівна, д-т кафедри, к-т економічних наук,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "16" квітня 2025 року № 4101-5/962

2. Строк подання студентом роботи "2" червня 2025 року

3. Перелік питань, які потрібно розробити:
   1) Дослідити архітектуру та функціональні можливості ETL-систем Microsoft SSIS, Informatica PowerCenter та data build tool (dbt);
   2) Дослідити їх можливість інтеграції за допомогою мови програмування Java;
   3) Розробити Java-додаток для збору метрик та моніторингу стану ETL-процесів;

4) Дослідити можливість інтеграції Microsoft SQL Server Reporting Services, Microsoft Power BI та інших за допомогою мови програмування Java;
5) Інтегрувати систему з BI-інструментами для візуалізації даних;
6) Провести тестування системи на реальних даних та оцінити її ефективність.

4. План роботи

№ з/п	Назви етапів роботи
1.	Вивчення основних характеристик та можливостей Microsoft SQL Server Integration Services (SSIS), Informatica PowerCenter та data build tool (dbt)
2.	Порівняльний аналіз цих систем за критеріями підтримки API, можливості збору метрик, легкості інтеграції з Java та ліцензійних обмежень.
3.	Вибір ETL-системи на основі результатів аналізу
4.	Дослідження наявних API для вибраної ETL-системи (SSIS, Informatica PowerCenter або data build tool).
5.	Оцінка сумісності API з мовою Java, включаючи перевірку наявності Java-бібліотек або SDK.
6.	Визначення методів збору метрик через API, таких як час виконання задач, помилки та кількість оброблених даних.
7.	Проектування архітектури Java-додатку, включаючи модулі збору даних, обробки та зберігання.
8.	Реалізація модуля збору даних з інтеграцією з API вибраної ETL-системи.
9.	Дослідження можливостей BI-інструментів (Microsoft SQL Server Reporting Services, Microsoft Power BI або інших бібліотек).
10.	Вибір BI-інструменту на основі легкості інтеграції з Java-додатком, можливості візуалізації даних у реальному часі та ліцензійних обмежень.
11.	Створення звітів та дашбордів для візуалізації ключових показників.
12.	Підготовка тестових даних шляхом створення тестових ETL-процесів у вибраній системі.
13.	Перевірка коректності збору, обробки та візуалізації даних.
14.	Запуск системи моніторингу та збір метрик протягом певного періоду.

5. Дата видачі завдання 15 лютого 2025 року

Студент  О.А. Гордієнко

Керівник роботи  О.І. Чуб

АНОТАЦІЯ

Ця кваліфікаційна робота, для здобуття ступеня бакалавра, присвячена дослідженню та впровадженню засобів автоматизованого створення PDF BI-звітів на основі результатів запусків ETL-систем. У рамках дослідження було проаналізовано сучасні підходи до побудови ETL-процесів, використання систем бізнес-аналітики, а також реалізовано прикладну програму, яка отримує інформацію про запуски ETL-завдань та формує аналітичні звіти у форматі PDF.

Предметом роботи є методи та засоби автоматизованого формування PDF BI-звітів на основі результатів запусків ETL-систем з використанням dbt, Snowflake та Java.

Об’єктом роботи є процес візуалізації та подання результатів виконання ETL-процесів у системах бізнес-аналітики.

Метою роботи є дослідження підходів до побудови звітності на основі результатів ETL-процесів та реалізація програмного рішення для створення PDF BI-звітів.

В ході розробки було проведено такі етапи:

- аналіз інструментів для реалізації ETL та побудови звітів;
- створення бази даних у Google BigQuery;
- розробка ETL-процесів за допомогою dbt;
- реалізація Java-додатку для отримання інформації через API;
- створення PDF BI-звітів на основі отриманих даних.

Обсяг роботи становить 61 сторінку, містить 12 рисунків, 3 таблиці, 9 формул та посилання на 24 джерела.

Ключові слова: ETL, BI-ЗВІТ, DBT, BIGQUERY, JAVA, PDF, API, АВТОМАТИЗАЦІЯ.

ABSTRACT

This bachelor's qualification thesis is dedicated to the research and implementation of tools for the automated creation of PDF BI reports based on the results of ETL system executions. As part of the research, modern approaches to building ETL processes and using business intelligence systems were analyzed. In addition, a software application was developed that retrieves information about ETL job runs and generates analytical reports in PDF format.

The subject of the thesis is the methods and tools for automated generation of PDF BI reports based on the results of ETL system executions using dbt, Snowflake, and Java.

The object of the thesis is the process of visualizing and presenting the results of ETL executions in business intelligence systems.

The aim of the thesis is to explore approaches to reporting based on ETL process results and to implement a software solution for generating PDF BI reports.

The development process included the following stages:

- analysis of tools for implementing ETL and reporting;
- creation of a database in Google BigQuery;
- development of ETL processes using dbt;
- implementation of a Java application to retrieve data via API;
- generation of PDF BI reports based on the retrieved data.

The thesis consists of 60 pages, includes 12 figures, 3 tables, 9 formulas, and references to 24 sources.

Keywords: ETL, BI REPORT, DBT, BIGQUERY, JAVA, PDF, API, AUTOMATION.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ETL – Extract, Transform, Load.

ELT – Extract, Load, Transform.

BI – Business Intelligence.

DWH – Data Warehouse

dbt – data build tool.

SSIS – SQL Server Integration Services.

API – Application Programming.

CSV – Comma-Separated Values.

SQL – Structured Query Language.

JDBC – Java Database Connectivity.

JSON – JavaScript Object Notation.

HTTP – HyperText Transfer Protocol.

REST – Representational State Transfer.

SDK – Software Development Kit.

DAG – Directed Acyclic Graph.

GCP – Google Cloud Platform.

Snowflake – хмарна платформа для зберігання та аналізу даних.

Power BI – система бізнес-аналітики від Microsoft.

Tableau – інструмент візуалізації та аналізу даних.

Matillion – хмарний ETL-інструмент.

CI/CD – Continuous Integration / Continuous Deployment.

Dataiku – платформа для розробки аналітичних рішень.

Alteryx – інструмент підготовки даних для аналітики.

Databricks – платформа для обробки великих даних на базі Apache Spark.

Redshift – аналітичне сховище даних від Amazon.

BigQuery – аналітична служба хмарної платформи Google.

YAML – Yet Another Markup Language – мова розмітки для конфігурацій.

ВСТУП

У сучасну епоху, що характеризується поширенням даних, організації стикаються з необхідністю не лише зберігати інформацію, але й ефективно обробляти, інтегрувати та аналізувати її. В аналітичних системах щодня виконується значна кількість процесів трансформації даних, які є основою для прийняття критично важливих бізнес-рішень. За таких умов особливої актуальності набувають інструменти, які полегшують організацію, автоматизацію та документування процесів обробки даних. Серед таких інструментів особливе місце займають системи ETL (Extract, Transform, Load) та ELT (Extract, Load, Transform).

Історично ETL-процеси використовувалися в традиційних сховищах даних, де дані піддавалися трансформації перед завантаженням в цільове середовище. Поява хмарних технологій і розвиток обчислювальних платформ породили попит на нові підходи, такі як модель ELT, де трансформація відбувається в самому сховищі. Ефективність кожного підходу залежить від особливостей бізнесу, інфраструктури, обсягу та типу даних. Вибір одного підходу над іншим визначається багатогранною оцінкою цих факторів.

Одночасно з еволюцією систем трансформації роль метаданих розширилася, охопивши інформацію про процеси обробки, час виконання, залежності між етапами, статуси та помилки. Метадані відіграють ключову роль у моніторингу, оптимізації, автоматизації та аудиті конвеєрів ETL/ELT. Ефективне використання цих інструментів дає змогу всебічно зрозуміти механізми, що лежать в основі інтеграційних процесів. Крім того, ці інструменти полегшують виявлення аномалій, моніторинг продуктивності системи та підтримку якості даних.

Серед сучасних інструментів для побудови ETL/ELT процесів особливе місце займає dbt (data build tool). Цей гнучкий фреймворк призначений для розробників SQL і полегшує побудову прозорої, модульної, параметризованої інфраструктури для обробки даних. Його зростаючу популярність можна

пояснити можливостями тестування, автоматичного документування, роботи з DAG (графами залежностей) та підтримкою хмарних рішень, таких як Google BigQuery.

Ця дипломна робота була присвячена розробці моделей в dbt та збору метаданих про ці процеси за допомогою dbt Cloud API. Також було розглянуто обробку результатів та візуалізацію цих результатів у вигляді автоматично згенерованих PDF BI звітів. Для цього було реалізовано Java-додаток з графічним інтерфейсом, а також систему генерації звітів на основі JasperReports. Ця система дозволяє користувачам зручно переглядати інформацію про статус завдань, моделі, типи матеріалізацій та інші ключові параметри прогонів.

Таким чином, робота інтегрує декілька важливих компонентів сучасної обробки даних, включаючи побудову ETL/ELT архітектури, управління метаданими, інтеграцію з BI-середовищами та автоматизацію звітності.

Предметом дослідження є методи та засоби автоматизованого формування PDF BI-звітів на основі результатів запусків ETL-систем із використанням dbt, Google BigQuery та Java.

Об’єктом дослідження є процес візуалізації та подання результатів виконання ETL-процесів у системах бізнес-аналітики.

Метою роботи є дослідження підходів до побудови звітності на основі результатів ETL-процесів та реалізація програмного рішення для створення PDF BI-звітів.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Сьогодні майже кожна приватна компанія зберігає десь свої дані. Щороку різноманітність і різновид таких форматів даних лише зростає і ускладнюється. Однак, якщо ми збираємося проводити будь-який аналіз цих даних, є ще деякі проблеми, які нам потрібно вирішити: недостатньо просто працювати з даними - ми повинні їх інтерпретувати, об'єднувати з різних місць та перевіряти якість процесу.

Через це існують інструменти, такі як системи ETL (Extract, Transform, Load) або ELT (Extract, Load, Transform), які створюються для переміщення, перетворення та завантаження величезних обсягів даних у місце для аналітичних запитів, як-от ваше Сховище даних або Lakehouse. Такі рішення, як Informatica PowerCenter, Microsoft SSIS, Talend, Apache NiFi, мають давно існуючі продукти, призначені для роботи в такий спосіб, тоді як dbt і Matillion є більш сучасними хмарними рішеннями, наприклад.

Але процес трансформації є лише одним елементом. Іноді навіть важливіше знати, що саме відбулося на всьому шляху. Ось чому метадані (інформація про процес ETL) настільки важливі: коли востаннє виконувалось завдання ETL, скільки часу це зайняло, скільки рядків було оброблено, чи завершилося воно успішно, які ресурси були використані...

Це стосується і PowerCenter, де вся ця інформація є та розподілена по різним технічним таблицям, і Microsoft SSIS, який може зберігати їх на SSISDB, логах. dbt генерує документацію метаданих автоматично, що є корисним для розробників і аналітиків.

Проте, коли всі ці дані зібрані, виникає інша проблема: вони часто зберігаються в роз'єднаних системах. Деякі дані зберігаються в серверних журналах, деякі поміщені в бази моніторингу, а деякі мають своє місце у Grafana або Kibana. Через це важко отримати повне уявлення про те, як функціонує конвеєр ETL/ELT. У великих міжнародних організаціях,

наприклад, може бути більше 100 таких процесів на будь-який день, що унеможливлює масштабування в усіх напрямах. "О, до речі", точність даних, ймовірно, страждає - неправильні значення метрик, затримки або вплив на бізнес (неправильні дані доставлені, звіти запізнюються, рішення не прийнято через неправильну інформацію).

Реальний випадок використання демонструє цінність поєднання метаданих і бізнес-аналітики. Аналітики й технічний персонал зможуть взяти метрики продуктивності й просто заявити: «о, ось де я завжди затримуюсь», «гей, ось машина, яка завжди нас затримує», «цікаво, чому ми маємо більше навантаження, ніж раніше». Це не тільки дозволяє їм швидко реагувати на виникаючі проблеми, але й сприяє оптимізації ресурсів. Це також дає організації уявлення про те, наскільки дійсними та корисними є дані, які вводяться в її системи прийняття рішень.

Нарешті, використання метаданих - це не просто технічна приємність, а суттєва необхідність для організацій у сфері управління їхніми даними, особливо для аналітики і бізнесу в цілому.

1.2 Принципи роботи ETL/ELT систем

Ефективність і гнучкість роботи з інформацією залежить від архітектури систем ETL і ELT.

ETL (Extract, Transform, Load) розбиває вихідні дані і потім виконує перетворення згідно з бізнес-правилами. Цільове зберігання є кінцевим пунктом призначення для всіх оброблених даних на цьому етапі. У випадку, коли перетворення відбувається перед завантаженням, ця модель часто використовується в класичних сховищах даних.

Зараз залишили дані, щоб їх можна було підготувати для аналізу заздалегідь. Однак із збільшенням обсягів даних швидкість обробки різко падає. Для великих обсягів даних вузьке місце виникає не на стороні джерела, а в цьому проміжному просторі.

ELT змінює цей підхід. Вона переміщує продуктивність і змінює все.

Замість того щоб виконувати перетворення після того, як сирі дані вже були поміщені в сховище - дуже трудомісткий процес, який може значно затримати аналітику - перетворення не викликають складнощів у світі, яким характеризується етап один: натомість, ELT (Extract, Load, Transform) змінює послідовність, спершу завантажуючи сирі дані в цільове сховище, а потім виконуючи перетворення. Цей шаблон особливо добре працює з новими хмарними середовищами, в яких обчислювальні ресурси можуть бути швидко пристосовані. Snowflake, BigQuery і Databricks - це хмарні сховища даних, які виконують складні перетворення без шкоди для якості даних. ELT використовує дані такими, якими вони є в природі, що означає, що легше адаптувати зміни в джерелах. Наприклад, перетворення відбуваються тільки тоді, коли це потрібно - для конкретних аналітичних запитів, таких як ті, що використовуються для фінансових даних. Хоча ETL вимагає чіткої визначеності правил перетворення на ранньому етапі, ELT дозволяє відкласти цей процес.

Системи ETL зазвичай використовуються в організаціях з чіткими схемами даних, такими як Informatica PowerCenter і Microsoft SSIS. Перед будь-яким введенням даних у систему ці системи перевіряють їх узгодженість і здійснюють валідацію інформації. Інструменти ELT набувають популярності в середовищах, де важливі швидкість і масштабованість завантажень, а бізнес-правила можуть змінюватися, таких як dbt, Fivetran або Matillion.

Процеси ETL/ELT часто включають деяку реструктуризацію даних. Більшість цих операцій виконують спеціальну роботу для підготовки інформації до аналізу. Найпоширеніший вид виконують очищувальні операції даних, такі як корекція форматів на півдорозі; позбавлення від дублікатів у записах; заповнення відсутніх значень, де це можливо; і узгодження зберігання вмісту кожного запису - цей останній крок є абсолютно критичним для забезпечення якості даних. Третя категорія, агрегаційні перетворення, включає підсумовування або усереднення, а також групування записів. Це підсумовує "новини" з детальних записів у підсумкові метрики, необхідні для

аналітики. З'єднувальні перетворення інтегрують дані з різних джерел шляхом комбінування таблиць за ключовими полями. Наприклад, це критично для створення єдиного джерела правди. Перетворення валідації шукають дані, які не відповідають як бізнес-правилам, так і технічним вимогам. Вони знаходять аномалії та помилки.

Спеціальна категорія - це бізнес-перетворення. Вони виконують доменно специфічну логіку, необхідну для різних професій — від обчислення фінансових індикаторів до формування складних вимірів, що використовуються для аналітичних цілей. Також сучасні системи дедалі більше містять трансформації збагачення даних (наприклад, геокодування, категоризація або машинні прогнози), що значно розширює можливий спектр аналітики. Кожен вид перетворення має свої вимоги до впровадження, а також тестування, моніторинг тощо.

Як правильно поєднати всі ці різні перетворення, важливого значення набуває побудова ефективних каналів обробки даних. Обидва підходи мають свої сильні та слабкі сторони. ETL краще підходить для складних перетворень, які вимагають високого рівня контролю якості даних до їх завантаження в сховище. З іншого боку, ELT є більш ефективним в обробці великих обсягів неструктурованих або напівструктурованих даних, оскільки він може застосовувати сире обчислювальне потужність у сучасних DWH та "озерах даних".

Вибір між ETL і ELT залежить від реалістичних вимог до даних, інфраструктури бізнесу та його потреб для аналізу. Сьогоднішні платформи дедалі частіше пропонують змішані підходи, які об'єднують обидва для досягнення більшої гнучкості.

Тут метадані, пов'язані з роботою системи Illume Premier і інтеграції даних Illume (SQL), не лише мають велике значення, але й є життєво важливими. Переваги від усіх цих даних включають: запам'ятовування, яке є джерело даних; збереження даних щодо виконаних на даних перетворень; коли були оброблені; і будь-які помилки, що сталися. Адміністрування

продуктивності, аудит змін та оптимізація процесів базуються на цих даних.

Незалежно від методології, детальне збирання та аналіз метаданих - чи то ETL, чи ELT - дозволять підприємствам впевнено мандрувати в невідоме. Це дозволяє швидко виявляти аномалії; забезпечувати стабільність інтеграційних процесів з часом.

1.3 Принципи роботи BI систем

Система BI використовується для збору сировинних даних, їх зміни, інтерпретації інформації, отриманої з цих даних, у звіт чи графік або інше. Системи, такі як Powerdata або Tableau, стають зараз також додатками для самостійного аналізу. Приємно чути, що аналітики, менеджери та звичайні користувачі можуть легко отримати те, що їм потрібно, з будь-якого джерела. Потреба у високій кваліфікації зникає. Сучасні рішення в області бізнес-аналітики справді орієнтовані на потреби звичайних користувачів.

Принципи роботи BI систем ґрунтуються на їх здатності трансформувати сирі дані в цінні інсайти, які допомагають у прийнятті управлінських рішень. Основна ідея полягає в тому, щоб зібрати дані з різних джерел, обробити їх, проаналізувати та наочно представити у вигляді звітів, дашбордів або інтерактивних візуалізацій. Сучасні BI-системи, такі як Power BI, Tableau або Qlik, працюють за принципом самодостатньої аналітики, де користувачі різних рівнів – від аналітиків до топ-менеджерів – можуть самостійно досліджувати дані без глибоких технічних знань. Ключовою особливістю сучасних BI-інструментів є їхня орієнтація на бізнес-користувача, що досягається за рахунок інтуїтивних інтерфейсів, можливості швидкого створення звітів та гнучкої настройки візуалізацій.

Ви очікуєте отримати багато вигоди від цієї нової домовленості! Усі мають інтерфейс, який легко використовувати, що дозволяє швидко запускати та переформатовувати звіти.

Налаштування обробки даних у внутрішньому вузлі визначає, як буде виконуватися цей вузол. Вони містять з'єднання з різними типами даних,

такими як бази даних, хмарні сховища та формати файлів, які не підтримуються нативно. Привабливість сучасних систем бізнес-аналітики полягає в тому, що ви можете імпортувати дані прямо з ваших улюблених додатків, тому немає потреби працювати з ручним введенням.

Як обробляються дані в системі BI? Спочатку ви отримуєте свої дані та видаляєте будь-які неточності. Потім ви встановлюєте модель із пов'язаними таблицями. Після цього ви обчислюєте метрики. Нарешті створюється інтерактивна графіка. Ми приділяємо велике значення якості даних, розуміючи, що будь-яка помилка в підготовці знижує цінність цих даних під час аналізу.

Нове покоління систем BI стає розумнішим. Вони застосовують технології автоматизації для аналітичної обробки. Бізнес-аналітика розвивається таким чином, з функціями, такими як автоматичне виявлення та запити на природній мові. Порівняно з традиційними підходами, це величезний прогрес. Саме звідси походить термін «вбудованої аналітики». Функції BI включені в бізнес-додатки, що полегшує впровадження аналітики прямо у ваше робоче середовище чи будь-що інше, чим ви займаєтеся в програмному забезпеченні!

Мобільна аналітика також розвивається, і це чудова новина! Тепер можна виводити звіти та панелі приладів на будь-який планшет. Це особливо корисно для менеджерів, які використовують їх для прийняття рішень на ходу. Це чудово бачити, що все більше систем BI сьогодні роблять аналітику доступною для всіх в межах організації. Дуже приємно, що сьогодні в бізнесі відбувається демократизація даних. Ця тенденція дозволяє людям з будь-яким рівнем освіти і сертифікації отримувати необхідну інформацію.

Більш того, ці інструменти дійсно прості у використанні! Завдяки інструментам самостійного BI ви можете відразу створювати звіти та отримувати доступ до даних без необхідності чекати, поки хтось інший це зробить за вас.

Тим часом з'явилася ідея «керованої аналітики». Тому ми можемо

об'єднати наші джерела даних, метрики та правила обробки. Це дозволяє всім отримувати стандартизовані аналітичні результати. На щастя, ми можемо сказати, що системи, які використовуються для бізнес-інтелекту сьогодні, підходять для будь-яких смаків. «Кожен приймаючий рішення, хто шукає гнучкості чи намагається зберегти якість даних, повинен відчувати радість, якщо ми скажемо їм, що ці інструменти дійсно необхідні для прийняття бізнес-рішень», — відповів експерт галузі в Державному архіві Китаю.

1.4 Можливості інтеграції ETL/ELT систем з BI системами

Дуже важливою частиною сучасної архітектури даних є можливість підключення систем ETL/ELT до систем бізнес-аналітики. Важко переоцінити важливість такого зв'язку між необробленими даними та справді цінною бізнес-інформацією. Найбільш поширеними методами інтеграції сучасних інструментів ETL/ELT, таких як Informatica PowerCenter, Microsoft SSIS, Talend, dbt тощо, є використання можливостей BI-платформ. Аналогічно, традиційними способами є вивантаження трансформованих даних в окреме сховище (наприклад, SQL Server Analysis Services для Power BI, Snowflake для Tableau). На відміну від цих методів, деякі більш сучасні технології інтегруються безпосередньо з BI-системою на рівні API-з'єднання або навіть у вигляді потоку даних. Основною характеристикою сучасної інтеграції є двостороння взаємодія. BI-системи не тільки приймають готовий результат даних, але й можуть диктувати ETL/ELT-процесам через спеціальні інтерфейси, наприклад, встановлювати час оновлення та вибирати за ключовими словами певні набори даних для подальшої обробки.

Важливою частиною інтеграції є надсилання не лише самих даних, але й пов'язаних з ними метаданих. Саме це допомагає BI-системам краще розуміти як структуру інформації, так і її бізнес-контекст. У сучасних платформах, таких як Alteryx або Dataiku, є поняття «родовід даних». Значення для BI-звітів полягає в тому, що кожен показник можна простежити по конвеєру, що включає безліч ETL/ELT-трансформацій, до його початкових

джерел. Це значно підвищує надійність аналітики. Спеціальні шаблони з'єднань розробляються для максимізації ефективності інтеграції з урахуванням сильних сторін конкретних інструментів ETL/ELT і BI. Прикладами тут можуть слугувати спрощені коннектори для Tableau до Snowflake або вставка Power Query в Power BI для роботи з даними після їх обробки в Azure Data Factory.

У сучасному ландшафті бізнес-аналітики існує окрема категорія рішень для інтеграції ETL/ELT з BI, які працюють у режимі реального часу. Конфігурації робочих процесів ETL/ELT постійно передають дані в BI-системи через механізми потокової обробки, такі як Kafka або Azure Event Hubs. Крім того, такі архітектури особливо актуальні для сценаріїв операційної аналітики, таких як моніторинг фінансових транзакцій та аналіз поведінки клієнтів у цифрових каналах. BI як послуга почала об'єднуватися. Робочі процеси ETL/ELT та аналітичні звіти надаються як єдиний керований сервіс, що дозволяє організаціям зосередитися на операціях з даними, а не на обслуговуванні інфраструктури. У той же час, забезпечення безпеки даних під час інтеграції, особливо в гібридних середовищах, залишається складним завданням, яке вимагає вдосконалених механізмів аутентифікації, шифрування та контролю доступу на всіх етапах передачі інформації між системами.

Сучасні тенденції інтеграції ETL/ELT з BI спрямовані на подальше спрощення та більшу автоматизацію. Зокрема, набуває популярності ідея «нульового ETL» - дані тепер можна аналізувати без їх явного перетворення: це відбувається завдяки віртуалізації даних і розширеним функціям сучасних DWH. Водночас з'являються гібридні середовища, в яких прості трансформації виконуються безпосередньо в BI-інструменті (наприклад, за допомогою DAX в Power BI або LOD-виразів в Tableau), а складні бізнес-правила реалізуються на рівні ETL/ELT-процесів. Попри це, важливо також зазначити, що інтеграція з системами управління метаданими (такими як Collibra та Informatica Metadata Manager) також є важливим напрямком, що

дозволяє автоматизувати документування потоків даних та забезпечити відповідність регуляторним вимогам. Все це робить об'єднання ETL/ELT з BI-системами ключовою ланкою в ланцюжку створення цінності даних, де можливості технічної трансформації та потужні інструменти візуалізації зливаються в одне ціле для підтримки прийняття рішень.

1.5 Аналіз програмних існуючих рішень

Однією з найважливіших складових сучасної архітектури даних для інтеграції систем ETL/ELT з BI-системами. Цей зв'язок дає нам змогу пройти шлях від необроблених даних до цінних бізнес-інсайтів. Сучасні інструменти ETL/ELT, такі як Informatica PowerCenter, Microsoft SSIS, Talend або dbt, мають різні методи інтеграції з платформами BI. Вони варіюються від традиційного підходу вивантаження перетворених даних в оптимізовані сховища (наприклад, SQL Server Analysis Services для Power BI або Snowflake для Tableau) до більш сучасних реалізацій, в яких вбудована пряма підтримка API-з'єднання та потокової передачі даних. Якщо обидві системи сумісні і добре синхронізовані, сучасна інтеграція робить можливим двосторонню роботу. BI-система може як отримувати консервовані дані, так і використовувати спеціальні інтерфейси, щоб дати ETL/ELT орієнтири для процесу ETL/ELT. Наприклад, вона може вказати час оновлення і вибрати точні набори даних для подальшої обробки. Невід'ємною частиною є передача не лише самих даних, але й метаданих, що їх супроводжують. Це дозволяє BI-системам краще розуміти структуру інформації та бізнес-семантику. У сучасних платформах, таких як Alteryx або Dataiku, ми реалізуємо концепцію «data lineage». Це означає, що будь-який показник в BI-звіті можна явно простежити до його джерела за допомогою серії ETL/ELT-обробок, що значно підвищує достовірність аналітики. Щоб максимізувати ефективність інтеграції, ми розробляємо специфічні шаблони з'єднувачів, які враховують як конкретні функції ETL/ELT, так і BI-інструментів. До них відносяться оптимізовані коннектори для Tableau до Snowflake, функції потужних запитів,

вбудовані в Power BI, працюють з даними після їх обробки в Azure Data Factory.

Іншу категорію цього типу складають рішення для інтеграції в режимі реального часу. Наприклад, процеси ETL/ELT часто налаштовуються для передачі даних в аналітичну систему через механізми безперервного потоку (наприклад, Kafka і Azure Event Hubs). Процедури для такого типу архітектури є цікавим викликом. Робота з даними в режимі реального часу (а хіба це може бути простіше?) підвищує технічний рівень кожного типу завдань, від створення баз даних до запуску звітів за запитами. Особливо в останньому випадку необхідно подбати про те, щоб будь-яка значна затримка була якомога меншою. Архітектурні питання тут пов'язані з рішенням про найбільш підходящий спосіб інтеграції BI-систем з реальними цільовими сховищами даних: чи будуть це прямі посилання (які дозволяють оновлення окремих таблиць негайно з'являтися в аналітичних звітах без будь-яких проміжних кроків у джерелі), додавання проміжних рівнів API або використання спеціалізованих сервісів для агрегованих метаданих. Синхронізація даних у розподілених системах може стати справжнім кошмаром. Процеси ETL виконуються в системах, що знаходяться на різних платформах (локальна мережа і хмара), і BI-система повинна переконатися, що дані переглядаються в єдиному контексті незалежно від їх фізичного розташування. Один інженер якось сказав мені, що проблем, які виникають через те, що різні підрозділи організації працюють окремо, безліч. Є інженери з обробки даних, які виконують ETL-процеси, аналітики, які створюють BI-звіти, і бізнес-користувачі (Елізі краще замінити цей термін пізніше), яким потрібні актуальні ваші емпіричні дані для прийняття рішень (щоб вони могли діяти). Відсутність стандартної документації для ETL-діяльності в BI-системах є поєднанням помилок в інтерпретації, труднощів для операторів і плутанини щодо того, які процедури насправді необхідні на яких етапах. Питання безпеки даних - з точки зору контролю доступу та приховування конфіденційної інформації - повинні бути ретельно продумані аж до таких деталей, як

взаємозв'язок різних систем між собою.

Для того, щоб забезпечити високу доступність і надійність системи, процес відновлення її відмовостійкості представляє значну складність. Збої в робочому процесі ETL не можна ігнорувати. Це невидимо для тих, хто бачить лише кінцевий продукт BI, і вимагає, щоб неточні дані, поширені в аналітичних звітах, були негайно вилучені та видаленіДля вирішення цих проблем необхідно побудувати складні системи процесу перевірки даних; необхідно скласти схему зв'язків даних; і вам також доведеться налаштувати систему раннього попередження про будь-яку аномаліюУсі ці елементи в сукупності роблять інтеграцію BI-систем в ETL складною, комплексною та багатовимірною роботою. Вона вимагає глибокого розуміння обох типів систем і загальної чіткої стратегії управління даними для всієї організації.

2 АНАЛІЗ ІНСТРУМЕНТІВ ТА СЕРЕДОВИЩ ДЛЯ ПОБУДОВИ ETL/ELT-СИСТЕМ І ЗВІТНОСТІ

2.1 Порівняльний аналіз ETL/ELT систем

Відповідно до ключових критеріїв, Informatica PowerCenter, Microsoft SSIS та dbt мають дещо різні підходи до управління метаданими та можливостей інтеграції. Це робить dbt потенційно чудовим вибором для сучасних завдань аналізу. Що стосується підтримки API, Informatica PowerCenter надає Rest API через Informatica Cloud. Його функціональність, однак, обмежена елементарними операціями простого моніторингу та управління. На противагу цьому, Microsoft SSIS взагалі не має власного REST API.

Для програмного доступу до метаданих він повинен покладатися на агента сервера sql або дозволяти розширення. Але dbt, здається, був розроблений з менталітетом «першого API». Його повнофункціональний REST API, здається, дозволяє не тільки зчитувати детальні показники продуктивності, але й керувати всіма роботоподібними аспектами для автоматизації за допомогою комп'ютерного програмування. Тут відкриваються широкі можливості для взаємодії із зовнішніми системами. Що стосується збору показників продуктивності, то Informatica PowerCenter і Microsoft SSIS надають дуже базові можливості через свої власні системи моніторингу. Тим не менш, слід зазначити, що вони мають справу зовсім не з наддеталізованими аналітичними даними, а зі звичайними показниками продуктивності. Кастомізація додає більше можливостей для отримання більш глибоких даних. DBT, з іншого боку, має вбудовану підсистему метрик, яка автоматично отримує детальну інформацію про кожен цикл.

Ця інформація включає в себе час, необхідний для виконання окремих моделей, об'єм даних, який було оброблено, і такі речі, як, наприклад, розтягування країв від одного завдання до іншого в залежному ланцюжку. Наскільки я розумію, всі ці дані надаються не тільки тим системам, які можуть

отримати їх через API, але й у форматі, сумісному з аналітичними системами. Схоже, що DBT має потенціал стати хорошим місцем для комплексної звітності про пропускну здатність ETL-процесів. І перевагою є відсутність додаткового навантаження на розробку контенту для моніторингу поверх них. Java-система через стандартні HTTP-клієнти або спеціально розроблені бібліотеки.

Крім того, архітектура DBT як інфраструктури на основі коду, ймовірно, дозволить дуже легко вбудовувати її в Java-додатки для контрольованого виконання перетворень на рівні командного рядка. Ліцензування може бути ще однією ключовою сферою, де DBT має перевагу над традиційними рішеннями. Наприклад, Informatica PowerCenter відома своєю високою вартістю ліцензії, а також складними методами виставлення рахунків. З іншого боку, хоча Microsoft SSIS поставляється з SQL Server, він має обмеження щодо масштабування і потребує додаткових ліцензій для розподілених сценаріїв.

DBT також пропонує модель ліцензування, яка зазвичай вважається більш гнучкою, з ядром компонентів з відкритим кодом (dbt Core) та комерційними пропозиціями в dbt Cloud. Ця модель дозволяє компаніям починати з невеликих проектів і масштабуватися, не збільшуючи при цьому свої витрати. Відкритість DBT може вирішити деякі проблеми, пов'язані з прив'язкою до певного постачальника, які можуть бути пов'язані з рішеннями Informatica та Microsoft. Це може дати підприємствам більшу гнучкість у виборі систем та інтеграційних рішень. В сукупності ці фактори вказують на те, що DBT може бути найкращим вибором для сучасних проектів, особливо якщо проект вимагає гнучкості, зростання та повної інтеграції в існуючу інфраструктуру Java.

Активна спільнота користувачів та широкий вибір плагінів і розширень забезпечують швидке вирішення проблем та обмін досвідом. Постійні оновлення та розвиток dbt Core свідчать про стабільну підтримку проєкту на довгострокову перспективу.

Таблиця 2.1 – Microsoft SSIS vs dbt

Критерій	Microsoft SSIS	dbt
Підтримка API	Обмежена (через SQL Server Agent або власні розширення)	Повноцінний REST API (Cloud) + CLI (Core)
Збір метрик	Базовий моніторинг через SSISDB, потрібні додаткові налаштування для аналітики	Вбудована система метрик (час виконання, залежності моделей, автоматичні звіти)
Інтеграція з Java	Складність через .NET-орієнтовану архітектуру	Гнучка через REST API та офіційні клієнтські бібліотеки
Ліцензійні умови	Вимагає ліцензії SQL Server (дорого для масштабування)	Відкрите ядро (dbt Core) + комерційні опції (Cloud)

Таблиця 2.2 – Informatica PowerCenter vs dbt

Критерій	Informatica PowerCenter	dbt
Підтримка API	Обмежений REST API (через Informatica Cloud)	Повноцінний REST API (документовані endpoints для jobs, runs, environments)
Збір метрик	Власні системи моніторингу (додаткові налаштування для деталізації)	Автоматичний збір метрик (у т.ч. lineage, час виконання на рівні моделей)
Інтеграція з Java	Вимагає використання власного SDK/конекторів	Стандартна інтеграція через HTTP-клієнти або офіційні бібліотеки
Ліцензійні умови	Висока вартість, складні моделі ліцензування, vendor lock-in	Вільне використання (Core), гнучкі тарифні плани (Cloud), відсутність vendor lock-in

2.2 Дослідження наявних API для вибраної ETL-системи

dbt - це ідеальний вибір для сучасних проектів, орієнтованих на дані. Завдяки гнучкості, масштабованості та чудовій інтеграції, він чудово підходить.

Такий висновок можна зробити на основі таких факторів, як підтримка API, збір метрик, можливості інтеграції та ліцензування програмного забезпечення. Тут dbt має перевагу над традиційними рішеннями, як-от Informatica PowerCenter або Microsoft SSIS. Слід зазначити, що dbt спеціально

розроблений для компаній, які працюють з хмарними сховищами даних та сучасними технологічними стеками. З цієї причини його архітектура була створена в такому середовищі.

Вивчаючи API dbt, можна побачити, що вони пропонують багато функцій. Ви можете не лише отримати метадані запусків, але й контролювати всі аспекти роботи системи в програмах. Це широке поле, яке відкриває можливості для автоматизації та розробки складних стратегій моніторингу.

API dbt складається з низки RESTful кінцевих точок, які надають всю необхідну інформацію про трансформації — від деталей про обставини кожної моделі до інформації про індекси продуктивності та навіть діаграми, що відображають залежності завдань. Важливою особливістю цього API є те, що він може повертати дані асинхронно або синхронно, що є критичним для систем реального часу.

Більш детальний огляд документації API показує, що він має інтерфейси для отримання історії запусків, статистики виконання окремих моделей, повідомлень про помилки та попередження, а також інформацію про організацію проекту. Ці можливості значно перевищують те, що пропонують подібні продукти, оскільки з іншими рішеннями доступ до таких важливих аспектів часто вимагає додаткової конфігурації або взагалі відсутній через стандартні інтерфейси.

API dbt заслуговує на похвалу за те, як він обробляє метадані. Система не лише надає чисто технічні деталі виконання завдань, але й метадані бізнесу, такі як описи та теги моделей, ідентифікатори власників, відповідальних за набори даних тощо. Це дозволяє створювати складні аналітичні рішення, які поєднують технічні та бізнес-аспекти роботи з даними.

API також надає систему подій, що дозволяє отримувати повідомлення в реальному часі про зміни в системі. Це має вирішальне значення при створенні наперед прогнозованих систем моніторингу. Ще одна перевага полягає в тому, що офіційні клієнтські бібліотеки для популярних мов програмування доступні, що значно спрощує інтеграцію dbt в існуючі рамки

розробки та BI-інструменти.

Важливим елементом оцінки API dbt є встановлення його меж і можливостей застосування в реальних проектах. Наприклад, деякі операції, пов'язані з керуванням операційним середовищем, вимагають комерційної версії dbt Cloud. Це може бути дорого. Також варто відзначити, що деякі статистичні дані про продуктивність доступні лише за подальшої інтеграції з такими системами, як Datalog або Snowflake.

Тим не менш, незважаючи на ці обмеження, API dbt надає набагато ширші та гнучкіші функції у порівнянні з аналогічними продуктами в своїй галузі. Він добре підходить для проектів, в яких дійсно потрібно інтегрувати робочі процеси ETL та аналітичні системи.

Таблиця 2.3 – Наявні API в системі ETL dbt

API Категорія	Опис функціоналу	Підтримка Jobs/Environments
Jobs API	Надає інформацію про створені jobs, їх статус, параметри запуску, історію виконання	Отримання списку jobs Деталі конкретного job
Run API	Керування запусками jobs (ручний/плановий), моніторинг статусу виконання	Запуск jobs Відстеження прогресу
Environments API	Керування середовищами (dev/prod), їх налаштуваннями, credentials	Перелік середовищ Оновлення параметрів
Metadata API	Доступ до метаданих проекту: моделі, тести, документація, lineage	Метадані для jobs у контексті середовищ
Artifacts API	Завантаження артефактів виконання (manifest, run results)	Результати конкретних запусків
Webhooks API	Налаштування webhook-сповіщень про зміни в jobs (наприклад, завершення run)	Автоматизація реакцій на події

2.3 Опис обраної ETL системи (dbt Cloud)

Архітектура системи DBT Cloud складається з трьох основних логічних

рівнів: проект, середовище та завдання. У результаті ця комбінація забезпечує безпечну та гнучку платформу управління перетворенням даних.

Проект є логічним контейнером для всіх артефактів, необхідних для здійснення перетворення даних. Ці артефакти включають SQL-моделі, макроси, плани тестування, документацію та файли конфігурації.

На відміну від цього, середовище визначає фактичний контекст, у якому відбувається виконання: з'єднання з базою даних, облікові дані та деякі налаштування. Існує принципова відмінність між цими двома рівнями: проект пояснює, що потрібно зробити (логіка перетворення), тоді як середовище вказує, як і де це слід виконати (контекст інфраструктури).

Наприклад, один проект може охоплювати кілька середовищ - dev для тестування, stg для валідації і prod для розгортання. Кожне середовище має різні параметри підключення до інших. Однак, принаймні база коду у всіх середовищах однакова, що демонструє як прихильність проекту, так і кінцеву заклопотаність повторним використанням коду.

2.3.1 Ідея параметризації в ETL-системах

Параметризація – це процес заміни жорстко закодованих значень у програмному коді або конфігурації на змінні (параметри), значення яких задаються під час виконання. Це дозволяє зробити програму більш гнучкою, масштабованою та легко адаптованою до змін у зовнішньому середовищі.

У контексті ETL (Extract, Transform, Load)-систем параметризація дозволяє конфігурувати обробку даних без необхідності змінювати логіку роботи потоків даних. Наприклад, можна задавати:

- Шляхи до джерел даних
- Назви таблиць або баз даних
- Діапазони дат для завантаження
- Типи завантаження (повне чи інкрементне)

Математичної формалізація параметризації. Розглянемо абстрактну модель ETL-процесу як функцію:

$$ETL: D_s \overset{T(\theta)}{\Longrightarrow} D_t \tag{2.1}$$

де:

- D_s - джерело даних (source dataset),
- D_t - цільовий набір даних (target dataset),
- $T(\theta)$ - трансформаційна функція, яка залежить від вектору параметрів θ.

Вектор параметрів θ може містити такі елементи:

$$\theta = \{\theta_1, \theta_2, \dots, \theta_n\} \tag{2.2}$$

де кожен θ_i окремий параметр (наприклад, назва таблиці, дата старту завантаження тощо).

Припустимо, що ми виконуємо інкрементне завантаження за допомогою параметра дати θ_{date}. Тоді фільтрація джерела даних математично описується як:

$$D_s' = \{x \in D_s \mid x.\text{date} \geq \theta_{\text{date}}\} \tag{2.3}$$

Трансформація застосовується лише до підмножини даних Ds′, після чого дані завантажуються в цільову систему:

$$D_t = \text{Load}\big(T(D_s', \theta)\big) \tag{2.4}$$

Переваги параметризації:

- Гнучкість - параметри можна змінювати без редагування основної логіки.
- Повторне використання - один і той самий процес можна застосовувати до різних джерел або діапазонів даних.
- Автоматизація - дозволяє інтегрувати ETL-процеси у пайплайни CI/CD.
- Зменшення помилок - менше ручного втручання під час запуску.

Приклад використання у практиці:

У системі, яка завантажує дані за певний день, можна використовувати параметр θdate, переданий у вигляді змінної середовища або конфігураційного файлу:

```
SELECT * FROM karazin_students
WHERE entollment_date >= '${START_DATE}'
```

При запуску ETL цей параметр підставляється в запит, наприклад: ${START_DATE} = '2025-01-01'.

2.3.2 Реалізація параметризації в ETL-системі dbt

У системах управління трансформаціями даних, зокрема в dbt Cloud, параметризація логічно організовується за рівнями Project, Environment та Job. Кожен із цих рівнів виконує специфічну роль у структурі керування даними, дозволяючи створювати відтворювані, масштабовані та контрольовані процеси трансформації.

Функціональної ролі кожного рівня системи:

1. Project - це контейнер, який включає в себе всі SQL-моделі, макроси, конфігурації, тести та документацію, що реалізують трансформаційний

шар даних. Він відображає логіку бізнесу і залишається незмінним для всіх середовищ.

2. Environment - це специфічна конфігурація для запуску проєкту. Вона включає параметри підключення до сховища даних, змінні середовища (env vars), версію dbt, а також значення змінних (vars), які впливають на логіку виконання моделей. Environments типово поділяються на: dev, staging, prod.
3. Job - це заплановане або ручне виконання dbt-команд у певному середовищі. Job запускає команди на основі конфігурації Environment і проєкту Project. При виконанні можна передавати змінні vars, вказувати конкретні моделі (--select) тощо.

Головна відмінність між Project та Environment полягає в зоні відповідальності. Тобто, project - це дизайн пайплайну, де описуються зв'язки та вирази котрі будуть виконуватися. В той час як environment - це контекст виконання, котрий перевизначає значення параметрів та змінних котрі були задані в проєкті в залежності від умов. Тобто в environment задаються зовнішні умови, такі як змінні, підключення, цільове середовище (target).

Математично цей розподіл можна виразити через композицію функцій:

$$ETL(P, E) = Run(P, \theta_E) \tag{2.5}$$

де:

- P - фіксований набір логіки (Project),
- E - Environment, який задає параметри виконання,
- θ_E - вектор параметрів, визначених у середовищі E,
- Run - оператор виконання dbt-команд.

Вектор параметрів середовища:

$$\theta = \{\theta_1, \theta_2, \dots, \theta_n\} \quad (2.6)$$

Наприклад, якщо у середовищі для розробкуи встановлено змінну (development):

```
vars:  is_incremental: false
```

А в середовищі виробництва (production):

```
vars:  is_incremental: true
```

Тоді логіка всередині моделі може змінюватися умовно:

```
{{ config(materialized='incremental' if var('is_incremental')
else 'table') }}
```

Таким чином, математичне уявлення залежності поведінки моделі від середовища:

$$M = f(P, \theta_E) \quad (2.7)$$

де M - результат виконання моделі, що є функцією логіки (P) та середовищних параметрів (θ_E).

Job є конкретною реалізацією запуску, яка фіксує момент часу та набір параметрів. Нехай кожен запуск Job можна описати як:

$$J_i = (P, E, C_i) \quad (2.8)$$

де C_i - конфігурація запуску (наприклад, --select, --vars, --defer), яка додатково уточнює поведінку моделі на момент виконання.

Тоді, можна описати повна формулу запуску як:

$$Result_i = Run(P, \theta_E, C_i) \quad (2.9)$$

Цей розподіл між рівнями дозволяє реалізувати повноцінну CI/CD-парадигму для аналітичних моделей, де логіка (Project) розробляється ізольовано від середовища виконання (Environment), а Job дозволяє повторно запускати логіку в різних контекстах з контролем версій та параметрів.

2.4 Дослідження підтримки баз даних

У контексті аналітичного стека dbt (інструмент складання даних) є частиною системи, яка забезпечує інтегровані дані та надає докази, що трансформація відбувається правильно.

dbt підтримує різноманітні аналітичні сховища даних, зокрема колоночні сховища, розроблені для обробки величезних обсягів даних за допомогою OLAP (онлайн аналітична обробка).

На даний час dbt офіційно підтримує такі типи з'єднань:

- Snowflake. Як сучасна хмарна аналітична платформа, Snowflake оптимізована для незалежної обробки та паралельних обчислень наборів даних - по суті, це завжди була її головна особливість. Це одна з небагатьох баз даних, адаптованих для "розділення обчислень і зберігання" (щоб ресурси масштабувалися незалежно). dbt інтегрується зі Snowflake, надаючи підтримку для таких функцій, як обчислювальні кластери, тимчасові таблиці та транзакційні таблиці. Управління ролями розширюється над цим рівнем для встановлення змінних середовища. Snowflake досить популярний у підприємствах завдяки своїй стабільній швидкості виконання, масштабованості та потужним функціональним запитам для високорівневого аналізу. Крім того, його ціновий механізм набагато простіший, ніж у інших хмарних баз даних (плата за фактичне використання), що відповідає критеріям частого виконання dbt.
- BigQuery. BigQuery діє як сховище даних на платформі Google Cloud. Окрім запуску SQL запитів на його інфраструктурі, BigQuery навіть дозволяє вам витягати дані безпосередньо з файлів, що зберігаються в

Google Cloud Storage. dbt має просту інтеграцію з BigQuery; це вимагає лише використання ключа облікового запису служби для простої аутентифікації. Підтримуються і настроювані конфігурації моделей. Чому його підтримують: BigQuery легко обробляє великі набори даних без необхідності управління кластерами, роблячи його ідеальним для компаній, що використовують GCP інфраструктуру. Перевага API полягає в тому, що dbt легко інтегрується в DevOps, якщо ви займаєтеся побудовою конвеєрів.

- PostgreSQL. Нативна підтримка dbt для PostgreSQL вимагає лише рядка з'єднання, тобто (хост, порт, користувач, пароль, dbname). Чому його підтримують: PostgreSQL дуже популярний серед стартапів, малих компаній та освітніх установ як база даних з відкритим кодом. З dbt ви можете перенести всю логіку трансформації даних безпосередньо в нього без необхідності додаткових витрат на інфраструктуру.
- Redshift. Amazon Redshift - це аналітичне сховище даних, розроблене для роботи в екосистемі AWS. Вона базується на PostgreSQL, але була адаптована для обробки великих даних, включаючи зберігання в стовпцях і структуру MPP (масово-паралельна обробка). Чому його підтримують: Одна з причин популярності Redshift серед компаній з існуючими сховищами даних на платформі AWS - це його широке використання. dbt підтримує його, розробивши на основі персоналізованого драйвера PostgreSQL, який розуміє ідіосинкразії Redshift (наприклад, обмеження на запити CTE, необхідність виконання VACUUM + ANALYZE).
- MySQL. Друга за величиною база даних з відкритим кодом у світі, MySQL (або його форк MySQL5) підтримує більшість функцій стандарту SQL. Dbt надає нативну підтримку підключення до MySQL. Зв'язуючись із MySQL, використання dbt для аналітичних запитів означає використання функцій, які може забезпечити лише база даних.
- Databricks / Apache Spark. Databricks — це платформа, яка пропонує

Apache Spark на основі розподіленого зберігання даних, разом з керованим виконанням SQL. Адаптер Dbt-Databricks вже підтримується dbt. Це середовище, яке визначає, чи потрібно використовувати JDBC чи HTTP API. Причини бути скептичними: у деяких великих неструктурованих середовищах аналізу даних, таких як обробка логів, JSON файли (.jsn) чи живі потоки (соціальні медіа), Spark і Databricks набули широкого використання. Таким чином, dbt як шар аналітичної логіки в такій архітектурі є частиною архітектури озера даних. Відмітні характеристики цих двох систем, включаючи їх популярність серед аналітиків даних, відкриті API, добре документовані діалекти SQL, масштабованість і гнучкість для майже будь-якого програмного забезпечення, доступного сьогодні на ринку, є важливими факторами, що визначають їх розташування. Отже, dbt поширює цей підхід до більшості сучасних платформ аналізу даних. Ви можете переключити логіку в моделі з будь-якої з цих систем на іншу без необхідності змінювати всю свою роботу (і так далі).

2.5 Дослідження базових одиниць операцій в ETL-системі Dbt

Система DBT (Data Build Tool) надає розробникам унікальну можливість декларативно і масштабовано формулювати перетворення даних. Реалізація цієї мети досягається за допомогою набору узагальнених об'єктів, які в DBT називаються «ресурсами». Кожен ресурс являє собою окрему логічну одиницю, яка виконує певну функцію в аналітичному конвеєрі. Було визначено, що певні ресурси відповідають за побудову таблиць на основі SQL-запитів. І навпаки, інші ресурси відповідають за надання документації, контроль якості, повторне використання коду або декларування зовнішніх джерел. Всебічне розуміння ресурсів у системі керування базами даних є необхідним для ефективної реалізації процесів вилучення, перетворення і завантаження (ETL) та вилучення, завантаження і перетворення (ELT). Ці процеси складають основу трансформаційного шару даних і визначають, як

інформація буде організована, перевірена, збережена і задокументована в аналітичному середовищі.

1. Моделі (models)

Моделі є ядром будь-якого проєкту dbt. Вони являють собою SQL-файли, в яких описується, яким чином повинна виглядати кінцева таблиця або представлення у сховищі даних. Під час виконання проєкту dbt компілює модель у повноцінний SQL-запит типу CREATE TABLE AS SELECT або CREATE VIEW AS SELECT, в залежності від конфігурації. Кожна модель - це самостійна одиниця логіки, яка відповідає за створення одного логічного шару аналітики. Завдяки цим моделям аналітик має змогу будувати багатошарові структури даних, розділяючи трансформацію на логічні кроки - від сирих таблиць до остаточних аналітичних агрегатів або дашбордів.

Моделі в dbt підтримують численні конфігурації, які дозволяють керувати матеріалізацією (тобто фізичним представленням даних), вказувати схему, теги, власника, політики документування колонок і багато іншого. Це забезпечує гнучкість, з якою можна налаштувати поведінку кожної моделі відповідно до бізнес-вимог. Наприклад, модель може бути створена як view (для динамічних запитів), як table (для стабільності та швидкого читання), або як incremental - що дозволяє оновлювати тільки нові або змінені записи, що дозволяє знизити навантаження на сховище.

Моделі також можуть викликати макроси, звертатись до інших моделей, використовувати тести, документацію та інтегрувати змінні середовища. Це робить їх не лише "будівельними блоками", а й повноцінними об'єктами програмованої логіки.

Приклад конфігурації моделі:

```
{{ config(
    materialized='table',
    schema='karazin',
    tags=['student', 'weekly'],
    persist_docs={"relation": true, "columns": true}
) }}
```

2. Сіди (seeds)

Сіди - це ресурс, що дозволяє імпортувати табличні дані з CSV-файлів безпосередньо в базу даних як повноцінні таблиці. Це надзвичайно зручний механізм для зберігання довідкової інформації, яка змінюється рідко або взагалі не змінюється. Наприклад, списки країн, валют, типів транзакцій або кодів категорій можна зберігати у вигляді CSV-файлів і завантажувати в базу автоматично в рамках пайплайну dbt seed.

Сіди забезпечують високу прозорість: вони можуть бути закомічені в репозиторій, проходити рев'ю, бути версіонованими, як звичайний код. У той же час, це повністю уникнення залежності від зовнішніх джерел під час розгортання тестових чи staging-середовищ - оскільки всі довідники вже локально наявні в репозиторії. dbt дозволяє конфігурувати типи колонок, кодування, символи лапок та обробку заголовків для сідів, забезпечуючи сумісність із різними SQL-движками.

Ці таблиці можуть бути матеріалізовані як table, view або навіть incremental, хоча в практиці найчастіше використовується матеріалізація table. Окрім цього, сіди часто застосовуються для генерації тестових даних у середовищах розробки або QA.

Приклад конфігурації сідів:

```
seeds:
  karazin_project:
    students:
      file: students.csv
      header: true
      quote_columns: true
      column_types:
        student_code: string
        stundent_fullname: string
```

3. Макроси (macros)

Макроси в dbt - це функції, які пишуться на основі шаблонізатора Jinja. Вони дозволяють створювати динамічні SQL-конструкції, зменшуючи дублювання коду і підвищуючи гнучкість моделей. Макроси можуть приймати аргументи, повертати SQL-рядки, здійснювати умови, цикли, обробку списків, виклик змінних середовища - і все це на етапі компіляції моделей.

Розробка макросів є способом підвищити рівень абстракції проєкту. Наприклад, замість того, щоб у кожній моделі вручну реалізовувати інкрементальну логіку, можна створити один макрос, який буде автоматично формувати WHERE-фільтр за останньою датою. У великих проєктах макроси дозволяють створювати бібліотеки стандартних трансформацій або аналітичних функцій, якими потім можуть користуватись усі учасники команди.

Макроси є основою для розширення dbt: плагіни, кастомні тести, навіть внутрішні функції самого dbt написані у вигляді макросів. Їх можна викликати у моделях, тестах, сідах, документації, навіть у schema.yml.

Приклад конфігурації макросів:

```
{% macro is_incremental() %}
    {{ return(flags.INCREMENTAL) }}
{% endmacro %}
```

Приклад використання макросу в моделі:

```
{% if is_incremental() %}
  WHERE updated_at > (SELECT max(updated_at) FROM {{ this
}})
{% endif %}
```

4. Тести (tests)

Тести в dbt виконують роль елементів контролю якості даних. На відміну від тестів у класичній розробці, вони не перевіряють функції, а

перевіряють, чи відповідають результати трансформації бізнес-вимогам або очікуванням. Це можуть бути такі перевірки, як "усі значення в колонці повинні бути унікальними", "значення не повинні бути null", "усі значення повинні входити до певного переліку" або навіть складні логічні залежності між таблицями.

Тести можуть бути вбудованими (generic) - які легко оголосити в schema.yml і які dbt інтерпретує автоматично - або кастомними, які створюються вручну у вигляді SQL-запитів. Якщо результат виконання тесту повертає хоча б один рядок, то вважається, що тест не пройдено.

Ключова цінність тестів у dbt полягає в тому, що вони тісно інтегровані з усім процесом складання проєкту. При запуску dbt test вони виконуються одразу після побудови моделей, що дає змогу виявляти помилки та аномалії на етапі розгортання, а не вже у звітах або продакшн-системах.

Приклад конфігурації тестів:

```
models:
  - name: karazin_students
    columns:
      - name: student_id
        tests:
          - unique
          - not_null
```

5. Документація (docs)

Документація в dbt - це мета-рівень опису моделей, колонок, джерел, сідів та макросів. Вона дозволяє аналітичній команді централізовано зберігати пояснення бізнес-сенсу таблиць і полів. Документація створюється через YAML-файли, де можна додавати описи до моделей і колонок, вказувати приклади, пояснення, метрики та пов’язані тести.

Цей підхід інтегрує процес написання документації безпосередньо у цикл розробки. Тобто при кожній зміні логіки трансформації або структури даних розробник має змогу - і навіть повинен - оновити опис відповідної таблиці

або поля. Це особливо корисно при зміні складу команди або масштабуванні проєкту.

Документацію можна згенерувати у вигляді інтерактивного веб-інтерфейсу (dbt docs generate), де автоматично будується граф залежностей (DAG), зв’язки між таблицями, тестами, джерелами. Таким чином, документація в dbt стає не просто описом, а важливим інструментом орієнтації в аналітичному ландшафті.

Приклад конфігурації джерел:

```
version: 2

models:
  - name: orders
    description: "Таблиця з усіма замовленнями"
    columns:
      - name: order_id
        description: "Унікальний ідентифікатор замовлення"
```

6. Джерела (sources)

Джерела - це декларативний опис таблиць, які вже існують у сховищі даних, але не створюються самим dbt. Це дозволяє dbt створити зв’язки між моделями і сирими даними без необхідності створювати ці таблиці вручну. Джерела вказуються в schema.yml, де описується база, схема та імена таблиць, які вже існують.

Цей механізм дозволяє підвищити прозорість: dbt фіксує, з яких саме таблиць розпочинається трансформація. Якщо змінився формат джерела або зникла таблиця, dbt одразу сповістить про це під час виконання, що захищає аналітичний стек від "тихих" помилок. Також джерела можуть бути предметом тестування, документування та версіонування.

Приклад конфігурації джерел:

```
sources:
  - name: raw
```

```
tables:
  - name: events
    description: "Сирі події з трекера"
```

Ресурси в dbt дозволяють логічно структурувати аналітичний стек, забезпечити гнучкість у трансформації, контроль якості, повторне використання коду та прозору документацію. Конфігурації дають можливість адаптувати поведінку ресурсів під конкретні потреби - від простих моделей до складних DAG-залежностей із condition-based виконанням.

2.6 Дослідження наявних рішень для створення BI звітів

У сучасних аналітичних екосистемах для аналізу метаданих із таких інструментів, як dbt (data build tool), активно використовуються BI-рішення, здатні не лише візуалізувати дані, але й надавати інтуїтивно зрозумілий інтерфейс для користувача, підтримку дашбордів, фільтрації та drill-down функцій. Серед найпоширеніших варіантів таких рішень - Tableau, Power BI, Looker, Apache Superset, Metabase, а також різні кастомні дашборди на базі бібліотек на кшталт Plotly Dash, Grafana чи Redash. Ці інструменти відмінно підходять для інтерактивної роботи з даними, які зберігаються у дата-складах (наприклад, Snowflake, BigQuery, Postgres), включно з метаданими про dbt-моделі, тести, залежності та Job-run'и.

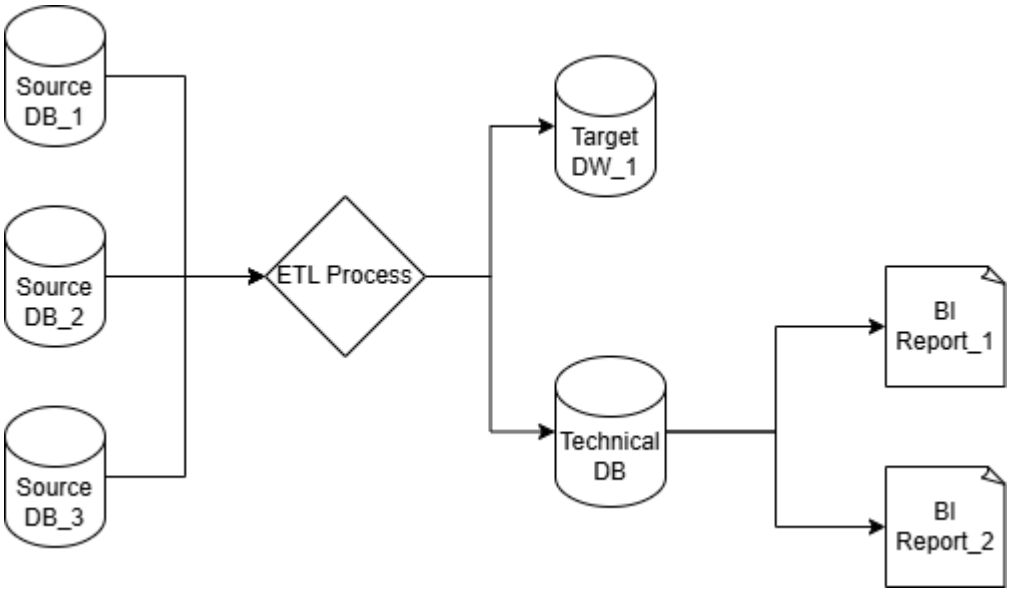

Рисунок 2.1 – Зв’язок ETL-систем із BI-системами

Однак у багатьох практичних випадках потрібні не лише інтерактивні дашборди, а й статичні, формалізовані звіти, що генеруються автоматично, зберігаються в архіві, надсилаються електронною поштою або прикріплюються до задач у Jira/Confluence. У таких ситуаціях рішення на кшталт Tableau чи Power BI можуть бути надлишковими або занадто складними в інтеграції. Саме тут на перший план виходить Jackson Reports, як легка, повністю Java-орієнтована альтернатива, що дозволяє програмно формувати звіти з високим рівнем кастомізації.

Однак є сценарії, де потрібна не інтерактивність, а формалізований, читабельний, автономний звіт, який можна зберегти, надіслати, прикріпити до звітності чи регламентованої документації. У таких випадках лідерами стають репортингові системи, здатні генерувати звіти у вигляді PDF, DOCX чи HTML. Одним із найпотужніших і найбільш поширених рішень у цій категорії є Jasper Reports. Цей інструмент вже багато років є де-факто стандартом звітності в Java-екосистемі, а його підтримка складних структур, багатих макетів, шаблонізації та сумісність з BI-системами робить його доречним вибором для роботи з dbt-метаданими.

На відміну від класичних BI-інструментів, Jasper Reports не вимагає

складного розгортання або налаштування середовища - звіт створюється в пам’яті програми і зберігається у PDF безпосередньо після збору метаданих із dbt Cloud API або іншого джерела. Завдяки цьому Jasper Reports ідеально підходить для автоматизованих пайплайнів, наприклад, nightly-job, що формує звіт про всі запуски моделі за день та відправляє його заздалегідь визначеним користувачам.

Хоча Jasper Reports не може замінити повноцінні дашборди з можливістю взаємодії, вона є ефективним доповненням до BI-інфраструктури, особливо в сценаріях, де важлива автоматизація, контрольована структура документа, легкість інтеграції в Java-проекти та формат PDF як стандарт обміну. Саме завдяки цим характеристикам Jackson Reports може успішно стояти поруч із провідними BI-рішеннями, покриваючи нішу репортингу для технічних команд, аудиту та регулярної документації.

JasperReports - це потужна, гнучка бібліотека для генерації звітів, яка дозволяє створювати високоякісні документи у форматах PDF, Excel, HTML, RTF, DOCX, ODT, ODS, XML та ін. Вона реалізована на Java, що робить її чудово сумісною з сучасними бекенд-сервісами та ETL-платформами, включно з тими, що використовують dbt як рушій аналітичної трансформації.

2.6.1 Особливості бібліотеки Jasper Reports

1. Шаблонна архітектура (JRXML)
   JasperReports базується на концепції попередньо створених шаблонів звітів у форматі JRXML - XML-документів, що визначають структуру, стилі, розмітку, прив’язку до джерела даних, таблиці, діаграми, підписи тощо. Це дає змогу чітко структурувати вигляд звіту незалежно від коду.
2. Візуальне проєктування (Jaspersoft Studio)
   Існує окрема IDE - Jaspersoft Studio, яка дозволяє створювати шаблони звітів у drag-and-drop режимі, додавати поля, тексти, графіку, лінії,

діаграми, умовне форматування, скрипти тощо. Після створення шаблон можна використовувати у Java-додатку для заповнення даними.

3. Підтримка складних джерел даних
   JasperReports може використовувати як SQL-бази (PostgreSQL, Oracle, Snowflake), так і Java Beans, CSV-файли, JSON-дані, REST API, і навіть кастомні джерела, що реалізують інтерфейс JRDataSource. Це дозволяє напряму подавати метадані з dbt Cloud API без проміжного збереження у базу.
4. Гнучке форматування та компоненти
   JasperReports підтримує динамічні таблиці, перехресні таблиці (cross-tabs), підзвіти, стилі, групування, секції, умовне оформлення, локалізацію. Можна створювати звіти, що змінюються залежно від вхідних параметрів або результатів аналізу.
5. Повна інтеграція з Java
   JasperReports має API (JasperFillManager, JasperExportManager, JasperReport) для генерації звітів із Java-коду. Це дозволяє вбудувати генерацію PDF у пайплайн, що реагує на події, наприклад, завершення dbt Job.

```
     JasperReport report =
JasperCompileManager.compileReport("job_report_template.jrxml");
     Map<String, Object> parameters = new HashMap<>();
     parameters.put("jobId", "123456");
     parameters.put("status", "Success");

     JRDataSource dataSource = new
JRBeanCollectionDataSource(jobMetadataList);
     JasperPrint print = JasperFillManager.fillReport(report,
parameters, dataSource);
     JasperExportManager.exportReportToPdfFile(print,
"job_summary.pdf");
```

6. Підтримка діаграм і візуалізацій

   JasperReports дозволяє включати лінійні діаграми, гістограми, pie charts, діаграми Ганта, які будуються на основі вхідних даних. Це особливо корисно для відображення метрик виконання моделей: час, кількість рядків, статуси.

7. Міжплатформність та розгортання

   JasperReports можна запускати як локально (у вигляді Java-програми), так і в вебсередовищі через JasperReports Server або інтеграцію з Spring Boot. Завдяки цьому він підходить для корпоративного використання.

2.6.2 Роль бібліотеки Jasper Reports у BI екосистемі

JasperReports займає унікальне місце в екосистемі бізнес-аналітики (BI), виступаючи не як класичний інструмент дашбордів для аналітиків, а як двигун формалізованої, структурованої звітності, яку можна вбудовувати у будь-який етап аналітичного або ETL-процесу. Його ключова сила полягає у здатності генерувати високоякісні документи, придатні не лише для перегляду, а й для архівування, ділового обігу, юридичної документації чи регулярної регламентованої звітності. Саме завдяки цьому JasperReports є незамінним у сферах, де потрібна повторювана, контрольована та стандартизована звітність з суворим дотриманням форматування та структури - у банківській сфері, аудиті, охороні здоров’я, державному управлінні, та багатьох інших галузях.

На відміну від таких BI-інструментів, як Power BI, Tableau, Metabase чи Looker, які орієнтовані на інтерактивну взаємодію користувача з даними, динамічну фільтрацію, перегляд у реальному часі та побудову дашбордів для аналітиків, JasperReports зосереджується на візуальному представленні заздалегідь визначеної логіки: шаблон, який формується один раз, може багаторазово наповнюватися актуальними даними, генеруючи стандартизований результат. Це робить його надзвичайно потужним

інструментом для тих випадків, коли BI-система має не лише допомагати приймати рішення, а й документувати ці рішення в уніфікованому вигляді.

Крім того, глибока інтеграція JasperReports із Java-екосистемою робить його природним вибором для підприємств, які вже використовують Java для створення backend-сервісів, ETL-сценаріїв чи корпоративних додатків. У такому випадку JasperReports може бути безпосередньо вбудований у системи, які опрацьовують дані, зокрема ті, що базуються на dbt: результати моделей, тести, статуси запусків job-ів, залежності моделей, або навіть результати валідації якості даних. Аналітик чи технічний користувач отримує готовий звіт - не лише з інформативним вмістом, а й із професійним візуальним оформленням, який не потребує ручної обробки.

У сучасному BI-ландшафті, де більшість рішень фокусуються на веб-візуалізаціях і онлайн-аналітиці, JasperReports впевнено закриває нішу автоматизованої PDF-звітності, яку часто ігнорують інші системи. Його можна ефективно використовувати для щоденних, щотижневих або місячних звітів, звітів за SLA, статусів моделей або якості даних у пайплайнах. Водночас, завдяки модульній архітектурі, гнучкості, open-source доступності та підтримці складних форматів виводу, JasperReports може повноцінно співіснувати в одному BI-середовищі з Looker чи Tableau, доповнюючи їх своїм орієнтованим на документ результатом

3 ОГЛЯД ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Створення вхідного набору даних в цільовій базі даних

Google BigQuery була обрана як основна цільова бази даних для зберігання, обробки та аналітичної трансформації даних у рамках реалізації проєкту з використанням dbt (data build tool). Такий вибір обумовлений не лише популярністю та хмарною природою цього рішення, але й його глибокою інтеграцією з dbt, підтримкою SQL-подібного синтаксису, автоматичним масштабуванням, а також розширеною підтримкою з боку Google Cloud Platform (GCP) для створення повноцінного аналітичного середовища.

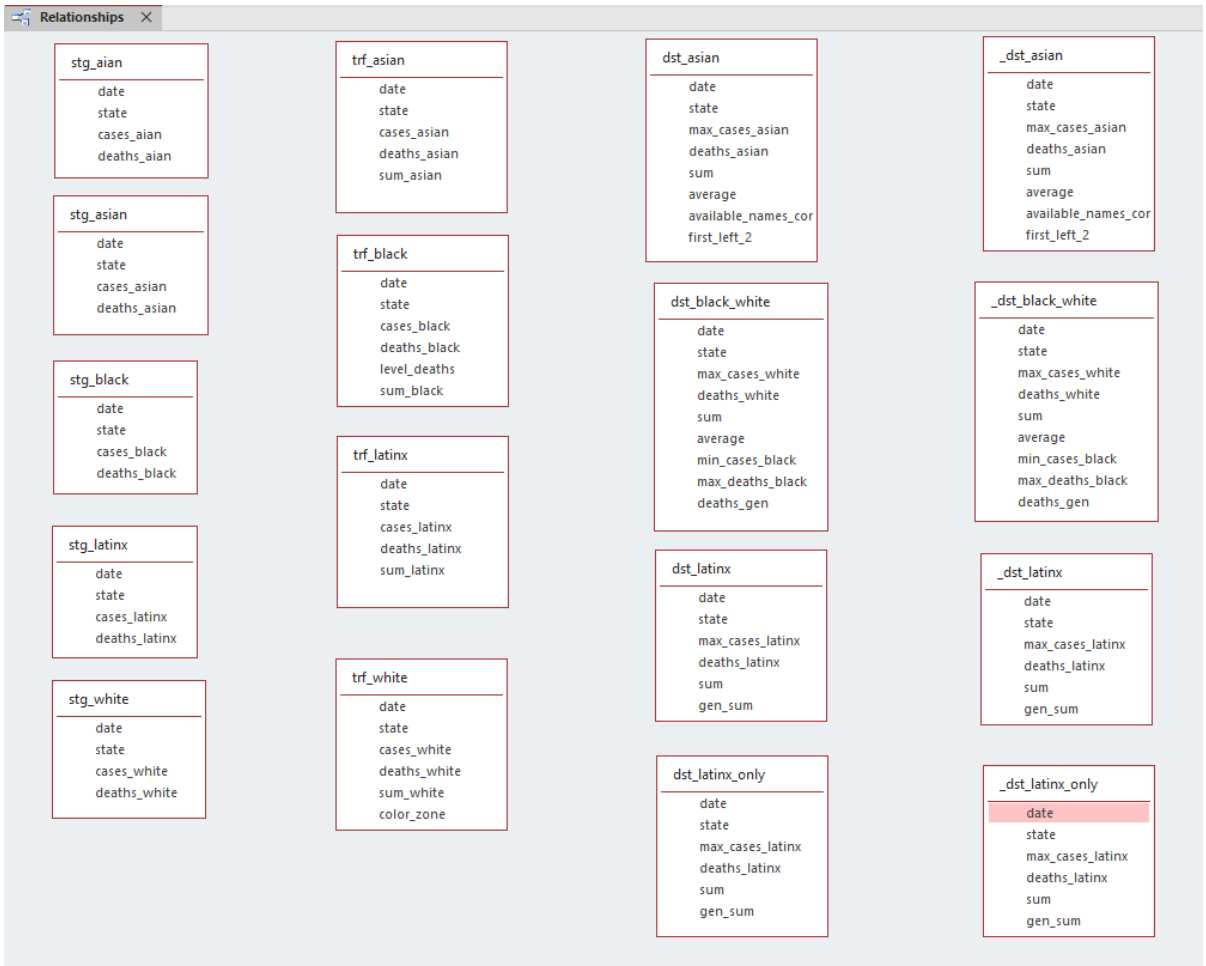

Рисунок 3.1 – Схема цільової бази даних

На скріншоті представлена ієрархія створених мною датасетів у рамках

проєкту maximal-boulder-389014, які логічно поділені на кілька функціональних блоків відповідно до етапів обробки даних. Було створено наступні набори даних: covid19, covid19_distribute, covid19_marts, covid19_open_data, covid19_staging та covid19_transform. Кожен з них виконує свою роль у загальному пайплайні підготовки та подальшої аналітики COVID-19 даних. Така структуризація дозволяє забезпечити чіткий поділ відповідальності між етапами: від завантаження та збереження сирих відкритих даних - до побудови аналітичних витягів (data marts), що вже використовуються для візуалізації або формування репортів.

Назви датасетів, які я використав у проєкті, обрані згідно з загальноприйнятими конвенціями в екосистемі dbt та сучасних практиках Data Engineering. Такі іменування дозволяють не лише структурувати великий об’єм даних за логікою обробки, але й забезпечити легке орієнтування у проєкті, масштабованість, повторне використання та прозору документацію. Кожен суфікс у назві датасету виконує семантичну функцію - він чітко вказує, на якому етапі життєвого циклу перебувають дані, які він містить.

Суфікс staging використовується для датасетів, які зберігають сирі або частково оброблені дані, зазвичай у тому вигляді, в якому вони надходять із зовнішніх джерел. Ці таблиці - це «буферна зона», де дані можуть бути перевірені, очищені, нормалізовані або переформатовані без порушення логіки більш «чистих» рівнів моделювання. Стейджинг є критичним етапом у будь-якому ETL-процесі, оскільки дозволяє ізолювати зовнішні зміни та виконувати контроль якості до їх подальшої обробки.

Суфікс transform означає, що у цьому датасеті відбувається логічна трансформація даних - наприклад, агрегації, об’єднання з іншими джерелами, обрахунок показників, нормалізація значень, зміна форматів тощо. Це основна зона аналітичної логіки, де dbt-моделі набувають своєї повної функціональності. У цьому шарі ми переходимо від «сирих» даних до аналітично осмислених структур. Він часто відповідає рівню “intermediate models” у dbt.

Суфікс marts означає, що ці таблиці — кінцева форма представлення даних для бізнесу або аналітиків. Це ті таблиці, з якими взаємодіє бізнес-користувач: вони можуть бути джерелами для BI-дашбордів, звітів або використовуватися як агреговані витяги. Дані тут уже максимально структуровані, готові до читання, мають фінальні назви колонок, одиниці вимірювання та найвищий рівень узгодженості. marts — це фактично "продукт" у Data-as-a-Product підході.

Суфікс distribute я використовую для датасетів, призначених для поширення або експорту даних у зовнішні системи. Це можуть бути спрощені витяги, які публікуються через API, передаються до інших баз даних, надсилаються в сторонні системи або використовуються для інтеграції. Дані тут часто мають мінімальні вимоги до внутрішньої аналітики, але повинні відповідати зовнішнім форматам і структурам.

Таке іменування підтримує layered architecture, яка є основною парадигмою в dbt: staging → transform → marts → distribute. Завдяки цьому легко зрозуміти, яка роль у моделі кожного з датасетів, і як вони взаємопов’язані. Це також дозволяє розмежувати права доступу: аналітики працюють лише з marts, тоді як розробники можуть перевіряти дані у staging та transform.

У рамках ознайомлення з Google BigQuery мені було необхідно вивчити його ключові особливості: модель зберігання даних у вигляді колонкових таблиць, особливості роботи з датасетами, обмеження на запити, вартість зчитування даних, а також можливості використання попередньо збережених запитів, розкладів та призначених таблиць-представлень. Окрему увагу також було приділено безпеці доступу, керуванню ролями та IAM-політиками в межах GCP, оскільки робота з великими обсягами даних вимагає чіткого контролю над дозволами.

Перевагою BigQuery є його здатність працювати у повністю серверлесс режимі - відсутня потреба в конфігурації фізичного середовища або ручному

масштабуванні. Це дозволяє зосередитися виключно на логіці ETL-процесів, при цьому не втрачаючи гнучкості в частині оптимізації продуктивності. Використання BigQuery як платформи для dbt-проєкту дало можливість швидко створювати, тестувати й валідувати моделі даних без необхідності турбуватися про ресурси виконання або підтримку інфраструктури.

3.2 Розробка проєкту в ETL-системі dbt

Проєктна реалізація ETL-системи здійснювалася за допомогою сучасного аналітичного фреймворку dbt (data build tool) у поєднанні з хмарною аналітичною платформою Google BigQuery. Основна мета - побудувати гнучкий, масштабований і підтримуваний пайплайн обробки COVID-19 даних, структурованих за расовими/етнічними групами в розрізі часу та штатів США. У межах проєкту було реалізовано повноцінну ієрархію моделей, поділених на логічні шари: staging, transform, marts, distribute. Такий підхід відповідає кращим практикам modular ETL, забезпечує контроль, повторне використання та простоту супроводу.

3.2.1 Вибір цільової бази даних

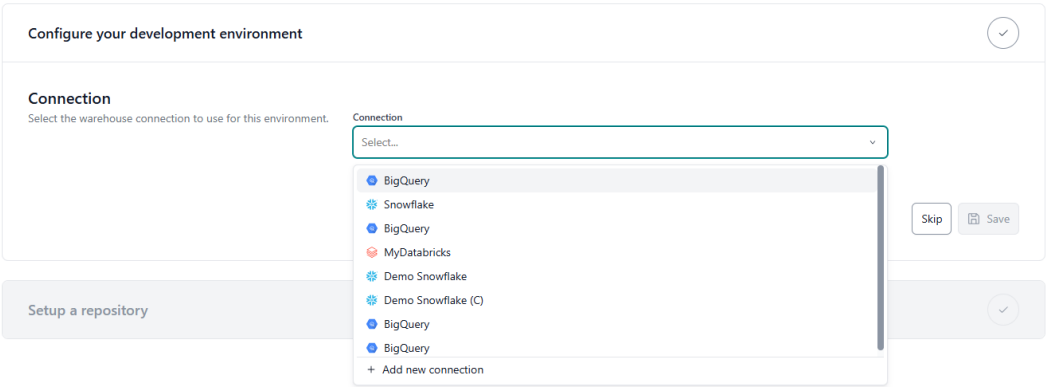

Рисунок 3.2 – Конфігурації з'єднань проєкту

Для зберігання та виконання SQL-запитів було обрано Google BigQuery, оскільки ця платформа має тісну інтеграцію з dbt, підтримує масштабування без необхідності налаштування кластерів, забезпечує високі швидкості

обробки завдяки колонковому сховищу, а також дозволяє легко працювати з великими об’ємами даних. Перевагою BigQuery також є можливість гнучко організовувати структуру сховища через датасети (наприклад, covid19_staging, covid19_transform, covid19_marts, covid19_distribute), що ідеально корелює з рівнями моделей dbt.

Для повноцінної реалізації ETL-логіки мені необхідно було ознайомитися з низкою особливостей BigQuery:

- квотами на виконання запитів і обсяг сканованих даних,
- правилами роботи з датасетами та IAM-політиками,
- нюансами SQL-сумісності (наприклад, функція EXTRACT, агрегати ARRAY_AGG),
- особливостями збережених views, materialized views, а також partitioned tables.

3.2.2 Ієрархія моделей dbt: загальна структура та підхід

У рамках dbt-проєкту було реалізовано повноцінну багаторівневу систему моделей. Вона побудована за принципами modular data modeling, де кожен логічний шар відповідає за окрему роль у трансформаційному процесі.

1. Моделі staging (наприклад: stg_latinx, stg_white, stg_asian, stg_states) — виконують попереднє очищення, фільтрацію, нормалізацію та найпростішу трансформацію вихідних даних. Вони слугують базовим джерелом для наступного шару.
2. Моделі transform (наприклад: trf_latinx, trf_white, trf_black, trf_asian, trf_aian_exccess) — виконують складнішу логіку: об’єднання з реєстрами штатів, генерацію нових показників (sum_white, color_zone, level_deaths), обчислення аналітичних атрибутів, умовну логіку (CASE WHEN). Також застосовується фільтрація (WHERE) за станом даних.
3. Моделі marts (наприклад: dst_latinx, dst_black_white, dst_asian, dst_latinx_only) — є аналітичним фінальним шаром, що агрегує та презентує дані у формі, придатній для звітності. Вони часто містять

підрахунок AVG, SUM, MAX, GROUP BY логіку, узгоджені назви колонок, і структуровані за фіксованим форматом, зручним для аналітиків.

4. Моделі distribute (наприклад: _dst_black_white, _dst_latinx, _dst_asian) — це "копії" фінальних моделей, які можна легко передавати зовнішнім системам або експортувати через API. Вони зберігаються у вигляді окремих views, розміщених в іншому схемі distribute.

Дерево моделей:

```
hierarchy
├── staging
│   ├── stg_aian.sql
│   ├── stg_asian.sql
│   ├── stg_black.sql
│   ├── stg_latinx.sql
│   └── stg_states.sql
├── transform
│   ├── trf_aian_exccess.sql
│   ├── trf_asian.sql
│   ├── trf_black.sql
│   ├── trf_latinx.sql
│   └── trf_white.sql
├── marts
│   ├── dst_asian.sql
│   ├── dst_black_white.sql
│   ├── dst_latinx.sql
│   └── dst_latinx_only.sql
└── distribute
    ├── _dst_asian.sql
    ├── _dst_black_white.sql
    ├── _dst_latinx.sql
```

```
└── _dst_latinx_only.sql
```

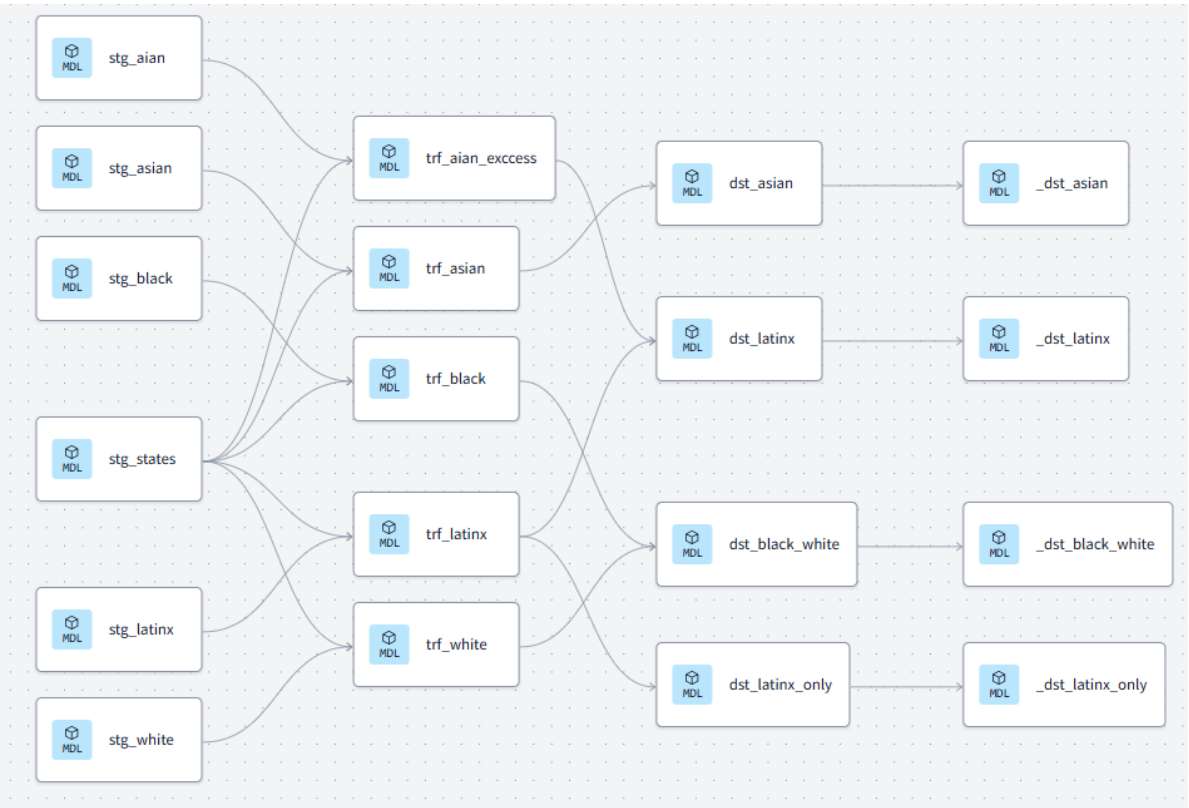

Рисунок 3.3 – Залежності dbt моделей

3.2.3 Покроковий процес розробки моделей

Розробка ETL-логіки у dbt потребує чіткого та послідовного підходу:

1. Імпорт вихідних даних

   Перший крок - визначити джерело (наприклад, BigQuery таблиці covid19.city_level_cases_and_deaths) та створити stg_-моделі, які фільтрують null-значення, обирають потрібні поля, забезпечують однорідність даних.

2. Побудова моделей transform

   Наступний етап — застосування логіки об’єднання з допоміжними таблицями (stg_states), розрахунок нових колонок, включення логіки умовного форматування (визначення «зон ризику»), а також перетворення даних у зручну форму для аналітики.

3. Формування marts-моделей

   Це аналітичні агрегати. Тут відбувається злиття даних з кількох джерел

(наприклад, trf_black + trf_white), агрегація по даті і штату, розрахунок ключових метрик — середніх значень, максимумів, загальних сум. Таким чином створюються фінальні звіти для виводу.

4. Створення distribute моделей
   На останньому етапі формуються lightweight views, які дублюють результат marts, але розміщуються окремо, що спрощує доступ зовнішніх систем або BI-інструментів (Power BI, JasperReports, ін.).

3.2.4 Конфігурація моделей

Для кожної моделі dbt задається конфігурація у Jinja-форматі. Найважливішими параметрами стали:

- materialized="table" - для важливих моделей з інтенсивною логікою, що не повинні повторно обчислюватися.
- materialized="view" - для легких обгорток, розрахованих на динамічне оновлення.
- schema="transform" / "marts" - явно вказано логічну зону для зберігання.
- Фільтрація, сортування, агрегації — реалізовані через SQL/Jinja.
- ref() — усі моделі використовують декларативну залежність через {{ ref('model_name') }}, що дозволяє dbt автоматично побудувати DAG.

Аналітичні (marts) моделі виконують ключову функцію: вони є інтерфейсом між складною сирою інформацією та кінцевим користувачем. Вони узагальнюють дані, уніфікують їх, згладжують варіації, приховують технічні деталі джерел. Саме на основі цих моделей можуть будуватися BI-звіти, PDF-документи (JasperReports), API-ендпоїнти або дашборди.

Ці моделі не просто збирають агрегати — вони також є місцем впровадження бізнес-логіки: визначення рівнів загрози, зон смертності, аналіз ексцесів по роках. Саме завдяки таким структурам забезпечується зрозумілий, сталий та повторюваний аналітичний шар.

3.2.5 Опис логіки моделей

Очистка та фільтрація (на рівні staging)

У початковому шарі, наприклад у stg_latinx, stg_black, stg_asian, виконується:

- відбір колонок: SELECT date, state, cases_latinx, deaths_latinx
- очистка даних: фільтрація null-значень у критичних полях, як-от cases_latinx IS NOT NULL AND deaths_latinx IS NOT NULL
- підготовка до нормалізації: уникаються агрегації та злиття на цьому етапі, щоб зберегти максимальну гнучкість на подальших рівнях.

Трансформації на рівні transform

У трансформаційних моделях (trf_white, trf_latinx, trf_asian) виконуються більш складні колонкові операції:

- Об’єднання даних: через JOIN з stg_states для додавання state_name

```
inner join states on states.state = latinx.state
```

- Формування нових колонок:
  - CONCAT(latinx.state, '-', states.state_name) AS state — створення розширеного географічного ідентифікатора.
  - latinx.cases_latinx + latinx.deaths_latinx AS sum_latinx — обчислення сумарного навантаження.
  - Фільтрація за регіоном: WHERE states.state = 'california' (або 'NY' для Asian).

Ці трансформації створюють колонкову базу для аналітичного аналізу, зберігаючи і контекст (географія), і обчислені значення.

Агрегації та обчислення (на рівні marts)

У моделях типу dst_asian, dst_latinx_only, dst_black_white виконуються складні агрегації та статистичні обчислення:

- MAX, MIN: визначення максимуму випадків і мінімуму смертей.

```
MAX(cases_asian) AS max_cases_asian,
```

```
MIN(deaths_asian) AS deaths_asian
```

- COALESCE та SUM:

```
SUM(COALESCE(cases_latinx, deaths_latinx)) AS sum
```

- AVG, LEFT():

```
AVG(sum_asian) AS average,
LEFT(state, 2) AS first_left_2
```

- Використання змінної:

```
{{ var('names') }} AS available_names_const
```

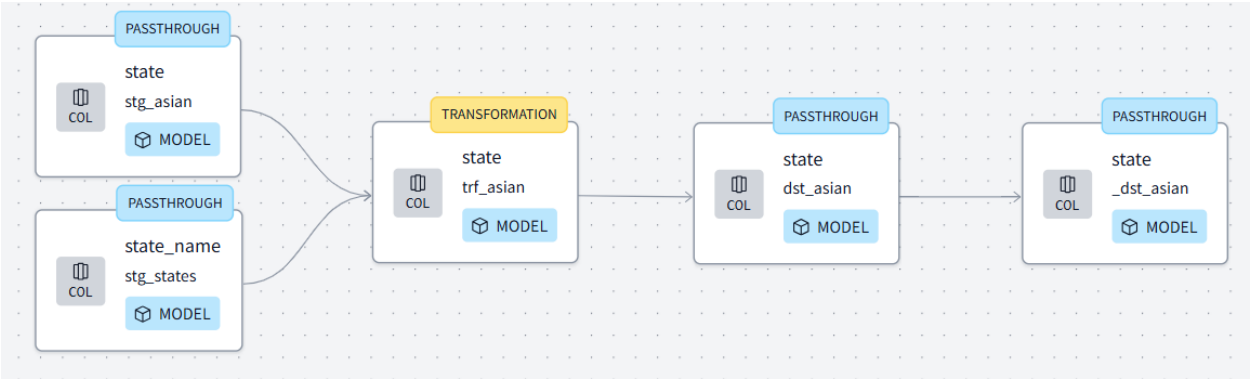

Рисунок 3.4 – Приклад течії даних для колонки «state»

Ці моделі забезпечують фінальну форму даних для аналітиків, агреговану за date та state. Підбиваючи підсумки, можна сказати що трансформації на рівні колонок охоплюють повний спектр:

- від очищення та фільтрації (staging),
- до обчислення похідних змінних та географічного збагачення (transform),
- і завершуються аналітичними агрегатами та узагальненням для бізнесу (marts).

Це дозволяє будувати структурований, стабільний і масштабований шар даних, що готовий до звітності та використання у BI-системах.

3.3 Розгортання проєкту

Наступним етапом у реалізації проєкту стало створення середовища

розгортання (Deployment Environment) у хмарному сервісі dbt Cloud. Це середовище було створене з метою забезпечення централізованого, стабільного та контрольованого процесу виконання моделей, тестів та документування в умовах наближених до продуктивних. Середовищу була надана назва Deployment, що чітко відображає його призначення: автоматизоване або ручне розгортання трансформаційних моделей до цільової бази даних - Google BigQuery.

У рамках цього середовища було створено Job - це окремий об’єкт в dbt Cloud, який визначає, які саме дії потрібно виконати в зазначеному середовищі. Для Job було вказано послідовність команд: dbt build, dbt test, dbt docs generate. Цей набір команд дозволяє не лише побудувати всі моделі, але й виконати контроль якості, та сформувати оновлену документацію про проєкт. Тестові запуски Job виконувалися в інтерактивному режимі, що дозволило поступово перевірити, як працюють всі компоненти dbt у контексті розгортання, які саме моделі компілюються, які дані генеруються у сховищі, та як відбувається валідація і документування.

3.3.1 Опис команд для розгортання проєкту

Команда dbt build є комплексною - вона поєднує в собі дії кількох інших команд: dbt run, dbt test, dbt snapshot і dbt seed. Основною її функцією є побудова всього графа моделей, включно з їх виконанням, застосуванням матеріалізації, запуском тестів, і створенням залежностей. Під час виконання dbt build, dbt:

1. Компілює кожну модель у повноцінний SQL-запит. Компіляція відбувається шляхом обробки Jinja-шаблонів, змінних (vars), макросів і викликів ref().
2. Визначає DAG-залежності між моделями. dbt будує орієнтований ациклічний граф (DAG), що відображає послідовність виконання моделей, враховуючи залежності через ref().
3. Послідовно виконує SQL-запити в базі даних. У моєму випадку, це

означає, що BigQuery отримує запити CREATE OR REPLACE TABLE, CREATE VIEW або MERGE INTO, залежно від обраної матеріалізації (table, view, incremental, тощо). Усі моделі компілюються у файли .sql, після чого dbt через API викликає BigQuery-движок для виконання запитів.

4. Створює таблиці, перегляди та тимчасові структури у відповідних датасетах. Наприклад, моделі типу trf_black чи dst_latinx_only створюються як фізичні таблиці в датасеті covid19_transform або covid19_marts.
5. Застосовує модульні тести (якщо включено). Після успішного створення кожної моделі, dbt запускає тести, які оголошені в YAML-документах або через макроси.

Окремо варто підкреслити важливість запуску dbt test. Ця команда виконує перевірку якості даних, засновану на заздалегідь визначених правилах. Наприклад, перевірки unique, not_null, accepted_values, relationships. Вони застосовуються до колонок у моделях, щоб гарантувати, що дані не містять критичних аномалій.

Під час виконання dbt test:

- dbt формує окремі SQL-запити, які повертають рядки, що порушують умови (наприклад, дублікати, null-значення).
- Якщо результат запиту не порожній — тест вважається неуспішним.
- Усі результати зберігаються та відображаються у web-інтерфейсі dbt Cloud у вигляді "помилок валідації".
- У базі жодних змін не відбувається, проте dbt виконує фактичні аналітичні запити до BigQuery для кожного тесту, що вимагає ресурсів.

Після успішного створення та перевірки моделей я виконав команду dbt docs generate, яка відповідає за створення інтерактивної документації по всьому проєкту. Під час виконання цієї команди:

- dbt збирає всі наявні моделі, сіди, джерела, тести, макроси та залежності.
- Аналізує DAG, типи колонок, описання з YAML-файлів (description:), документацію до кожної моделі й кожної колонки.
- Формує повноцінний HTML-сайт у вигляді графу, де кожна модель має свою сторінку з SQL-кодом, описом, тестами, вхідними залежностями.
- Результат можна опублікувати або переглянути локально (dbt docs serve).

Це критично важливо у великих проєктах, де без документації аналітикам складно зрозуміти структуру моделей, їхню мету, вхідні дані та бізнес-логіку.

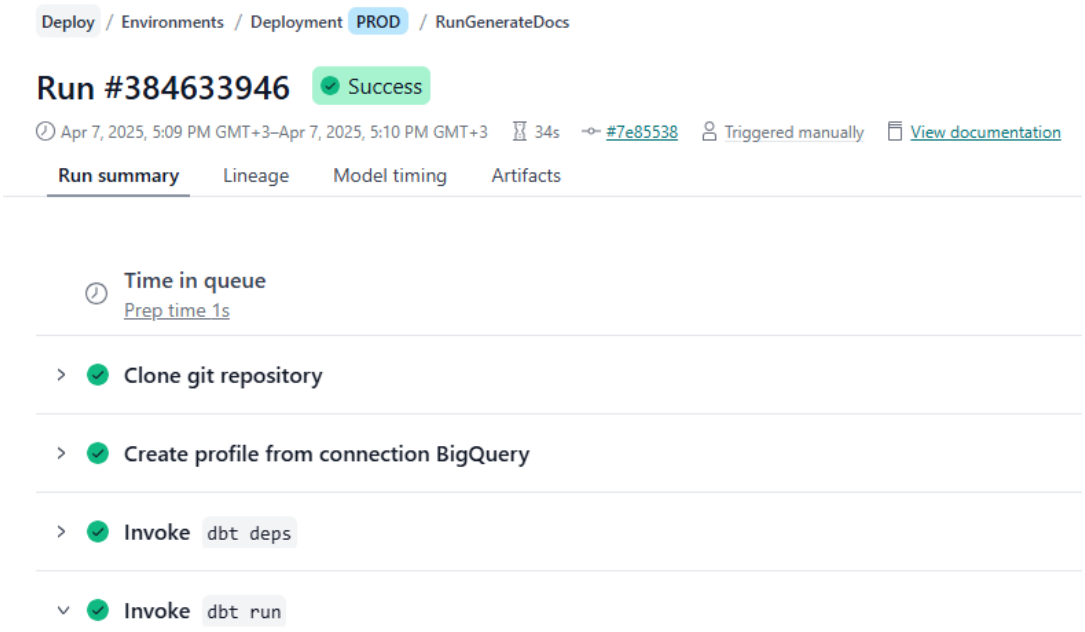

Рисунок 3.5 – Приклад успішного запуску Job у ETL-системі dbt

3.3.2 Опис процесу у цільовій базі даних під час розгортання проєкту

У Google BigQuery dbt створює фізичні структури - таблиці або перегляди - у вказаних датасетах (staging, transform, marts, distribute). Під час запуску Job, BigQuery отримує на виконання всі скомпільовані SQL-запити. Ці запити створюють, оновлюють або замінюють таблиці. У разі використання incremental-матеріалізації застосовується умовна логіка оновлення — лише

нові записи додаються. Таблиці в BigQuery мають автоматичну схему колонок, типи яких визначаються у SQL-моделях dbt.

dbt також створює спеціальні службові таблиці (наприклад, dbt_artifacts, dbt_metadata, dbt_run_results) для ведення логів виконання, статусів моделей і історії запусків. Ці таблиці використовуються пізніше для моніторингу або створення аналітики поверх ETL-процесу.

Запуск Job у середовищі Deployment не лише реалізовує ETL, а й гарантує повноту, контроль якості та документованість усіх етапів. Це формує надійний, повторюваний і масштабований пайплайн, придатний для щоденної експлуатації в будь-якому BI-проєкті.

3.4 Розробка Java-додатку для інтеграції з dbt

У рамках програмної реалізації проєкту було розроблено графічний інтерфейс користувача (GUI) для зручної взаємодії з параметрами, необхідними для збору метаданих із dbt Cloud та подальшого створення звітів. Інтерфейс було реалізовано з використанням JavaFX — сучасного фреймворку для створення графічних додатків на Java. Його використання забезпечило гнучкість компонування елементів інтерфейсу, високу продуктивність, а також кросплатформену підтримку, що важливо для роботи в різних середовищах (Windows, macOS, Linux).

Головне вікно додатку містить набір полів введення, які дозволяють користувачу вказати ключові параметри: шлях до каталогу, в якому буде збережено PDF-звіт, URL-адресу орендованого інстансу dbt Cloud, API-токен для авторизації, ідентифікатор облікового запису та ідентифікатор середовища. Така структура введення дозволяє повністю абстрагувати користувача від внутрішніх механізмів API-викликів і зробити запуск генерації звіту інтуїтивно зрозумілим.

У основі інтерфейсу лежить сіткова розмітка (GridPane), яка дозволяє впорядкувати елементи у вигляді таблиці, де кожне текстове поле розміщене поряд з відповідною міткою. Це робить форму логічною та зручною для

заповнення. Користувач вводить необхідні дані в поля, після чого натискає кнопку "Process", яка ініціює обробку введених параметрів.

Натискання кнопки активує обробник подій, що читає значення з усіх текстових полів і передає їх у метод для подальшого використання. На цьому етапі додаток виконує вивід параметрів у консоль (що є базовим кроком для налагодження), однак у реальній реалізації ця логіка доповнюється викликами API, отриманням метаданих із dbt Cloud, обробкою результатів та генерацією PDF-документу на основі зібраної інформації.

Цей інтерфейс створює зручну точку входу до всієї системи звітності, дозволяючи запускати процеси без необхідності використовувати консоль або передавати аргументи вручну. Він також відкриває перспективу подальшого розвитку в напрямку повноцінної настільної BI-утиліти, яку можна передати аналітикам для щоденного використання без знання внутрішньої структури dbt чи BigQuery.

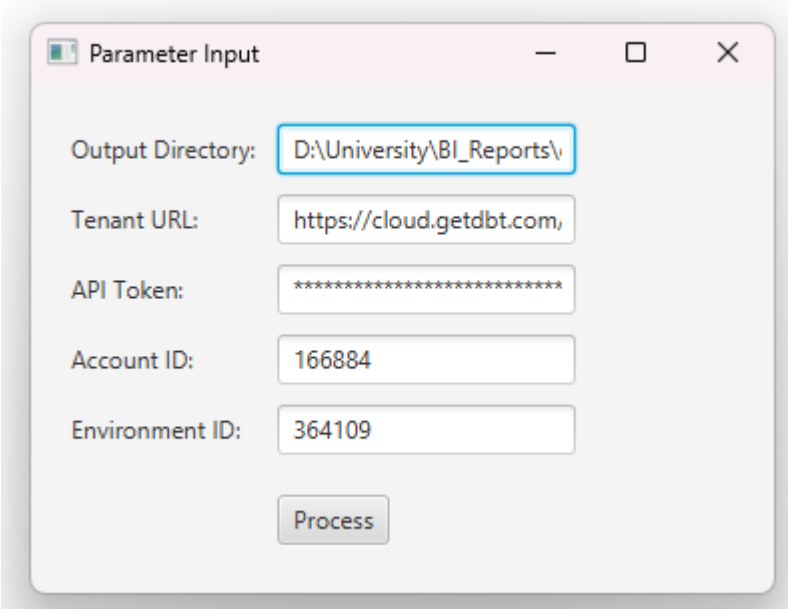

Рисунок 3.6 – Інтерфейс Java-додатку для генерації BI-звітів

3.5 Отримання та підготовка метаданих із серверу dbt

Одним із ключових завдань у реалізації системи звітності було отримання метаданих про виконання ETL-процесів, ресурси та моделі з

хмарного сервісу dbt Cloud. З цією метою було реалізовано окремий програмний компонент, відповідальний за взаємодію з API dbt Cloud, обробку отриманої інформації, а також структурування метаданих у зручному для аналізу форматі. Весь процес реалізований мовою Java, з використанням вбудованих засобів мережевої комунікації та обробки JSON-структур.

У центрі цього компоненту знаходиться спеціальний клас, який відповідає за з'єднання з API dbt, авторизацію, побудову запитів, обробку відповіді та трансформацію отриманих даних у Java-об'єкти. При ініціалізації система отримує основні параметри підключення — це URL-адреса орендованого облікового середовища, токен доступу API, а також ідентифікатори облікового запису й середовища розгортання. Ці параметри зберігаються для формування цільових HTTP-запитів до серверу dbt Cloud.

Процес взаємодії починається з того, що програма надсилає запити до API, використовуючи захищений HTTP-протокол з авторизаційним токеном. Система послідовно виконує запити до різних endpoint-ів API, таких як перелік середовищ, список доступних job-ів, інформація про останні виконання, а також JSON-маніфести моделей (включаючи файл manifest.json). Кожен відповідь API — це структура у форматі JSON, яка розбирається за допомогою парсера, перетворюється у внутрішні об'єкти (наприклад, Job, Resource, Statistic) і зберігається в оперативну пам'ять для подальшого аналізу.

Окрему увагу приділено обробці ресурсів, які були створені або оновлені під час запусків моделей. Програма враховує лише ті ресурси, які є актуальними на момент запиту — для цього реалізована логіка порівняння міток часу створення моделей. Також фільтруються типи ресурсів, зокрема, тестові об'єкти (test) не враховуються, щоб у фінальному звіті були лише суттєві одиниці трансформаційної логіки (моделі, сіди, snapshots, макроси тощо).

Система також проводить агрегацію метаданих, визначаючи загальну кількість job-ів, частку успішних, помилкових або пропущених запусків. Для кожного job-у відстежується час початку, час завершення, тривалість та статус

виконання. Також виконується класифікація моделей за типами матеріалізації (наприклад: view, table, incremental) та за логічними namespace-ами (наприклад, STAGING, TRANSFORM, MARTS). Це дозволяє візуально відстежити, які типи моделей домінують у кожному шарі аналітичного стека.

Особливу роль відіграє обробка артефактів запуску — зокрема, manifest.json, який містить опис усіх моделей та їхню компіляцію. Програма витягує поля, що визначають назву моделі, унікальний ідентифікатор, матеріалізацію, розташування у схемі, дату створення та інші службові параметри, включно з повною назвою таблиці у Google BigQuery.

Отримані метадані не зберігаються в базі, а використовуються безпосередньо у пам’яті для генерації звіту та побудови агрегованої статистики - наприклад, кількість моделей на кожен рівень, співвідношення успішних запусків, розподіл типів ресурсів. Це забезпечує високу швидкодію та відсутність потреби в додатковому сховищі.

Такий підхід дозволив створити гнучкий і надійний канал збору інформації про dbt-проєкт, який може бути використаний як у внутрішніх аналітичних задачах, так і для зовнішнього аудиту, моніторингу стабільності пайплайнів, або формування регулярної звітності для керівництва.

3.6 Створення шаблону для автоматичної генерації BI-звітів

У рамках створення BI-звіту на основі метаданих з ETL-системи, зокрема dbt (Data Build Tool), важливим етапом є побудова шаблону звіту. Шаблон, розроблений у JasperReports, виконує роль каркасу, який дозволяє автоматизовано формувати звіти на основі динамічних даних, отриманих з аналітичної моделі даних. Створений JRXML-файл (JasperReports XML) описує структуру, компоненти та логіку візуалізації звіту.

У тілі секції detail шаблону звіту JasperReports розміщено ключові компоненти візуалізації, які дозволяють глибше проаналізувати стан і структуру аналітичного проєкту, що побудований на базі dbt. Кожен з цих компонентів відповідає за специфічну частину інформації, і візуально

трансформує сирі метадані у зручну для сприйняття аналітиком форму.

Одним із центральних візуальних елементів у detail є Meter Chart, який використовується для відображення статусів виконання dbt-завдань (Job). Це напівкругла шкала, яка вказує поточний стан останнього запуску dbt-моделі, наприклад, успішно завершено, завершено з помилкою або взагалі не запущено. Таке подання дозволяє миттєво оцінити технічну справність пайплайна і виявити проблемні зони. Завдяки кольоровому кодуванню (зелений, червоний, жовтий) аналітик або інженер даних без зчитування тексту бачить загальний стан проєкту.

Рисунок 3.7 – Приклад візуального компоненту «Meter Char»

Поруч із цим розміщується Pie Dataset, що подається у вигляді кругової діаграми (pie chart) для деталізації статусів Job по категоріях — скільки з них успішні, скільки з помилками, скільки не запускалися. Ця діаграма ґрунтується на агрегованих даних і дозволяє швидко охопити загальний розподіл статусів у проєкті. Її використання особливо корисне в процесі аудиту чи моніторингу роботи моделей, оскільки вона миттєво візуалізує дисбаланс чи наявність критичних помилок.

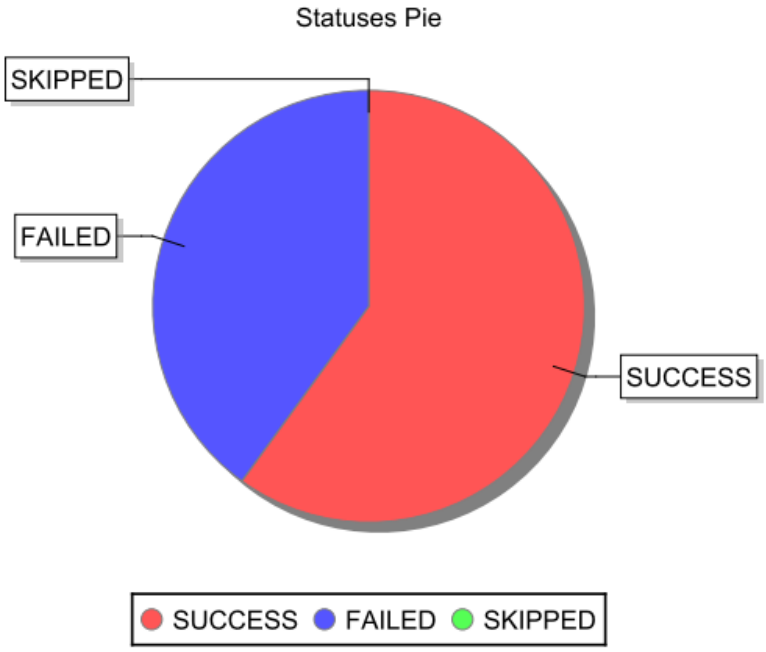

Рисунок 3.8 – Приклад візуального компоненту «Pie Chart»

Далі у звіті використовується Bar Chart, який демонструє розподіл типів ресурсів у проєкті: моделі, сіди, аналізи, макроси, джінджі-теги тощо. Гістограма показує кількість кожного з цих типів у проєкті, дозволяючи зрозуміти структуру та складність dbt-рішення. Аналітик може з цього блоку зробити висновки про масштаби проєкту, рівень автоматизації (наприклад, наявність макросів), і навіть загальну організованість репозиторію.

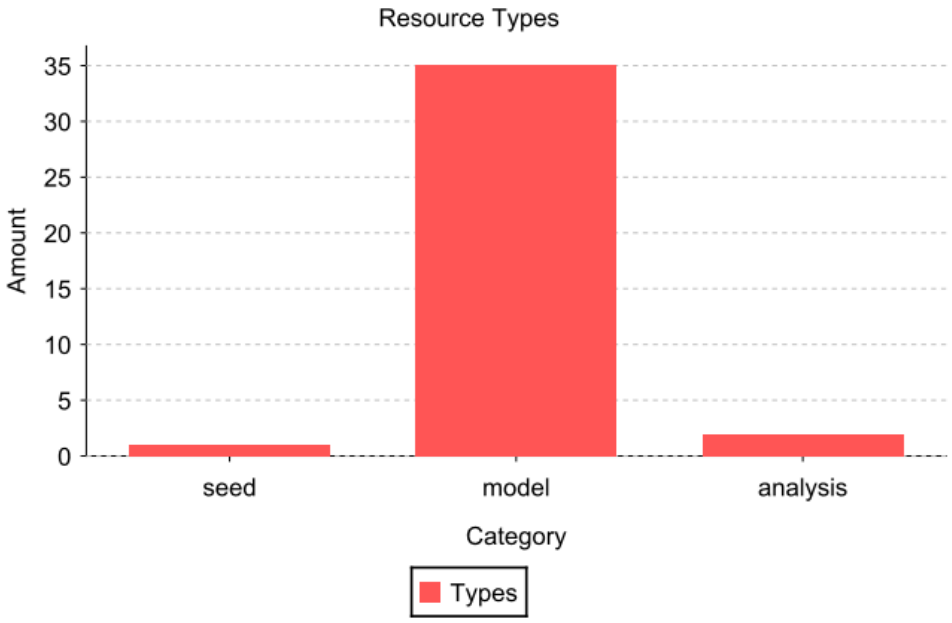

Рисунок 3.9 – Приклад візуального компоненту «Bar Chart»

Нижче представлена таблична візуалізація моделей, яка формує детальний перелік моделей з метаданими: назва, тип, опис, джерело, статуси. Це - найбільш інформативна частина звіту, оскільки вона детально описує кожен об’єкт, що присутній у моделі даних. Усі ці атрибути зчитуються автоматично з dbt-документації чи артефактів, і подаються у стандартизованому форматі, що спрощує рев’ю моделі, пошук помилок або порушень стандартів іменування чи структурування. Також, таблиця надає можливість перейти безпосередньо до об'єкту у ETL-системі. Адже додаток динамічно генерує відповідне URL-посилання для кожного об’єкту таблиці.

Name	Status	Resource Type	Materialization	Job ID
lineage_investigation	success	model	table	847371
trf_employee_department_project	success	model	table	847371
stg_employee_details	unknown	model	table	735303
dbt_config_materialized_view	success	model	view	735303
dbt_config_test_sources	success	model	table	735303
dbt_config_materialized_table	success	model	table	735303
dst_employee_department_project	unknown	model	table	735303
stg_project_assignments	unknown	model	table	735303
dbt_config_test_sources	success	model	table	847371
dbt_config_database	success	model	table	735303
stg_employee_details	success	model	table	847371
stg_departments	success	model	table	847371

Рисунок 3.10 – Приклад візуального компоненту «Interactive Table»

Завершує секцію detail Stacked Bar Chart, яка демонструє скільки моделей і якої матеріалізації зберігаються у відповідних схемах баз даних. Кожна колонка складається з шарів, що відповідають різним типам моделей або станам. Це дає змогу не лише побачити кількість об’єктів на рівні схем, а й оцінити розподіл типів моделей між ними. Така візуалізація особливо важлива в умовах багатосхемних проєктів, де організація моделей має значення для контролю доступу, продуктивності та логічного групування.

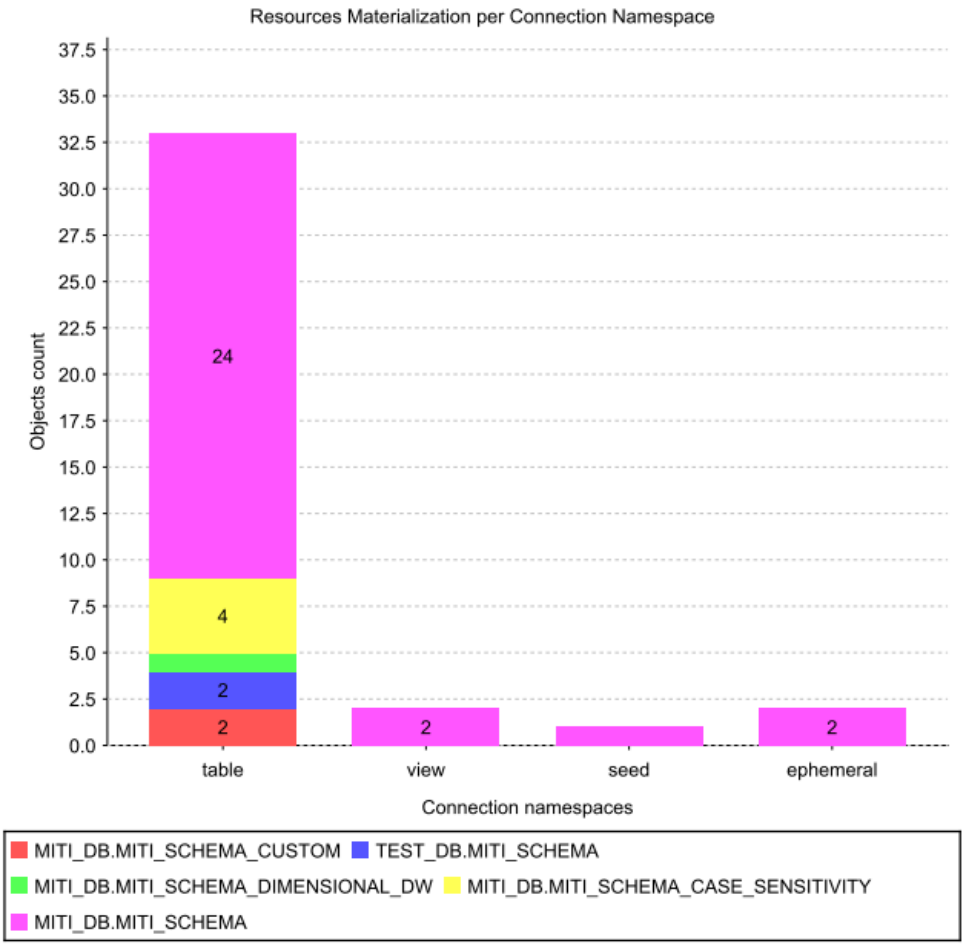

Рисунок 3.11 – Приклад візуального компоненту «Stacked Chart»

Усі ці компоненти разом створюють повноцінну BI-візуалізацію dbt-проєкту, що дозволяє аналітикам не лише переглядати, а й аналізувати, виявляти аномалії, оцінювати якість та прогнозувати потенційні проблеми в процесі побудови аналітичних моделей.

ВИСНОВКИ

У роботі було проведено дослідження, як працюють сучасні ETL-системи, зокрема інструмент dbt, та реалізувано власний аналітичний проєкт з використанням Google BigQuery. Основною метою було створити зручну, зрозумілу й ефективну структуру для обробки даних, яка дозволяє легко керувати процесом трансформації, тестування та розгортання моделей.

Було організувано моделі в кілька рівнів: `staging`, `transform`, `marts` та `distribute`. Кожен з них відповідає за свій етап роботи з даними - від первинної очистки до фінальної підготовки для звітності. Такий підхід допомагає краще розуміти, що відбувається на кожному етапі ETL-процесу, і спрощує подальше обслуговування та розвиток проєкту.

Під час розробки було використано різні ресурси dbt - моделі, тести, документацію, макроси та параметри. Було проаналізовано, як працює параметризація в dbt й був зроблений висновок наскільки вона корисна для налаштування моделей під різні умови. Також було порівняно типи баз даних, з якими працює dbt, і обрано BigQuery, адже він добре підходить для хмарної роботи з великими обсягами інформації.

Було створено Java-додаток з графічним інтерфейсом. У цьому додатку користувач може ввести необхідні параметри підключення до dbt Cloud, і програма автоматично збирає метадані про моделі, job-и та середовище. Після цього інформація обробляється й передається у звіт.

Застосована бібліотека JasperReports, яка дозволила сформувати зручні й зрозумілі документи, які можна зберігати, переглядати або надсилати іншим користувачам. Це дуже зручно, коли потрібно показати стан проєкту або результати роботи команди без доступу до самої бази чи хмарного сервісу.

У результаті було досягнуто повного циклу реалізації: від створення ETL-проєкту та запуску тестів у хмарному середовищі - до інтеграції з зовнішніми інструментами та створення фінальних звітів. Здобуті знання охоплюють сучасні підходи до побудови data pipeline-ів, а також специфіку

використання dbt у середовищах, де необхідна стабільна, прозора й масштабована трансформація даних у рамках системи бізнес-аналітики.

Загалом реалізована система може служити фундаментом для подальшого розвитку BI-рішень, автоматизації контролю якості даних, формування звітності для керівництва та інтеграції з іншими частинами data-екосистеми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Elaine R. User modeling via stereotypes. Cognitive Science. 1979. Vol. 3, no. 4. P. 329–354.

2. R. Kimball – M. Ross. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. 3rd ed. Wiley. 2013. 600 p.

3. M. Chambers. Designing Data-Intensive Applications. O'Reilly Media. 2017. 616 p.J. Mukherjee. Practical Business Intelligence. Apress. 2019. 260 p.

4. T. Jackson. JasperReports for Java Developers. Packt Publishing. 2007. 336 p.

5. dbt Labs. dbt Documentation. dbt Docs. URL: https://docs.getdbt.com/. Дата звернення: 04.04.2025.

6. dbt Labs. dbt Cloud API Reference. dbt Docs. URL: https://docs.getdbt.com/docs/dbt-cloud-apis/overview. Дата звернення: 08.03.2025.

7. Google. BigQuery Documentation. Google Cloud. URL: https://cloud.google.com/bigquery/docs. Дата звернення: 27.03.2025.

8. Google. Gson. A Java serialization/deserialization library. GitHub. URL: https://github.com/google/gson. Дата звернення: 12.04.2025.

9. TIBCO Software Inc. JasperReports Library. TIBCO Jaspersoft Community. URL: https://community.jaspersoft.com/project/jasperreports-library. Дата звернення: 06.04.2025.

10. Oracle. Java Platform, Standard Edition Documentation. Oracle Docs. URL: https://docs.oracle.com/en/java/. Дата звернення: 02.03.2025.

11. OpenJDK. JavaFX Documentation. OpenJFX.io. URL: https://openjfx.io/. Дата звернення: 05.04.2025.

12. dbt. ETL vs ELT: What's the Difference? getdbt.com. URL: https://www.getdbt.com/blog/etl-vs-elt. Дата звернення: 15.03.2025.

13. Informatica. What is ETL (Extract, Transform, Load)? Informatica Docs. URL: https://www.informatica.com/resources/articles/what-is-etl.html. Дата

звернення: 18.03.2025.

14. Microsoft. Business Intelligence – BI Definition & Tools. Microsoft Learn. URL: https://learn.microsoft.com/en-us/power-bi/fundamentals/ . Дата звернення: 01.04.2025.

15. Microsoft. SQL Server Integration Services. Microsoft Learn. URL: https://learn.microsoft.com/en-us/sql/integration-services/sql-server-integration-services?view=sql-server-ver17. Дата звернення: 21.03.2025.

16. IBM. What is Business Intelligence (BI)? URL: https://www.ibm.com/think/topics/business-intelligence/. Дата звернення: 03.04.2025.

17. Redgate. Parameterization in SQL and Reporting. Redgate Hub. URL: https://www.red-gate.com/simple-talk/databases/sql-server/t-sql-programming-sql-server/performance-implications-of-parameterized-queries/. Дата звернення: 25.03.2025.

18. Snowflake Inc. Snowflake Documentation. URL: https://docs.snowflake.com/. Дата звернення: 27.03.2025.

19. Snowflake inc. Snowflake vs Google BigQuery comparison. URL: https://snowflakemasters.in/snowflake-vs-bigquery/. Дата звернення: 27.03.2025.

20. Matillion. Matillion ETL Documentation. URL: https://docs.matillion.com/. Дата звернення: 22.03.2025.

21. Fivetran. Documentation. URL: https://fivetran.com/docs/. Дата звернення: 01.04.2025.

22. JasperReports. JasperReports charts documentation. URL: https://jasperreports.sourceforge.net/sample.reference/charts/README.html. Дата звернення: 07.04.2025.

23. dbt. Resource types documentation. URL: https://docs.getdbt.com/reference/global-configs/resource-type.

24. Oracle. Parameterization in ETL systems. URL: https://docs.oracle.com/cd/E15586_01/fusionapps.1111/e14849/dacparameters.htm .