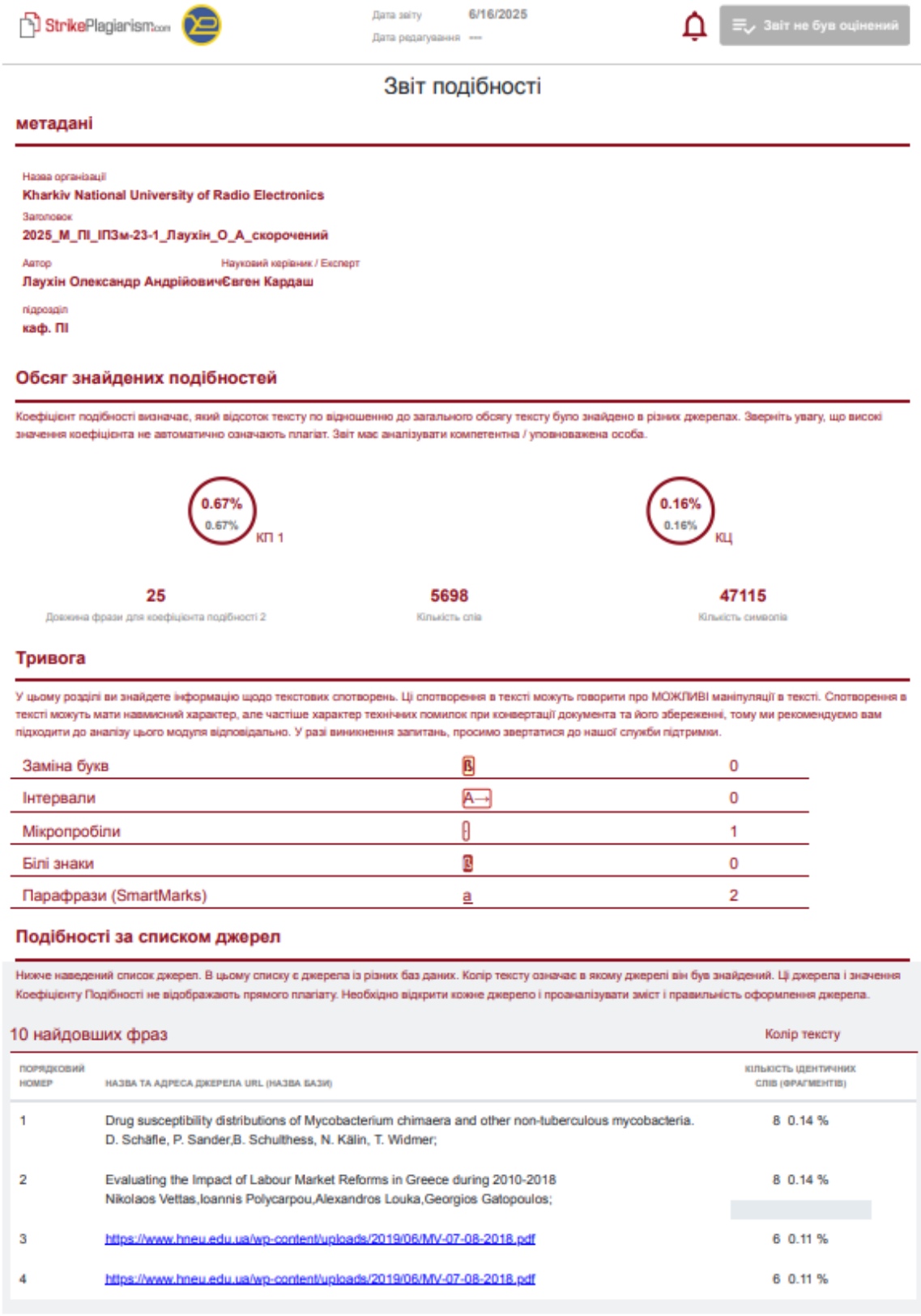
ДОДАТОК А

ЗВІТ РЕЗУЛЬТАТІВ ПЕРЕВІРКИ НА УНІКАЛЬНІСТЬ ТЕКСТУ В БАЗІ ХНУРЕ



StrikePlagiarism.com

Дата звіту 6/16/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_М_ПІ_ІПЗм-23-1_Лаухін_О_А_скорочений
Автор
Лаухін Олександр Андрійович
Науковий керівник / Експерт
Євген Кардаш
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

0.67% 0.67% КП 1

0.16% 0.16% КЦ

25
Довжина фрази для коефіцієнта подібності 2

5698
Кількість слів

47115
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		0
Інтервали		0
Мікропробіли		1
Білі знаки		0
Парафрази (SmartMarks)		2

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Drug susceptibility distributions of Mycobacterium chimaera and other non-tuberculous mycobacteria. D. Schäfle, P. Sander,B. Schulthess, N. Kälin, T. Widmer;	8 0.14 %
2	Evaluating the Impact of Labour Market Reforms in Greece during 2010-2018 Nikolaos Vettas,Ioannis Polycarpou,Alexandros Louka,Georgios Gatopoulos;	8 0.14 %
3	https://www.hneu.edu.ua/wp-content/uploads/2019/06/MV-07-08-2018.pdf	6 0.11 %
4	https://www.hneu.edu.ua/wp-content/uploads/2019/06/MV-07-08-2018.pdf	6 0.11 %

ДОДАТОК Б

СЛАЙДИ ПРЕЗЕНТАЦІЇ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ДОСЛІДЖЕННЯ МЕТОДІВ ОЦІНЮВАННЯ ТА ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ JAVA-ФРЕЙМВОРКІВ ДЛЯ ОБРОБКИ ВЕЛИКИХ ОБСЯГІВ ДАНИХ

Лаухін О.А., ІПЗм-23-1
Науковий керівник: проф. Власенко Л.А.

software engineering

23 червня 2025

Дослідження

Актуальність:

- Обсяги даних подвоюються кожні 2 роки;
- Java-фреймворки потребують оптимального налаштування продуктивності;
- Правильний вибір фреймворку критично важливий.

Стан розвитку галузі:

- Активні дослідження методів оптимізації продуктивності;
- Використання статистичних і когнітивних моделей;
- Популяризація цифрових інструментів оцінювання.

software engineering

Дослідження

Напрям дослідження:

- Розробка методів оцінювання продуктивності Java-фреймворків;
- Формування багатокритеріальної моделі порівняння;
- Орієнтація на оптимізацію обробки великих даних.

Об’єкт дослідження:

- Процеси обробки великих обсягів даних;
- Методи оцінки продуктивності фреймворків;
- Підходи до оптимізації конфігурацій;
- Аналіз Apache Spark, Kafka, Storm, Spring Batch;
- Оцінювання масштабованості систем.

Огляд літератури

Ключові джерела:

- Ishizaki K. & Liang, K. "Analyzing and Optimizing Java Code Generation for Apache Spark Query Plan" — роль оптимізації генерації коду та JIT-компіляції у підвищенні продуктивності;
- Ekanayake, S., Kamburugamuve, S. "Java thread and process performance for parallel machine learning" — вплив паралельних обчислень на ефективність обробки великих даних;
- Zeuch, S., Breß, S., Rabl, T. "Analyzing Efficient Stream Processing on Modern Hardware" — необхідність врахування апаратної архітектури при оцінці продуктивності;
- Song, W. W., Yang, Y. "Apache Nemo: A Framework for Optimizing Distributed Data Processing" — проблема адаптивної оптимізації та вплив методів конфігурації на результати.

Огляд літератури

Основні теоретичні підходи:

- Оптимізація генерації коду
  Використовується для підвищення продуктивності фреймворків, що обробляють великі дані, наприклад, методи JIT-компіляції в Apache Spark, що дозволяє прискорити виконання аналітичних запитів на основі динамічної оптимізації;
- Багатокритеріальне оцінювання
  Теорія багатокритеріального аналізу є основою порівняння фреймворків, де система враховує множину показників продуктивності залежно від типу навантаження. Це дозволяє об'єктивно визначати оптимальні рішення;
- Аналіз продуктивності розподілених систем
  Використовується для виявлення закономірностей у поведінці фреймворків, ефективності масштабування та використання ресурсів. Сюди входять методи профілювання, бенчмаркінгу та моделювання навантаження.

Огляд літератури

Виявлені прогалини:

- Недостатня інтеграція різних методів оцінювання продуктивності в єдину комплексну модель;
- Обмежений підхід до вимірювання ефективності в реальних умовах експлуатації;
- Обмежений рівень об'єктивності порівняльного аналізу фреймворків;
- Низька прозорість та інтерпретованість результатів оптимізації для розробників і архітекторів систем.

Постановка задачі

Проблема:

Відсутність комплексної методики для об'єктивного порівняння та оптимізації Java-фреймворків при роботі з великими обсягами даних.

Очікувані результати:

- Створення системи критеріїв для комплексного аналізу ефективності фреймворків;
- Розробка багатокритеріальної моделі вибору оптимального рішення;
- Формування практичних рекомендацій з налаштування та конфігурації систем;
- Впровадження тестової методології для валідації результатів дослідження в реальних умовах експлуатації.

Методологія

Використані методи дослідження:

- Систематичний огляд літератури – вивчення існуючих підходів до оцінювання продуктивності фреймворків для обробки великих даних;
- Порівняльний аналіз – співставлення характеристик різних Java-фреймворків за встановленими критеріями;
- Багатокритеріальне оцінювання – застосування принципу Парето та адитивної згортки для об'єктивного ранжування альтернатив;
- Експериментальне тестування – практична перевірка продуктивності фреймворків у контрольованому середовищі з використанням реальних наборів даних.

Інструментарій та технології

- Java 17 + фреймворки – платформа для реалізації тестових сценаріїв;
- JMeter + JMC – збір метрик продуктивності та профілювання систем;
- Apache Spark, Kafka, Storm, Spring Batch – досліджувані фреймворки для обробки даних;
- DigitalOcean кластер (Droplets) – інфраструктура для проведення навантажувальних тестів.

Зміст проведеного дослідження

Методи:

- Систематичний огляд літератури;
- Порівняльний аналіз характеристик фреймворків
- Багатокритеріальне оцінювання альтернатив.

Вхідні дані:

- Публікації з оптимізації Java-систем;
- Технічна документація та бенчмарки фреймворків;
- Метрики продуктивності кожного рішення.

Зміст проведеного дослідження

Критерії:

- Продуктивність обробки;
- Масштабованість;
- Надійність;
- Зручність розробки;
- Функціональні можливості.

Послідовність:

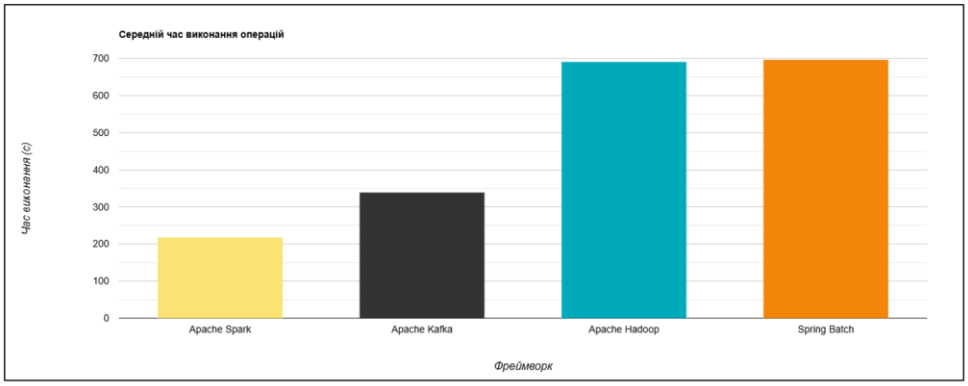
1. Відбір релевантних фреймворків;
2. Визначення метрик оцінювання;
3. Нормування критеріїв;
4. Застосування принципу Парето;
5. Проведення багатокритеріального аналізу.

Зміст проведеного дослідження

Вимірювання:

- Оцінка продуктивності кожного фреймворку;
- Виявлення Парето-оптимальних рішень;
- Ранжування альтернатив для практичного впровадження в системи.

Результати дослідження



Результати дослідження

Фреймворк	Агрегація	З'єднання	Потокова обробка	Пакетна трансформація
Apache Spark	282.3	566.5	64.1	814.7
Apache Kafka	748.6	-	34.8	1242.1
Apache Storm	843.7	1861.2	-	1665.3
Spring Batch	1064.4	1910.1	-	1241.8

Архітектура системи

Архітектура:

- Тестове середовище з модульною структурою для порівняння фреймворків:
  - модуль Apache Spark;
  - модуль Apache Kafka;
  - модуль Apache Storm;
  - модуль Spring Batch.
- Кластерна інфраструктура (DigitalOcean) – розгортання тестових навантажень, збір метрик продуктивності, моніторинг ресурсів.

Опис програмного забезпечення

Опис процесу розробки:

- Реалізація базується на результатах теоретичного дослідження: обрані найефективніші Java-фреймворки;
- Етапи розробки:
  - створення тестового середовища та конфігурації кластера;
  - налаштування інструментів моніторингу в Docker-контейнерах;
  - імплементація однотипних операцій обробки даних;
  - розробка модулів для Spark, Kafka, Storm та Spring Batch;
  - проведення експериментів та збір метрик продуктивності.

Публікація результатів

1 Міжнародна науково-практична конференція «СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ MIT@AIS-2025»

Підсумки

Результати теоретичного та експериментального досліджень демонструють різну ефективність Java-фреймворків для обробки великих даних, акцентуючи увагу на їхніх перевагах і недоліках у специфічних сценаріях, що не було висвітлено у попередніх роботах.

Розроблена методика тестування об'єднує критерії для Apache Spark, Kafka, Storm та Spring Batch, що забезпечує комплексне й об'єктивне порівняння підходів за розширеним набором метрик і заповнює прогалину щодо недостатності даних для повної оцінки.

Сформульовані практичні рекомендації дозволяють розробникам робити обґрунтований вибір ефективного інструменту для досягнення оптимальної продуктивності систем обробки даних, виходячи з домінуючих типів операцій:

- Apache Spark для комплексної аналітики та пакетної обробки;
- Apache Kafka для потокової обробки у реальному часі;
- Apache Storm для систем з обмеженими ресурсами пам'яті.

Дякую за увагу!

ДОДАТОК В

АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ

1 Міжнародна науково-практична конференція «СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ MIT@AIS-2025»

МЕТОДИ ОЦІНЮВАННЯ ТА ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ JAVA-ФРЕЙМВОРКІВ ДЛЯ ОБРОБКИ ВЕЛИКИХ ОБСЯГІВ ДАНИХ

Лаухін О. А.[1], Власенко Л. А.[1]

[1] *Харківський національний університет радіоелектроніки, пр. Науки, 14, м. Харків, 61166, Україна*

oleksandr.laukhin@nure.ua, larysa.vlasenko1@nure.ua

Ключові слова: великі дані, Java, фреймворки, оптимізація, продуктивність;

АНОТАЦІЯ

Було досліджено ефективність 4-х Java-фреймворків у контексті обробки великих обсягів даних - Apache Hadoop, Apache Spark, Apache Kafka та Spring Batch. Дослідження відбувалось в експериментальному середовищі, яке дозволило оцінити продуктивність фреймворків при виконанні таких типових операцій, як: агрегація, з'єднання, потокова обробка та пакетна трансформація.

Дослідження включало вимірювання метрик продуктивності: час виконання кожної з операцій, масштабування та стабільність роботи під навантаженням.

Результати експериментів показали, що Apache Spark є найкращим вибором у рамках загальної продуктивності обробки та масштабованості, Apache Kafka має найменші затримки під час роботи, Apache Hadoop є найбільш надійним, а Spring Batch має найнижчі результати з наведених параметрів оцінювання.

ПЕРЕДУМОВА

Сьогоденний етап розвитку інформаційних технологій вимагає обробки та аналізу більших обсягів даних, ніж раніше. За оцінками експертів, кількість даних у світі збільшується у 2 рази кожні 2 роки, що потребує більших ресурсів для їх опрацювання [1]. Для такого роду обробок даних, Java-фреймворки пропонують широкий асортимент інструментів, кожен з яких має свої переваги та недоліки, що виражаються в різних сценаріях використання [2].

Виникає питання вибору оптимального фреймворку, який забезпечив би максимальну продуктивність та ефективне використання наданих ресурсів.

МЕТА

Метою є дослідити ефективність Java-фреймворків в контексті обробки великих об'ємів даних методом порівняння їх продуктивності та визначити оптимальні налаштування для їх оптимізованої роботи в різних сценаріях використання.

МЕТОДИ

Було порівняно 4 Java-фреймворки: Apache Hadoop, Apache Spark, Apache Kafka та Spring Batch. Розроблено тестове середовище, де виконувались операції: агрегація, з'єднання, потокова обробка та пакетна трансформація даних. Тестування було проведено на кластері з 5 вузлів із використанням наборів даних різного обсягу (5-300 ГБ). Для кожного сценарію вимірювались час виконання операцій, ефективність масштабування та стабільність роботи під навантаженням [3]. Результати наведено у таблицях.

РЕЗУЛЬТАТИ

В даних таблиць показано, що для більшості операцій Apache Spark має найкращу продуктивність. Для потокової обробки Apache Kafka демонструє найнижчу затримку, а Apache Hadoop має найвищу надійність. Spring Batch показав найнижчі результати у масштабованості та обробці великих обсягів даних.

Таблиця 1. Середній час виконання операцій (секунди) для набору даних 50 ГБ.

Фреймворк	Агрегація	З'єднання	Потокова обробка	Пакетна трансформація
Apache Spark	142.3	286.5	32.1	412.7
Apache Kafka	378.6	-	17.8	623.1
Apache Hadoop	423.7	781.2	-	865.3
Spring Batch	534.4	932.1	-	621.8

Таблиця 2**.** Ефективність масштабування для різної кількості вузлів.

Фреймворк	2 вузли	4 вузли	8 вузлів	16 вузлів
Apache Spark	0.97	0.92	0.87	0.81
Apache Kafka	0.95	0.89	0.80	0.72
Apache Hadoop	0.93	0.85	0.74	0.62
Spring Batch	0.91	0.78	0.66	0.54

Також, після оптимізації параметрів JVM було досягнуто значне покращення продуктивності. Для Apache Spark середній час виконання операції агрегації зменшився на 17,3%, а частота GC-пауз зменшилась на 23,6%. Використання оптимальних стратегій з'єднання таблиць (broadcast join) дозволило скоротити час виконання операцій на 78% порівняно зі стандартним shuffle join для невеликих таблиць.

ВИСНОВКИ

На основі дослідження можна рекомендувати: для комплексної аналітики та пакетної обробки - Apache Spark, для потокової обробки в реальному часі - Apache Kafka, для систем з обмеженими ресурсами пам'яті - Apache Hadoop, а для простих пакетних завдань - Spring Batc.

ДЖЕРЕЛА

K. Ishizaki, Analyzing and Optimizing Java Code Generation for Apache Spark Query Plan, in: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, 2019, pp. 74-79. doi:10.1145/2961111.2962586.

S. Ekanayake, S. Kamburugamuve, P. Wickramasinghe, Java thread and process performance for parallel machine learning on multicore HPC clusters, in: 2016 IEEE International Conference on Big Data, 2016, pp. 3063-3072. doi:10.1109/BigData.2016.7840937.

S. Zeuch, S. Breß, T. Rabl, B. Del Monte, J. Karimov, C. Lutz, M. Renz, J. Traub, V. Markl, Analyzing Efficient Stream Processing on Modern Hardware, Proc. VLDB Endow. 12 (2019) 516-530. doi:10.14778/3311880.3311881.

ДОДАТОК Г

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008:2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-1 (група)

Олександр ЛАУХІН
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
7.1.25	Не дозволено розміщувати назву розділу, підрозділу, а також пункту й підпункту на останньому рядку сторінки.	43
	7.3 Нумерація сторінок звіту	
	7.5 Рисунки	
	7.6 Таблиці	
7.6.8	Назву таблиці друкують з великої літери і розміщують над таблицею з абзацного відступу.	39
7.6.9	Якщо рядки або колонки таблиці виходять за межі формату сторінки, таблицю поділяють на частини, розміщуючи одну частину під іншою або поруч, чи переносять частину таблиці на наступну сторінку. У кожній частині таблиці повторюють її головку та боковик. У разі поділу таблиці на частини дозволено її головку чи боковик заміняти відповідно номерами колонок або рядків, нумеруючи їх арабськими цифрами в першій частині таблиці. Слово «Таблиця » подають лише один раз над першою частиною таблиці. Над іншими частинами таблиці з абзацного відступу друкують «Продовження таблиці » або «Кінець таблиці ____ » без повторення її назви.	28
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
7.10.6	Пояснення позначок, які входять до формули чи рівняння, треба подавати безпосередньо під формулою або рівнянням у тій послідовності, у якій їх наведено у формулі або рівнянні. Пояснення позначок треба подавати без абзацного відступу з нового рядка, починаючи зі слова «де» без двокрапки. Позначки, яким встановлюють визначення чи пояснення, рекомендовано ви-рівнювати у вертикальному напрямку.	41
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні позначки	
	7.15 Додатки	

Експерт _____ (підпис) Вадим НЕЧВОЛОД (прізвище, ініціали)

17.06.2025

ДОДАТОК Д

Код для виконання оціночних операцій (на прикладі Apache Spark)

```
1. @Slf4j
2. public class SparkDataOperations {
3.
4.     public static Dataset<ReviewDTO>
   loadReviewsFromJson(SparkSession spark, String filePath) {
5.         try {
6.             log.info("Loading reviews from: {}", filePath);
7.
8.             List<String> jsonLines =
   Files.readAllLines(Paths.get(filePath));
9.
10.                Dataset<String> jsonDataset =
   spark.createDataset(jsonLines, Encoders.STRING());
11.                Dataset<Row> rawData = spark.read()
12.                        .option("multiline", "false")
13.                        .json(jsonDataset);
14.
15.                rawData.createOrReplaceTempView("raw_reviews");
16.
17.                String selectSQL = """
18.                        SELECT
19.                            overall,
20.                            vote,
21.                            verified,
22.                            reviewTime,
23.                            reviewerID,
24.                            asin,
25.                            reviewerName,
26.                            reviewText,
27.                            summary,
28.                            unixReviewTime,
29.                            style.`Format:` as format
30.                        FROM raw_reviews
31.                        WHERE overall IS NOT NULL AND asin IS NOT
   NULL AND asin != ''
32.                        """;
33.
34.                Dataset<Row> cleanData = spark.sql(selectSQL);
35.
36.                Encoder<ReviewDTO> encoder =
   Encoders.bean(ReviewDTO.class);
37.                Dataset<ReviewDTO> reviews = cleanData.as(encoder);
38.
39.                long count = reviews.count();
40.                log.info("Loaded {} valid reviews", count);
41.                return reviews;
42.
43.            } catch (Exception e) {
44.                log.error("Error loading reviews: {}",
   e.getMessage());
45.                throw new RuntimeException("Failed to load reviews
   data", e);
```

```
46.           }
47.       }
48.
49.       public static Dataset<ReviewAggregationDTO>
   performAggregation(SparkSession spark, Dataset<ReviewDTO> reviews)
   {
50.           try {
51.               reviews.createOrReplaceTempView("reviews");
52.
53.               String aggregationSQL = """
54.                       SELECT
55.                           asin,
56.                           COALESCE(format, 'Unknown') as category,
57.                           COUNT(*) as totalReviews,
58.                           CAST(ROUND(AVG(overall), 2) AS DOUBLE)
   as averageRating,
59.                           SUM(CASE WHEN verified = true THEN 1
   ELSE 0 END) as verifiedReviews,
60.                           SUM(CASE WHEN verified = false OR
   verified IS NULL THEN 1 ELSE 0 END) as unverifiedReviews,
61.                           CAST(ROUND(SUM(CASE WHEN verified = true
   THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) AS DOUBLE) as
   verificationRate,
62.                           first(reviewerName) as topReviewer
63.                       FROM reviews
64.                       WHERE overall IS NOT NULL
65.                       GROUP BY asin, COALESCE(format, 'Unknown')
66.                       HAVING COUNT(*) >= 3
67.                       ORDER BY totalReviews DESC
68.                       """;
69.
70.               Dataset<Row> aggregationResult =
   spark.sql(aggregationSQL);
71.
72.               Encoder<ReviewAggregationDTO> encoder =
   Encoders.bean(ReviewAggregationDTO.class);
73.               Dataset<ReviewAggregationDTO> aggregated =
   aggregationResult.as(encoder);
74.
75.               long count = aggregated.count();
76.               log.info("Aggregation completed. Generated {}
   records", count);
77.
78.               aggregated.show(10, false);
79.               return aggregated;
80.
81.           } catch (Exception e) {
82.               log.error("Aggregation operation failed", e);
83.               throw new RuntimeException("Aggregation failed", e);
84.           }
85.       }
86.
87.       public static Dataset<ProductPerformanceDTO>
   performJoinOperation(SparkSession spark,
   Dataset<ReviewAggregationDTO> aggregated) {
88.           try {
89.               log.info("Starting join operation...");
90.
```

```
91.
   aggregated.createOrReplaceTempView("aggregated_reviews");
92.
93.              String joinSQL = """
94.                      SELECT
95.                          ar.asin,
96.                          ar.category as productCategory,
97.                          CAST(ar.averageRating *
   (ar.verificationRate / 100.0) AS DOUBLE) as overallScore,
98.                          ar.totalReviews as reviewCount,
99.                          ar.averageRating,
100.                         CASE
101.                             WHEN ar.averageRating >= 4.5 AND
   ar.verificationRate >= 60 THEN 'EXCELLENT'
102.                             WHEN ar.averageRating >= 3.8 AND
   ar.verificationRate >= 40 THEN 'GOOD'
103.                             WHEN ar.averageRating >= 3.0 THEN
   'AVERAGE'
104.                             ELSE 'POOR'
105.                         END as performanceCategory
106.                     FROM aggregated_reviews ar
107.                     WHERE ar.totalReviews >= 5
108.                     ORDER BY overallScore DESC
109.                     """;
110.
111.             Dataset<Row> joinResult = spark.sql(joinSQL);
112.
113.             Encoder<ProductPerformanceDTO> encoder =
   Encoders.bean(ProductPerformanceDTO.class);
114.             Dataset<ProductPerformanceDTO> performance =
   joinResult.as(encoder);
115.
116.             long count = performance.count();
117.             log.info("Join operation completed. Generated {}
   records", count);
118.
119.             performance.show(10, false);
120.             return performance;
121.
122.         } catch (Exception e) {
123.             log.error("Join operation failed", e);
124.             throw new RuntimeException("Join operation failed",
   e);
125.         }
126.     }
127.
128.     public static void performStreamProcessing(SparkSession
   spark, Dataset<ReviewDTO> reviews) {
129.         try {
130.             log.info("Starting stream processing
   simulation...");
131.
132.
   reviews.createOrReplaceTempView("streaming_reviews");
133.
134.             String streamSQL = """
135.                     SELECT
136.                         asin,
```

```
                            DATE(FROM_UNIXTIME(unixReviewTime)) as
  review_date,
                            COUNT(*) as daily_reviews,
                            ROUND(AVG(overall), 2) as
  daily_avg_rating,
                            SUM(CASE WHEN verified = true THEN 1
  ELSE 0 END) as daily_verified
                        FROM streaming_reviews
                        WHERE unixReviewTime IS NOT NULL
                        GROUP BY asin,
  DATE(FROM_UNIXTIME(unixReviewTime))
                        ORDER BY review_date DESC, daily_reviews
  DESC
                        """;

                Dataset<Row> streamResult = spark.sql(streamSQL);
                long count = streamResult.count();

                log.info("Stream processing completed. Processed {}
  daily aggregations", count);
                streamResult.show(20, false);

            } catch (Exception e) {
                log.error("Stream processing failed", e);
                throw new RuntimeException("Stream processing
  failed", e);
            }
        }

        public static void performBatchTransformation(SparkSession
  spark, String inputPath, String outputPath) {
            try {
                log.info("⚙Starting batch transformation...");

                Dataset<ReviewDTO> rawReviews =
  loadReviewsFromJson(spark, inputPath);

                rawReviews.createOrReplaceTempView("raw_data");
                Dataset<Row> cleanedData = spark.sql("""
                        SELECT * FROM raw_data
                        WHERE overall IS NOT NULL
                        AND asin IS NOT NULL
                        AND reviewText IS NOT NULL
                        AND length(trim(reviewText)) > 0
                        """);

                Encoder<ReviewDTO> reviewEncoder =
  Encoders.bean(ReviewDTO.class);
                Dataset<ReviewDTO> cleanedReviews =
  cleanedData.as(reviewEncoder);

                log.info("Cleaned data: {} -> {} records",
  rawReviews.count(), cleanedReviews.count());

                Dataset<ReviewAggregationDTO> aggregated =
  performAggregation(spark, cleanedReviews);

```

```
            Dataset<ProductPerformanceDTO> performance =
  performJoinOperation(spark, aggregated);

            try {
                aggregated.coalesce(1)
                        .write()
                        .mode(SaveMode.Overwrite)
                        .json(outputPath + "/aggregated");

                performance.coalesce(1)
                        .write()
                        .mode(SaveMode.Overwrite)
                        .parquet(outputPath + "/performance");

                log.info("Results saved to: {}", outputPath);
            } catch (Exception e) {
                log.warn("⚠Could not save to file system: {}",
  e.getMessage());
            }

            log.info("Batch transformation completed
  successfully");

        } catch (Exception e) {
            log.error("Batch transformation failed", e);
            throw new RuntimeException("Batch transformation
  failed", e);
        }
    }
}
```