https://doi.org/10.31891/2307-5732-2026-361-47
УДК 004.6

СОЛОГУБ ВОЛОДИМИР
Національний університет «Львівська політехніка»
https://orcid.org/0000-0003-1553-530X
e-mail: volodymyr.r.solohub@lpnu.ua
ПАШКЕВИЧ ВОЛОДИМИР
Національний університет «Львівська політехніка»
https://orcid.org/0000-0002-6849-652X
e-mail: volodymyr.z.pashkevych@lpnu.ua

МЕТОД ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ОБРОБКИ ETL/ELT ПРОЦЕСІВ НА ОСНОВІ МЕТАДАТА ПАРАМЕТРІВ

Статтю присвячено розв'язанню однієї з актуальних проблем сучасної інженерії даних — обмеженої гнучкості та високих витрат на супровід традиційних ETL/ELT-процесів. Основна проблема полягає у тому, що в поширених імперативних підходах логіка перенесення, трансформації та завантаження даних жорстко фіксується безпосередньо у великій кількості SQL-скриптів або програмного коду. Це зумовлює перетворення навіть незначних змін бізнес-вимог на складні інженерні завдання, що потребують внесення модифікацій до коду, проведення повторного тестування та розгортання.

У роботі розроблено та експериментально валідовано метод на основі метаданих, що дає змогу керувати процесами опрацювання даних т реалізує декларативний підхід до їхньої організації. Наукова новизна методу полягає у чіткому розмежуванні логічної конфігурації завдання та механізму його фізичного виконання. На відміну від існуючих підходів, у запропонованому рішенні логіка обробки даних повністю винесена з програмного коду у зовнішній шар метаданих. Цей шар представлено у вигляді системи реляційних таблиць, які декларативно описують параметри завдання: джерело даних приймач та стратегію завантаження (повне перезавантаження, інкрементальне оновлення, заміна секції).

Імперативний рівень виконання реалізовано у вигляді єдиного уніфікованого рушія, який не містить бізнес-логіки. Під час виконання рушій зчитує конфігурацію із таблиць метаданих та динамічно формує SQL-код для конкретного завдання. Такий підхід забезпечує можливість оперативної зміни поведінки пайплайнів — наприклад, переключення з інкрементального режиму на повне завантаження — шляхом модифікації одного запису в таблиці метаданих без необхідності втручання у програмний код чи інфраструктуру.

Запропонований метод має переваги не лише порівняно з традиційними сценаріями ETL, але й із сучасними low-code/no-code платформами (зокрема, Azure Data Factory), які, незважаючи на декларативність, часто обмежують гнучкість та призводять до залежності від конкретного постачальника програмного забезпечення. Експериментальні результати засвідчили, що застосування методу дає змогу скоротити час розроблення та модифікації процесів обробки даних більш ніж на 90% порівняно з традиційними методами. Підтверджено відсутність істотних накладних витрат на продуктивність та досягнення значного зниження сукупної вартості володіння (TCO).

Розроблений метод демонструє практичну реалізацію переходу від імперативної до декларативної парадигми в інженерії даних, забезпечуючи підвищення гнучкості, прозорості, масштабованості та економічної ефективності аналітичних систем.

Ключові слова: *ETL, ELT, метадані, декларативний підхід, інженерія даних, опрацювання даних, пайплайн даних, база даних, сховище даних*

SOLOHUB VOLODYMYR, PASHKEVYCH VOLODYMYR
Lviv Polytechnic National University

METHOD FOR INCREASING THE EFFICIENCY OF ETL/ELT PROCESSING BASED ON METADATA PARAMETERS

This paper addresses one of the key challenges in contemporary data engineering: the limited flexibility and high maintenance costs of traditional ETL/ELT processes. The issue arises from the fact that in common imperative approaches, the logic of data extraction, transformation, and loading is rigidly hard-coded within numerous SQL scripts or program modules. As a result, even minor changes in business requirements become complex engineering tasks requiring code modification, thorough testing, and redeployment.

The study proposes and validates a novel metadata-driven method for managing data processing workflows that employs a purely declarative paradigm. The scientific novelty of the approach lies in the fundamental separation of concerns between the logical configuration of a task and the mechanism of its physical execution. Unlike existing methods, the entire pipeline logic is externalized from the codebase into a dedicated metadata layer. This layer consists of specialized relational tables that declaratively define what should be executed — including the data source, target, and loading strategy (e.g., Full reload, Incremental update, or PartitionReplace mode).

The imperative execution layer is represented by a unified processing engine (implemented as a stored procedure, e.g., usp_Execute_Data_Movement_Task) that contains no embedded business logic. During execution, this engine dynamically reads configuration parameters from the metadata tables and generates the required SQL statements at runtime. Consequently, a major behavioral change — for instance, switching from incremental to full load mode — can be achieved through a simple update of a metadata field, without any modification to the underlying code or infrastructure.

The proposed approach demonstrates significant advantages over both traditional scripting and modern low-code/no-code platforms (e.g., Azure Data Factory), which, despite offering declarative design, often lead to vendor lock-in and reduced flexibility. Experimental validation has confirmed that the metadata-driven method reduces the time required for pipeline development and modification by more than 90% compared to conventional approaches. Moreover, it introduces negligible performance overhead while achieving a substantial reduction in total cost of ownership).

The results of this study illustrate a successful and practical transition from the imperative to the declarative paradigm in data engineering, enabling the creation of more flexible, transparent, scalable, and cost-efficient analytical systems.

Keywords: *ETL, ELT, metadata, declarative approach, data engineering, data processing, data pipeline, database, data warehouse*

Стаття надійшла до редакції / Received 06.12.2025
Прийнята до друку / Accepted 11.01.2026
Опубліковано / Published 29.01.2026

Вступ

В епоху цифрової трансформації ефективне управління даними стало ключовим фактором успіху для будь-якої організації. Основою сучасних аналітичних систем є процеси перенесення даних (ETL/ELT)[1,2], що забезпечують наповнення сховищ даними з різноманітних операційних джерел. Проте традиційний підхід до реалізації цих процесів, що спирається на імперативні, жорстко закодовані скрипти, стикається з низкою фундаментальних проблем. Серед них — низька гнучкість, високий рівень дублювання коду та відсутність централізованого управління, що значно ускладнює підтримку та розвиток аналітичних платформ.[3,4,5]

Зокрема, зміна стратегії завантаження даних — наприклад, тимчасовий перехід від інкрементального оновлення до повного перезавантаження для виправлення історичних аномалій — перетворюється на складне інженерне завдання, що вимагає модифікації коду та тривалого тестування.[6,7,8]

Для вирішення цих викликів у роботі пропонується метод метадано-керованої оркестрації процесів перенесення даних. Ключова ідея полягає в абстрагуванні фізичної реалізації пайплайну від його логічної суті. Замість того, щоб кодувати логіку, користувач декларативно описує її в метаданих: джерело, приймач, тип завантаження та його параметри. Це дає змогу централізовано керувати всіма потоками даних, динамічно змінювати їхню поведінку без втручання в код та значно підвищити швидкість розробки й надійність системи.

Метою цієї статті є розроблення, опис та валідація методу, а також демонстрація його переваг на практичних прикладах.

Аналіз останніх досліджень і публікацій

В основі сучасних процесів інтеграції даних лежать дві стратегії: ETL та ELT. Обидві оперують трьома незмінними етапами — отримання (Extract), трансформація (Transform) та завантаження (Load). Фундаментальна різниця між ними зводиться до послідовності цих дій: ETL передбачає трансформацію даних до їх завантаження в кінцеву систему, тоді як ELT змінює цей порядок, виконуючи трансформацію після. Ця відмінність не є формальністю; вона визначає всю схему потоку даних — від вимог до ресурсів до можливостей аналітичної обробки.[9,10,11]

ETL (Extract, Transform, Load) та ELT (Extract, Load, Transform) — це дві ключові парадигми інтеграції даних, фундаментальна різниця між якими полягає в моменті та місці виконання етапу трансформації. У той час як ETL передбачає перетворення даних на проміжному етапі до їх завантаження в цільовий репозиторій, ELT виконує трансформацію вже після завантаження сирих даних безпосередньо в кінцеву систему. Ця відмінність є визначальною для всієї схеми опрацювання даних.[12,13,14]

ETL (Extract, Transform, Load) — це класичний процес, що складається з трьох послідовних кроків:

- Отримання (Extraction): Збір сирих даних із різноманітних, часто розрізнених джерел.
- Перетворення (Transformation): Дані передаються на проміжний сервер, де вони очищуються, стандартизуються, збагачуються та приводяться до єдиної структури згідно з бізнес-вимогами.
- Завантаження (Loading): Уже підготовлені та структуровані дані завантажуються в цільову систему, наприклад, у корпоративне сховище даних .

Історично цей підхід був розроблений для роботи з реляційними базами даних та традиційними сховищами, які вимагають, щоб дані надходили вже у структурованому вигляді для ефективної підтримки аналітичних OLAP-запитів.[15,16]

Зі стрімким зростанням обсягів даних та посиленням потреби в їх швидкій обробці для бізнес-аналітики, набув популярності альтернативний підхід — ELT (Extract, Load, Transform).[17,18]

У цій моделі порядок дій змінюється: трансформація відбувається після завантаження даних безпосередньо в цільову систему.

- Отримання (Extraction): Дані, як і раніше, видобуваються з джерел.
- Завантаження (Loading): Сирі, необроблені дані завантажуються напряму в цільовий репозиторій (найчастіше — хмарне сховище або озеро даних).
- Перетворення (Transformation): Усі необхідні перетворення виконуються безпосередньо в цільовій системі з використанням її потужних обчислювальних ресурсів.

Такий підхід став можливим завдяки потужності сучасних хмарних сховищ та озер даних, здатних ефективно обробляти величезні масиви сирих даних. Таким чином, ELT усуває потребу в окремому сервері для трансформації, переносячи обчислювальне навантаження безпосередньо на кінцеву систему.[19]

Під час роботи з великими обсягами даних у процесах ETL вибір правильної стратегії завантаження даних є вирішальним. Два найпоширеніші підходи – це повне завантаження та інкрементальне завантаження. Розуміння відмінностей між ними допомагає забезпечити ефективну обробку даних, швидшу продуктивність та точну аналітику.[20,21,22]

Повне завантаження – це метод, за якого всі дані з джерела завантажуються в цільову систему, незалежно від того, чи були вони завантажені раніше. По суті, він стирає існуючі дані в цільовій системі та замінює їх повною, свіжою копією з джерела.[23]

Інкрементальне завантаження передає лише нові або змінені дані (на основі позначок часу, ідентифікаторів або відстеження змін) від джерела до місця призначення. Воно уникає перезавантаження всіх даних, тим самим заощаджуючи час і ресурси. Інкрементальне завантаження також називається дельта-завантаженням.[24]

Виділяють також метод завантаження із заміною партиції. Ця високоефективна техніка управління даними, що передбачає атомарну заміну цілого логічного сегмента (партиції) цільової таблиці новим, попередньо підготовленим набором даних. Технічно операція реалізується через створення проміжної таблиці, структура якої повністю ідентична цільовій, та її наповнення новими даними. Після цього виконується операція обміну партиціями, яка є маніпуляцією з метаданими на рівні сховища. Оскільки фізичного переміщення даних на цьому етапі не відбувається, операція виконується практично миттєво. Ключовими перевагами даного підходу є висока продуктивність, забезпечення узгодженості даних завдяки атомарності, та мінімальний час блокування цільової таблиці, що гарантує її високу доступність під час завантаження. Найбільш доцільним є застосування цього методу для періодичного повного оновлення великих, дискретних наборів даних, таких як щоденні або щомісячні зрізи. [25]

Таким чином, інженерія даних оперує різноманітними стратегіями завантаження — від простого інкрементального додавання записів (append) та злиття змін (merge/upsert) до повної заміни партицій (replace partition). Кожен із цих підходів є оптимальним для конкретних технічних завдань, проте в традиційних системах вибір та реалізація стратегії жорстко кодуються в логіці кожного окремого пайплайну. Це призводить до дублювання коду та втрати гнучкості. Звідси виникає нагальна потреба у створенні єдиного, уніфікованого методу, який би абстрагував фізичну реалізацію цих операцій. Ідеальна система повинна надавати користувачеві можливість декларативно, на основі метаданих, обирати та конфігурувати найбільш доцільний метод завантаження для кожного потоку даних, забезпечуючи гнучкість, централізоване управління та легкість супроводу всієї аналітичної платформи.

Виклад основного матеріалу дослідження

Традиційні ETL/ELT процеси часто покладаються на імперативні, жорстко закодовані скрипти, де логіка перенесення даних (наприклад, повне перезавантаження чи інкрементальне оновлення) є невід'ємною частиною коду.

Проблеми такого підходу:

- Низька гнучкість: Зміна стратегії завантаження (наприклад, з інкрементальної на повну для виправлення історичних даних) вимагає значних змін у коді та втручання розробника.
- Дублювання коду: Схожа логіка для різних стратегій повторюється у багатьох пайплайнах.
- Відсутність централізованого управління: Немає єдиного місця, де можна побачити та керувати всіма стратегіями завантаження даних в системі.

Мета роботи — розробити та валідувати метод, який абстрагує фізичну стратегію перенесення даних від її логічної суті, дозволяючи користувачам керувати нею через метадані.

Ключова відмінність розробленого методу від традиційних підходів полягає у фундаментальному переході від імперативної до декларативної парадигми управління даними. У той час як традиційні системи покладаються на жорстко закодовані скрипти, де логіка та стратегія перенесення (наприклад, повне чи інкрементальне завантаження) є невід'ємною частиною коду, розроблений метод повністю відділяє логічну суть завдання від його фізичної реалізації. Завдяки цьому, зміна стратегії завантаження (наприклад, тимчасовий перехід від Incremental до Full для виправлення даних) перетворюється зі складного інженерного завдання, що вимагає модифікації коду, на просту конфігураційну операцію — оновлення одного поля в таблиці метаданих.

В основі розробленого методу лежить принцип розділення відповідальності, який чітко розмежовує декларативну конфігурацію від імперативної логіки виконання. Цей принцип реалізовано через два основні, тісно пов'язані компоненти: шар метаданих та виконання.

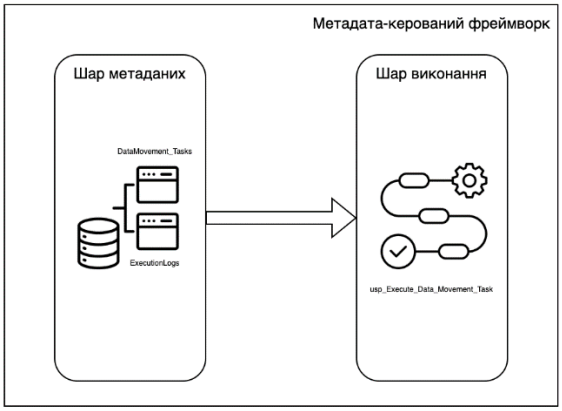

Рис. 1. Схема розробленого методу

Шар метаданих є декларативним ядром та "єдиним джерелом правди" для всього методу. Він реалізований у вигляді набору реляційних таблиць у базі даних, що забезпечує транзакційну цілісність, структурованість та можливість легкого доступу для аналізу. Цей шар описує всі аспекти завдань з перенесення даних, але не містить логіки їх фізичного виконання. Розглянемо детальніше його компоненти.

Таблиця конфігурації завдань (DataMovement_Tasks) - це головна конфігураційна таблиця, що реєструє кожне завдання з перенесення даних та його параметри.

1. TaskID (PK): Унікальний цілочисельний ідентифікатор, що слугує первинним ключем та забезпечує унікальність кожного завдання.
2. SourceSchema, SourceTable: Текстові поля, що однозначно визначають схему та назву таблиці-джерела даних.
3. TargetSchema, TargetTable: Аналогічні поля, що визначають таблицю-приймач.
4. LoadStrategy: Ключове поле типу, що визначає одну з реалізованих у методу стратегій завантаження. Наприклад: Full (повне перезавантаження), Incremental (інкрементальне оновлення), PartitionReplace (заміна партиції). Використання переліку стандартизує вибір та запобігає помилкам конфігурації.
5. StrategyParameters: Поле типу JSON, що містить специфічні для кожної стратегії параметри. Вибір формату JSON обґрунтований його гнучкістю та розширюваністю, що дає змогу легко додавати нові стратегії та параметри без зміни структури таблиці. Приклад для Incremental: {"IncrementalKey": "ModifiedDate", "Watermark": "2025-09-24T10:00:00Z"}. Приклад для PartitionReplace: {"PartitionKey": "OrderDate", "PartitionValue": "2025-09-24"}.
6. IsActive: Булевий прапорець, що дає змогу вмикати та вимикати завдання без фізичного видалення його конфігурації. Це спрощує управління та тимчасову деактивацію потоків даних.

Таблиця аудиту та моніторингу (DataMovement_ExecutionLogs) - ця таблиця виконує критично важливу функцію аудиту, моніторингу та зневадження, зберігаючи історію виконання кожного завдання.

- LogID (PK): Унікальний ідентифікатор запису логу.
- TaskID (FK): Зовнішній ключ, що пов'язує запис логу з конкретним завданням у таблиці DataMovement_Tasks.
- ExecutionStart, ExecutionEnd: Поля типу TIMESTAMP, що фіксують точний час початку та завершення виконання завдання, дозволяючи аналізувати його тривалість.
- Status: Поле типу ENUM зі значеннями Success, Failure, що надає швидку оцінку результату виконання.
- RowsAffected: Числове поле, що зберігає кількість оброблених (вставлених, оновлених, видалених) рядків. Цей показник є важливим для виявлення аномалій (наприклад, нульова кількість рядків при очікуваному оновленні).
- LogMessage: Текстове поле для збереження детальної інформації, зокрема повідомлень про помилки та стеку викликів у разі невдалого виконання.

Нижче буде подано ER-діаграму метадата таблиць, які було описано вище.

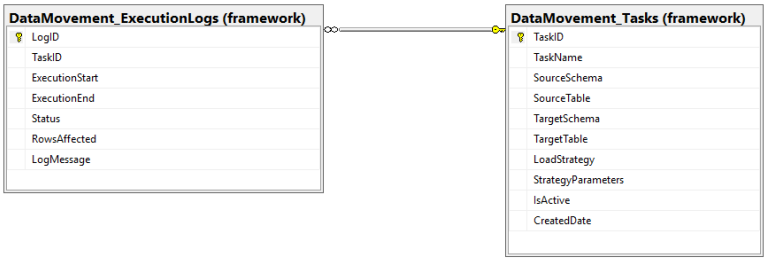

Рис. 2. ER-діаграму метадата таблиць

Процедурний компонент системи відповідає за інтерпретацію метаданих та виконання операцій перенесення даних. Він реалізований у вигляді універсальної, незмінної збереженої процедури usp_Execute_Data_Movement_Task, яка виступає єдиною точкою входу. Вхідним параметром процедури є ідентифікатор завдання @TaskID, на основі якого з таблиці DataMovement_Tasks зчитуються всі необхідні параметри. На початковому етапі виконання формується запис у таблиці DataMovement_ExecutionLogs зі статусом «In Progress». Далі, залежно від значення параметра LoadStrategy, обирається відповідна гілка логіки, що передбачає динамічну генерацію SQL-коду:

- Full load – формування команд TRUNCATE TABLE та INSERT INTO … SELECT …;
- Incremental load – використання оператора MERGE або конструкцій UPDATE/INSERT з урахуванням параметра IncrementalKey;
- PartitionReplace – створення тимчасової таблиці, її наповнення та виконання операції ALTER TABLE … SWITCH PARTITION ….

Після завершення обчислень процедура оновлює запис у DataMovement_ExecutionLogs, фіксуючи фінальний статус виконання, кількість оброблених рядків та текстове повідомленн

Процес використання розробленого методу має чітко визначену послідовність етапів. На етапі декларації користувач (аналітик або інженер) формує новий запис у таблиці DataMovement_Tasks, визначаючи джерело даних, цільове середовище та стратегію завантаження. На етапі виконання зовнішній оркестратор ініціює виклик збереженої процедури usp_Execute_Data_Movement_Task, передаючи ідентифікатор відповідного завдання. Подальша інтерпретація метаданих забезпечується рушієм, який, на основі отриманої

інформації, переходить до етапу дії, виконуючи відповідні операції з даними. Завершальним етапом є аудит, під час якого результати виконання фіксуються у логах.

Важливо, що для зміни стратегії завантаження даних у вже існуючому завданні користувачу достатньо оновити лише одне поле у таблиці DataMovement_Tasks, що значно спрощує управління процесом.

Для наочної демонстрації динамічної взаємодії між описаними компонентами, розглянемо діаграму послідовності (Рис. 3). Якщо схема показує статичну структуру методу, то ця діаграма розкриває логіку його роботи в часі. На ній детально проілюстровано повний життєвий цикл обробки одного завдання: починаючи від ініціації процесу зовнішнім планувальником, через отримання конфігурації з шару метаданих, виконання операцій з даними, і закінчуючи записом фінального статусу в лог. Ця візуалізація дає змогу чітко простежити потік керування та даних, а також зрозуміти роль кожного компонента в загальному процесі.

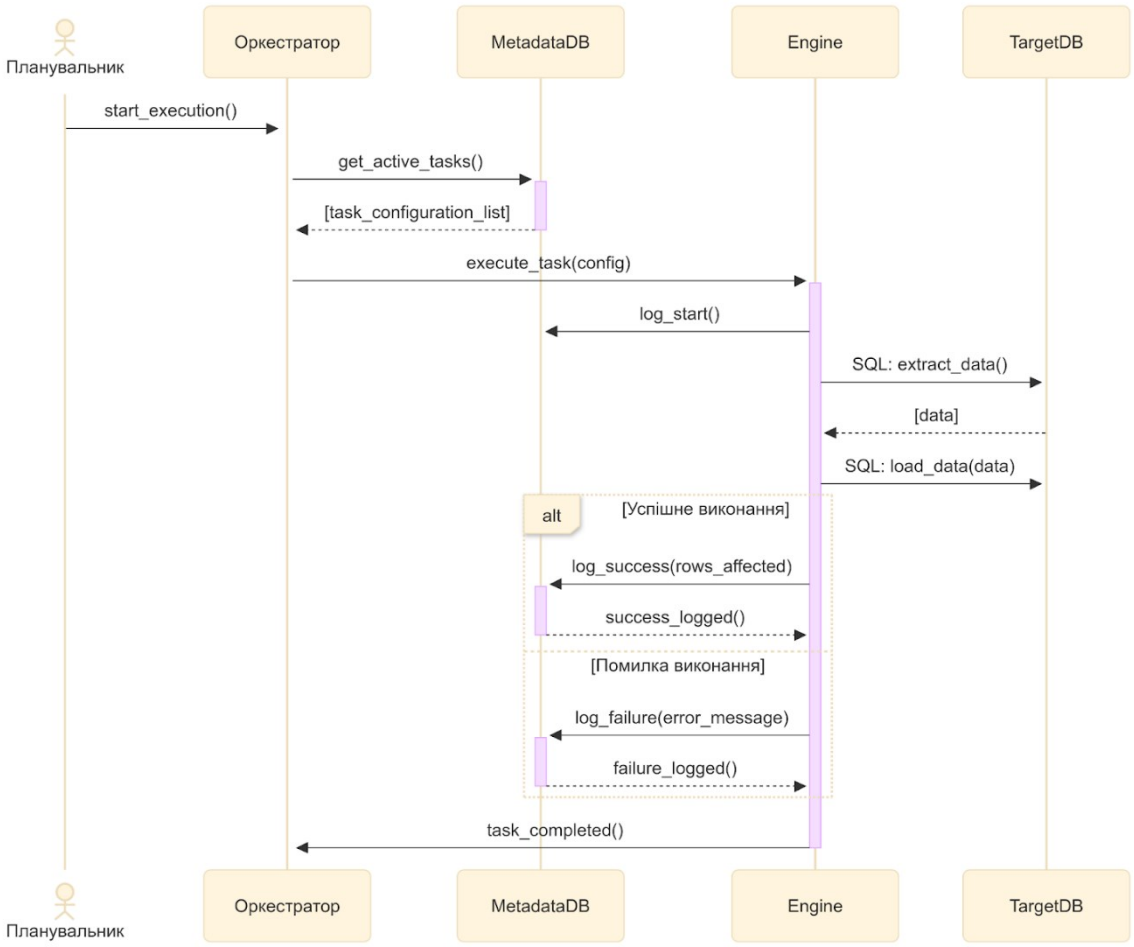

Рис. 3. Схема взаємодії між компонентами методу

На рисунку 4 представлена діаграма операційного потоку, де кожна доріжка відповідає за окремого учасника процесу: Користувача, Зовнішнього Оркестратора, Рушія Виконання та шару метаданих. Ця візуалізація чітко розмежовує ручні та автоматизовані етапи, демонструючи повний життєвий цикл завдання. Процес починається з декларативної дії Користувача, ініціюється автоматизованим викликом від Оркестратора, після чого Рушій Виконання вступає у двосторонню взаємодію з шаром метаданих: спочатку для зчитування конфігурації, а наприкінці — для запису результату операції в лог.

Таким чином, діаграма підкреслює центральну роль шару метаданих як точки входу для конфігурації та точки виходу для аудиту.

На рисунку 5 представлено алгоритм методу, що детально ілюструє роботу збереженої процедури usp_Execute_Data_Movement_Task, яка є практичною реалізацією Рушія Виконання. Алгоритм починається з отримання ідентифікатора завдання (TaskID), валідації його метаданих та створення початкового запису в лозі виконання.

Центральним елементом логіки є блок прийняття рішень, який аналізує параметр LoadStrategy і направляє потік до однієї з трьох гілок, що реалізують відповідні стратегії завантаження: Full, Incremental або PartitionReplace. Незалежно від обраної стратегії, робота процедури завершується оновленням запису в лозі, фіксуючи фінальний статус (Success або Failure в межах блоку TRY...CATCH) та кількість оброблених рядків, що забезпечує повний аудит кожної операції.

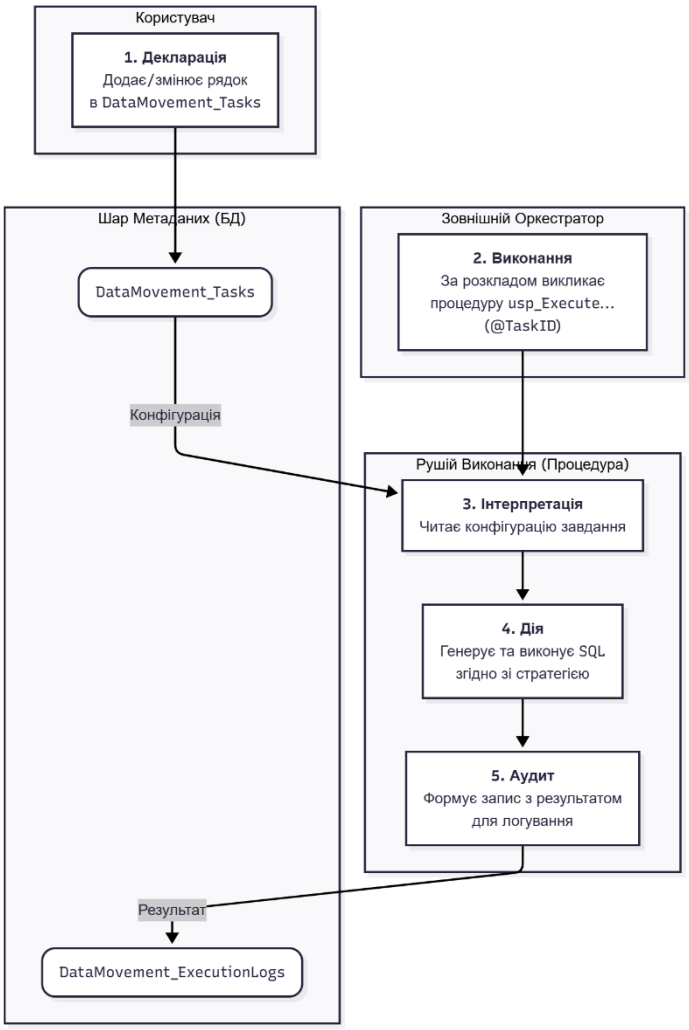

Рис. 4. Діаграма операційного потоку

Наведемо математичний опис розробленого алгоритму, позначеного як процедура З. Цей алгоритм керує переміщенням даних між таблицею-джерелом S, що є множиною кортежів $\{s_1, s_2, \ldots, s_m\}$, та таблицею-приймачем T, що є множиною кортежів $\{t_1, t_2, \ldots, t_n\}$. Кожен кортеж x (де x ϵ S або x ϵ T) характеризується набором атрибутів $A = \{A_1, A_2, \ldots, A_k\}$ та має унікальний ідентифікатор id(x). Керування цим процесом здійснюється на основі таблиці метаданих M (що відповідає DataMovement.Tasks), яка являє собою множину окремих завдань μ. Кожне завдання μ ϵ M визначається як кортеж $(\mu_{id}, S, T, L, N, \alpha)$. У цьому кортежі μ_{id} ϵ унікальним ідентифікатором завдання; S та T - це, відповідно, таблиці джерела та приймача; L - це стратегія завантаження, що належить до множини {{'Full'}, {'Incremental'}, {'PartitionReplace'}}; N містить параметри обраної стратегії у форматі JSON; α - це булевий прапор активності завдання, що приймає значення 0 або 1.

Алгоритм є процедурою P(τid), яка виконує такі кроки:

1. Ініціалізація:
   - Знайти завдання τ ∈ M таке, що його ідентифікатор співпадає з вхідним параметром τid.
   - Якщо τ не знайдено або α=0, завершити виконання з помилкою.
2. Логування:
   - Створити запис у журналі виконання ExecutionLogs зі статусом 'In Progress'.
3. Виконання Стратегії:
   - Виконати операцію для оновлення стану таблиці-приймача T до нового стану T′, залежно від стратегії L.
4. Завершення Логування:
   - У разі успіху: оновити запис у журналі, встановивши статус 'Success' та кількість оброблених рядків.
   - У разі помилки: оновити запис у журналі, встановивши статус 'Failure' та повідомлення про помилку.

Експериментальне порівняння розробленого методу з традиційним підходом (набір окремих скриптів) проводилося за трьома ключовими метриками. Отримані результати підтвердили висунуту гіпотезу та продемонстрували переваги розробленого методу.

Час розробки та гнучкість - ця метрика оцінює трудовитрати, необхідні для створення нового пайплайну та для зміни його логіки (гнучкість). Час вимірювався в хвилинах, що включали написання коду/конфігурації, тестування та підготовку до розгортання.

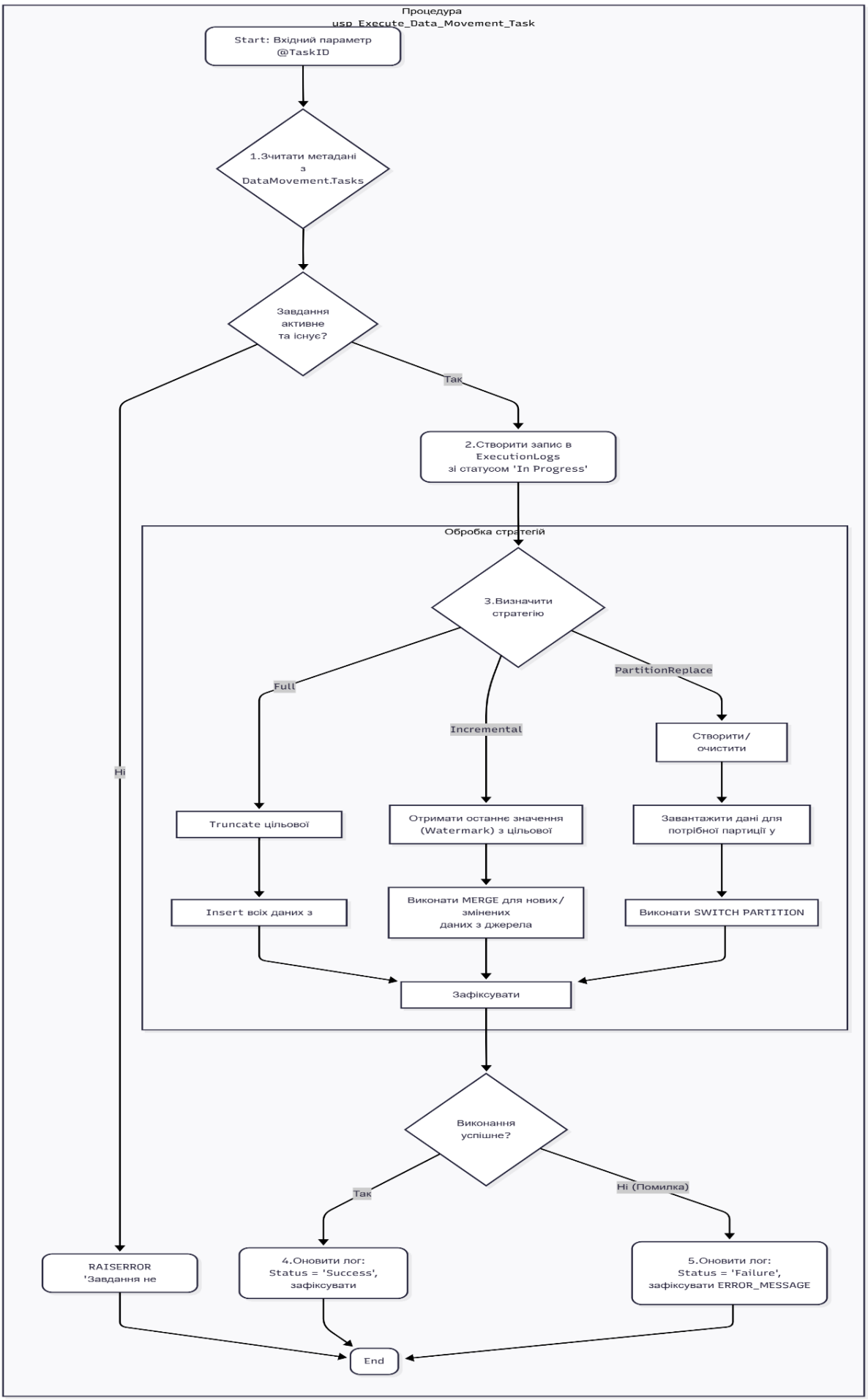

Рис. 5. Алгоритм реалізації методу

Таблиця 1

Порівняння затраченого часу при різних підходах

Операція	Традиційний підхід (час, хв.)	метод (час, хв.)	Скорочення часу (%)
Створення нового 'Full' пайплайну	~35	~2	~94.3%
Створення нового 'Incremental' пайплайну	~60	~3	~95.0%
Зміна стратегії з 'Full' на 'Incremental'	~45	~<1	~>98.0%

Згідно з результатами, які зображено в таблиці, метод забезпечує кардинальне скорочення часу на розробку та підтримку. Це пояснюється фундаментальною зміною парадигми:

- Традиційний підхід вимагає інженерної роботи: написання SQL-коду, тестування логіки, створення та розгортання окремих файлів.
- розроблений метод перетворює ці завдання на прості конфігураційні операції. Створення нового пайплайну — це INSERT одного рядка в таблицю метаданих, а зміна стратегії — UPDATE одного поля.

Таким чином, підтверджено, що абстрагування логіки в метадані дає змогу підвищити швидкість розробки та гнучкість системи, мінімізуючи залежність від інженерних ресурсів для виконання типових операційних завдань.

Продуктивність виконання - ця метрика оцінює час виконання операцій з перенесення даних для наборів різного розміру. Метою було перевірити, чи не створює метод значних накладних витрат порівняно з традиційними скриптами.

Таблиця 2

Час виконання запиту

Набір даних / Стратегія	Традиційний підхід (сек.)	Розроблений Метод (сек.)	Різниця (%)
Малий (1 млн) / Full	15.2	15.3	-0.7%
Середній (50 млн) / Full	710.5	711.2	-0.1%
Середній (50 млн) / Incremental (5% змін)	45.8	43.9	+4.1%
Великий (500 млн) / Incremental (5% змін)	485.1	462.5	+4.7%
Великий (500 млн) / PartitionReplace	25.4	24.1	+5.1%

На основі даних з таблиці, можна зробити висновок, що розроблений метод не поступається, а в деяких випадках навіть перевершує традиційний підхід за продуктивністю:

Відсутність накладних витрат: Для Full завантажень, де виконується ідентичний SQL-код (TRUNCATE/INSERT), різниця в часі є мінімальною (<1%). Це свідчить про те, що час, витрачений на зчитування метаданих, є незначним і не впливає на загальну продуктивність.

Перевага стандартизації: Для більш складних стратегій (Incremental, PartitionReplace) метод показав невеликий, але стабільний приріст швидкості (4-5%). Це пояснюється тим, що рушій генерує код на основі стандартизованих та попередньо оптимізованих SQL-шаблонів. Це виключає ймовірність написання менш ефективного коду вручну та гарантує використання оптимальних планів виконання запитів, особливо на великих обсягах даних.

Отже, експеримент підтверджує, що метод не створює затримок за продуктивністю, а навпаки, може слугувати гарантом стабільної та ефективної роботи завдяки стандартизації.

Окрім технічних переваг, було проведено оцінку економічної переваги методу порівняно з підходом, що базується на інтеграції кількох комерційних інструментів

Таблиця 3

Порівняння економічної переваги методу

Аспект вартості	**Комерційні інструменти**	**Розроблений метод**
Ліцензування	Високі щорічні/щомісячні ліцензійні платежі, що зростають з об'ємом даних.	Відсутні. Рішення є внутрішньою розробкою.
Інфраструктура	Потреба в окремих серверах/хмарних сервісах для ETL-інструмента та оркестратора.	Мінімальні витрати. Використовує існуючі обчислювальні ресурси бази даних.

Продовження таблиці 3

Аспект вартості	Комерційні інструменти	Розроблений метод
Інтеграція та підтримка	Високі витрати на інтеграцію різних систем та підтримку складного стеку технологій.	Низькі витрати. Єдина система, що спрощує підтримку та усуває потребу в інтеграції.
Навчання персоналу	Потреба в навчанні інженерів роботі з кількома спеціалізованими інструментами.	Мінімальні витрати. Інженерам потрібно знати лише SQL та принципи роботи з метаданими.

Як показує аналіз, розробка власного метадано-керованого методу забезпечує значну економію в довгостроковій перспективі. Основні фактори економії:

- Нульові ліцензійні витрати: метод усуває необхідність у дорогих підписках на комерційні ETL-продукти, що є однією з найбільших статей витрат у багатьох data-платформах.
- Використання ресурсів: Рішення є "легковагим" і працює безпосередньо в середовищі бази даних, не вимагаючи розгортання та оплати додаткової інфраструктури, на відміну від багатьох зовнішніх інструментів.
- Спрощення підтримки: Найбільша прихована вартість комерційних рішень полягає у складності їх інтеграції та подальшої підтримки. Уніфікований підхід методу кардинально знижує цю складність.

Таким чином, хоча початкова розробка методу вимагає інвестицій часу, його сукупна вартість володіння є значно нижчою, що робить його стратегічно вигідним рішенням для організації.

Висновок

У даній роботі розглянуто та вирішено проблему низької гнучкості та надмірної складності супроводу традиційних ETL/ELT процесів, що базуються на імперативному, жорстко закодованому підході. Як рішення, було розроблено та валідовано метадано-керований метод для уніфікованого перенесення даних. У статті детально описано метод, що складається з декларативного шару метаданих та імперативного виконання, а також представлено алгоритми роботи та операційний потік. Ключовою ідеєю методу є розділення опису завдання від його фізичної реалізації.

Експериментальна валідація повністю підтвердила висунуту гіпотезу. Було доведено, що розроблений підхід дає змогу кардинально (на >90%) скоротити час на розробку та модифікацію пайплайнів, перетворюючи інженерну роботу на просту конфігураційну операцію. Водночас метод не створює значних накладних витрат на продуктивність, а в деяких випадках навіть покращує її завдяки використанню стандартизованих SQL-шаблонів. Аналіз сукупної вартості володіння також показав значну економічну перевагу методу над інтеграцією комерційних аналогів.

Таким чином, стаття демонструє успішний перехід від імперативної парадигми до декларативної в галузі інженерії даних. розроблений метод дає змогу підвищити рівень автоматизації, знизити залежність від розробників для виконання операційних завдань та побудувати більш гнучку й масштабовану аналітичну схему

Література

1. Abadi, D. J., et al. (2009). The Beckman report on database research. SIGMOD Record, 37(4), 61–70. https://doi.org/10.1145/1519103.1519114
2. Apache Software Foundation. (2024). Apache Airflow Documentation. https://airflow.apache.org/docs/
3. Armbrust, M., et al. (2020). The lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics [White paper]. Databricks. https://www.databricks.com/resources/whitepapers/lakehouse-a-new-generation-of-open-platforms
4. Atif, M. (2023, January 20). The rise of active metadata: What it is, what it isn't, and why it's the future. Towards Data Science. https://towardsdatascience.com/the-rise-of-active-metadata-what-it-is-what-it-isnt-and-why-its-the-future-19269986b54a
5. Bannister, R. (2022, November 15). Declarative vs. imperative data engineering. Firebolt Blog. https://www.firebolt.io/blog/declarative-vs-imperative-data-engineering
6. Böhm, C., et al. (2021). The data preparation wall of shame: Towards a FAIR data processing framework. In Datenbanksysteme für Business, Technologie und Web (BTW) (pp. 317-336).
7. Dagster Labs. (2024). Dagster concepts: Assets. https://docs.dagster.io/concepts/assets
8. dbt Labs. (2024). What is dbt? https://docs.getdbt.com/docs/introduction
9. Dehghani, Z. (2022). Data mesh: Delivering data-driven value at scale. O'Reilly Media.
10. Fowler, M. (2019, March 25). How to move beyond a monolithic data lake to a distributed data mesh. MartinFowler.com. https://martinfowler.com/articles/data-monolith-to-mesh.html
11. Gartner, Inc. (2021). Augmented data catalogs: Now an enterprise must-have for data and analytics leaders.
12. Gonzalez, M., et al. (2021, February 17). What is data observability? Monte Carlo Data Blog. https://www.montecarlodata.com/what-is-data-observability/

13. Great Expectations. (2024). Core concepts. https://docs.greatexpectations.io/docs/reference/core_concepts

14. Halevy, A., et al. (2016). Goods: Organizing Google's datasets. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16) (pp. 795–806). Association for Computing Machinery. https://doi.org/10.1145/2882903.2903735

15. HashiCorp. (2024). Terraform Documentation. https://www.terraform.io/docs

16. Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley Professional.

17. Interlandi, M., et al. (2015). Tiresias: A data placement advisor for geo-distributed cloud data stores. Proceedings of the VLDB Endowment, 8(13), 2040–2051. https://doi.org/10.14778/2824032.2824101

18. Kimball, R., & Ross, M. (2013). The data warehouse toolkit: The definitive guide to dimensional modeling (3rd ed.). Wiley.

19. Kleppmann, M. (2017). Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media.

20. Microsoft. (2024). Azure Data Factory documentation - Mapping data flows. https://docs.microsoft.com/en-us/azure/data-factory/concepts-data-flow-overview

21. Prefect Technologies, Inc. (2024). Prefect Documentation. https://docs.prefect.io/

22. Reis, J., & Housley, M. (2022). Fundamentals of data engineering: Plan and build robust data systems. O'Reilly Media.

23. Schrock, N. (2021, July 21). Software-defined assets. Dagster Blog. https://dagster.io/blog/software-defined-assets

24. Snowflake Inc. (2023). Dynamic tables: Continuous, automated, and declarative data pipelines. https://docs.snowflake.com/en/user-guide/dynamic-tables-about

25. Uber Engineering Blog. (2019, January 24). Piper: Uber’s centralized workflow orchestrator. https://www.uber.com/en-UA/blog/piper/