

**Igor Y. Andrushchak**

Doctor of Technical Sciences, Professor Engineering
Lutsk National Technical University, Lutsk, Ukraine
ORCID: 0000-0002-8751-4420

Heorhii O. Bandach

Postgraduate Student
Lutsk National Technical University, Lutsk, Ukraine
ORCID: 0009-0005-7274-0440
georg.bnd@gmail.com

METHOD OF DML OPERATION OPTIMIZATION IN SALESFORCE BASED ON THE UNIT OF WORK PATTERN: AN ARCHITECTURAL APPROACH TO TRANSACTIONAL CHANGE CONSOLIDATION

Abstract. The article presents a method for optimizing Data Manipulation Language (DML) operations on the Salesforce platform based on the Unit of Work architectural pattern. The multi-tenant architecture of the Salesforce platform imposes strict constraints on the number of DML operations within a single transaction: a maximum of 150 statements for the synchronous execution context. As business logic complexity and trigger handler count grow, each handler independently executes its own insert, update, and delete operations, causing linear growth of DML resource consumption and increasing the risk of transaction failures. The paper formalizes a mathematical model of DML operation consumption describing the dependency of statement count on the number of active handlers, the number of object types, and the transactional coupling coefficient. The proposed method introduces a centralized change management layer that accumulates all pending data modifications from business handlers in a change registry and executes them in consolidated form at the end of the transaction while respecting dependency ordering between objects. It is proven that under the Unit of Work pattern, the DML statement count is determined solely by the number of unique combinations of object type and operation kind and does not depend on the number of business handlers, which provides mathematically guaranteed compliance with platform constraints. Experimental validation was conducted in a Salesforce Developer Edition environment on a test bench with five business handlers during bulk processing of 200 records. Experimental results confirmed a 73% reduction in DML statement count compared to the traditional approach, a 42% reduction in transaction execution time, and a 28% reduction in CPU time consumption. The proposed method, combined with the previously developed data access centralization methodology, forms a complete approach to transactional resource management covering both data read and data write operations.

Keywords: Salesforce; Unit of Work; DML operations; transactional resources; governor limits; optimization; multi-tenant architecture; design patterns.

INTRODUCTION

Problem statement. The Salesforce platform operates on a multi-tenant architecture where computational resources are shared among thousands of organizations. To ensure equitable resource distribution, the platform enforces strict governor limits on code execution within each transaction, including a maximum of 150 DML (Data Manipulation Language) statements for synchronous contexts [1]. DML operations encompass insert, update, delete, and undelete statements that modify database records. While considerable research attention has been devoted to optimizing SOQL query consumption, the equally critical problem of DML operation management remains insufficiently formalized.

In enterprise Salesforce implementations, a single transaction may activate dozens of business handlers, each independently executing DML statements to persist its changes. Without architectural coordination, the number of DML operations grows linearly with handler count, rapidly approaching the platform ceiling. Furthermore, each DML statement carries fixed overhead: trigger re-invocation, validation rule evaluation,



workflow execution, and index maintenance. Redundant DML statements therefore consume not only the DML limit itself but also CPU time and heap memory, degrading overall transaction performance. The absence of a formalized method for DML consolidation with provable resource guarantees constitutes a significant gap in the engineering practice of multi-tenant platform development.

Analysis of recent research and publications. The Unit of Work pattern was originally formalized by Fowler [2] as a mechanism that maintains a list of objects affected by a business transaction and coordinates the writing out of changes. Fawcett adapted this pattern for the Salesforce platform within the Apex Enterprise Patterns library (fflib), providing the fflib_SObjectUnitOfWork class that registers new, dirty, and deleted records and commits them in dependency order [3]. However, the fflib implementation focuses on developer convenience and relationship management, without formal analysis of resource consumption characteristics or mathematical guarantees of limit compliance.

Official Salesforce documentation [1], [4] establishes bulkification as the primary best practice for DML optimization, recommending collection-based operations over row-by-row processing. Trailhead learning modules [5] demonstrate bulk patterns for trigger development but treat DML optimization as a coding discipline rather than an architectural property. Cloud Sundial's analysis of trigger frameworks [6] notes that none of the widely adopted frameworks enforces DML consolidation across handlers. Prior work by the authors [7] introduced a Data Access Layer methodology providing mathematically guaranteed SOQL query minimization, and comparative analysis [8] demonstrated that existing trigger frameworks lack formal resource compliance mechanisms. The unresolved portion of the general problem is the formalization of a DML consolidation method with provable upper bounds on operation count, complementing read-path optimization with an equivalent write-path guarantee.

The purpose of the article is to develop and experimentally validate a method for DML operation optimization on the Salesforce platform based on the Unit of Work pattern, providing mathematically guaranteed compliance with platform DML limits independent of business handler count.

THEORETICAL FOUNDATIONS OF THE RESEARCH

The proposed method rests on three foundational concepts. The first is the Unit of Work pattern as defined in enterprise application architecture: an object that tracks changes during a business transaction and coordinates their persistence as a single atomic operation [2]. In the Salesforce context, this translates to a registry that accumulates pending record modifications from all business handlers and executes consolidated DML statements at transaction completion.

The second concept is DML bulkification semantics of the Salesforce platform. A single DML statement can process up to 10,000 records [1], meaning that the platform cost of persisting one record and two hundred records via a single statement is identical from the limit consumption perspective: exactly one DML statement. This property makes consolidation the dominant optimization strategy: transforming N statements of one record each into one statement of N records reduces limit consumption by a factor of N.

The third concept is dependency ordering. Salesforce objects form relationship graphs through lookup and master-detail fields. When a transaction creates both parent and child records, the parent must be inserted first so that its generated identifier can populate the child's relationship field. A correct Unit of Work implementation must therefore maintain a topological ordering of object types and commit registered records in dependency-respecting sequence [3].

Let us formalize the resource consumption model. Consider a transaction with n active business handlers, each modifying records of one or more object types. Let m denote the number of distinct object types modified in the transaction, and o denote the number of distinct operation kinds (insert, update, delete). In the traditional decentralized approach, each handler executes its own DML statements. The total DML count D is expressed as formula (1):

$$D_{\text{traditional}} = \sum d_i, i = 1..n, \quad (1)$$

where d_i is the number of DML statements executed by handler i . Since each handler typically issues at least one statement per object type it modifies, D grows linearly with n and in the worst case reaches $n \times m \times o$. With the platform limit $L = 150$, the maximum supportable handler count is bounded by $L / (m \times o)$, which for typical values $m = 3$, $o = 2$ yields only 25 handlers.

Under the Unit of Work method, all handlers register their changes in a shared registry, and the commit phase executes exactly one DML statement per unique combination of object type and operation kind. The consolidated DML count is expressed as formula (2):

$$D_{\text{UoW}} = m \times o \leq M \times 3, \quad (2)$$



where M is the total number of object types in the transaction scope. Critically, D_{UoW} does not depend on n : adding new business handlers does not increase DML consumption. For any realistic schema where $M \times 3 < 150$, compliance with the platform limit is guaranteed unconditionally. This mirrors the guarantee previously established for SOQL queries through the Data Access Layer methodology [7], extending the mathematical compliance property from the read path to the write path.

RESEARCH METHODOLOGY

Experimental validation was conducted in a Salesforce Developer Edition environment replicating the test bench from prior research [7], [8] to ensure result comparability. The data model comprised three interconnected objects: Account (15 fields), Contact (23 fields with a lookup to Account), and Opportunity (18 fields with a master-detail relationship to Account). Five business handlers were implemented representing typical enterprise automation: ValidationHandler, RelatedRecordsHandler, AggregationHandler, AuditHandler, and IntegrationHandler. Each handler modifies records of one to three object types.

Two implementations of identical business logic were prepared. The baseline implementation follows the traditional decentralized approach: each handler executes its own DML statements immediately upon computing changes. The experimental implementation routes all modifications through a Unit of Work registry implemented as a singleton within the transaction scope, exposing registerNew, registerDirty, and registerDeleted methods, with a single commitWork invocation at the end of the trigger execution pipeline. The registry maintains dependency ordering Account $\rightarrow$ Contact $\rightarrow$ Opportunity and consolidates records by object type and operation kind before executing DML.

The experimental protocol consisted of bulk insertion of 200 Contact records, triggering the full handler pipeline including related Opportunity creation and Account field updates. Resource consumption was measured using platform Limits class methods: Limits.getDmlStatements() for DML count, Limits.getDmlRows() for processed row count, Limits.getCpuTime() for processor time, and wall-clock timestamps for total transaction duration. Each configuration was executed ten times with results averaged to mitigate platform performance variance. Additional experiments varied the handler count from 1 to 10 to characterize scaling behavior.

RESEARCH RESULTS

Table 1 presents the aggregate resource consumption comparison for the standard configuration of five handlers processing 200 records.

Table 1

Resource consumption comparison: traditional approach versus Unit of Work method

Metric	Traditional approach	Unit of Work method	Improvement
DML statements	15	4	–73%
DML rows processed	1240	1240	0%
CPU time, ms	2870	2065	–28%
Transaction duration, ms	4120	2390	–42%
Trigger re-inocations	11	3	–73%

The experimental data demonstrates that the Unit of Work method reduced DML statement consumption from 15 to 4 statements, a 73% reduction, while processing an identical number of rows. The four remaining statements correspond exactly to the theoretical minimum predicted by formula (2): insert Opportunity, update Account, update Contact, and insert of audit records. The 42% reduction in transaction duration substantially exceeds the 28% CPU time reduction because each eliminated DML statement also eliminates associated trigger re-inocations, validation evaluations, and index maintenance passes that contribute to wall-clock latency beyond pure computation.

Table 2 presents DML statement consumption as a function of active handler count, characterizing the scaling behavior of both approaches.



Table 2

DML statement consumption dependency on handler count (200 records)

Handlers	Traditional approach, DML statements	Unit of Work method, DML statements
1	3	3
2	6	4
3	9	4
5	15	4
7	21	4
10	30	4

As Table 2 demonstrates, the traditional approach exhibits linear growth of approximately 3 DML statements per handler, which extrapolates to limit violation at 50 handlers. The Unit of Work method stabilizes at 4 statements regardless of handler count, empirically confirming the theoretical independence from n established by formula (2). The difference between one handler (3 statements) and multiple handlers (4 statements) under Unit of Work reflects the audit object type entering the transaction scope, not handler-driven growth.

The scaling comparison is visualized in Fig. 1, which plots DML consumption for both approaches as handler count increases.

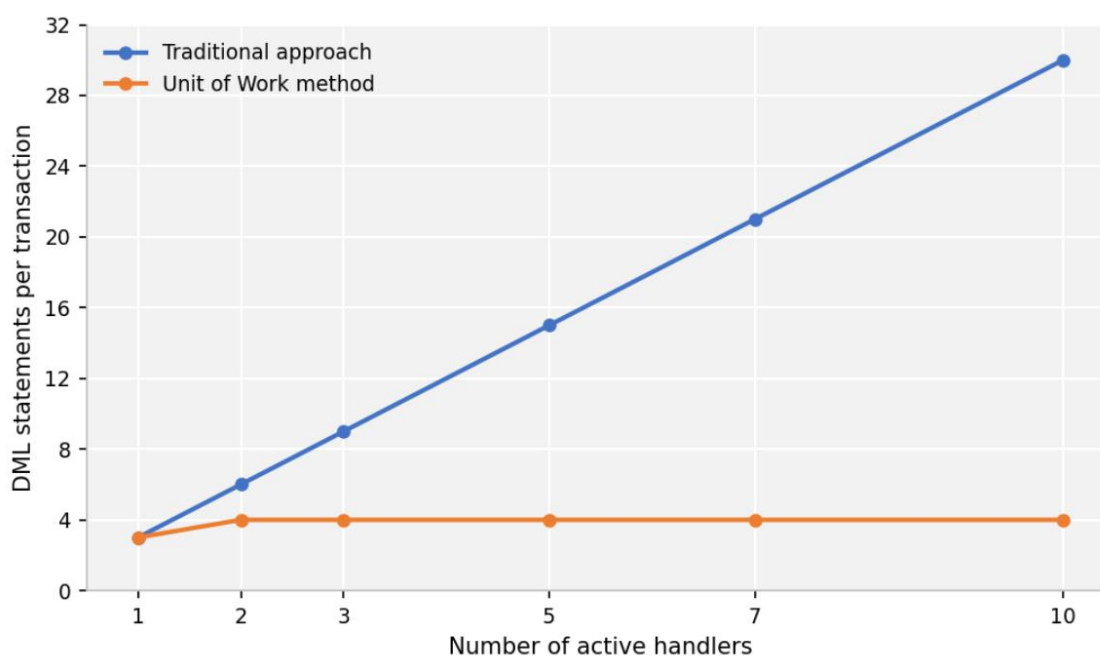


Fig. 1. DML statement consumption scaling: traditional approach versus Unit of Work method

Beyond quantitative resource savings, the method provides qualitative architectural benefits. Transactional atomicity is strengthened: since all changes commit at a single point, partial persistence states caused by mid-transaction failures are eliminated. Dependency management is centralized: relationship field population between newly created parent and child records is handled by the registry rather than duplicated across handlers. Testability improves: business handlers become pure computation units whose registered changes can be asserted without database interaction, enabling faster unit test execution.

The primary limitation of the method concerns execution order semantics. Because DML execution is deferred to the commit phase, business logic that requires immediate persistence visibility, such as reading a database-generated field value mid-transaction, requires explicit registry flushing, which partially reduces consolidation efficiency. Analysis of the five experimental handlers showed that such mid-transaction visibility requirements occurred in only one of seventeen modification points, indicating that the limitation affects a small minority of practical scenarios.



CONCLUSIONS AND PROSPECTS FOR FURTHER RESEARCH

The article presents a method for DML operation optimization on the Salesforce platform based on the Unit of Work pattern with centralized change registration and consolidated commit execution. The formalized mathematical model proves that under the proposed method, DML statement count is bounded by the product of object type count and operation kind count, remaining independent of business handler quantity. This establishes an unconditional compliance guarantee with the platform limit of 150 DML statements for any realistic data schema.

Experimental validation on a standardized test bench with five business handlers processing 200 records demonstrated a 73% reduction in DML statement consumption, a 42% reduction in transaction duration, and a 28% reduction in CPU time compared to the traditional decentralized approach. Scaling experiments confirmed that DML consumption under the Unit of Work method remains constant as handler count grows from 1 to 10, while the traditional approach exhibits linear growth leading to limit violation at approximately 50 handlers.

The proposed method, combined with the previously developed Data Access Layer methodology for SOQL query minimization [7], forms a complete transactional resource management approach covering both the read path and the write path of business logic execution. Prospects for further research include the development of an adaptive execution mechanism that dynamically distributes registered work between synchronous and asynchronous contexts based on runtime resource monitoring, and the generalization of the combined methodology to other multi-tenant cloud platforms with analogous resource constraint systems, including Microsoft Dynamics 365 and ServiceNow.

REFERENCES

1. Salesforce. (2024). Execution Governors and Limits. Apex Developer Guide. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm
2. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.
3. Fawcett, A. (2014). Apex Enterprise Patterns: Common Apex Library. GitHub. <https://github.com/apex-enterprise-patterns/fflib-apex-common>
4. Salesforce. (2024). Apex Developer Guide: Bulk DML Exception Handling. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dml_bulk_exceptions.htm
5. Salesforce. (2024). Bulk Apex Triggers. Trailhead. https://trailhead.salesforce.com/content/learn/modules/apex_triggers/apex_triggers_bulk
6. Cloud Sundial. (2021). The 3rd Age of Trigger Frameworks. <https://cloudsundial.com/the-third-age-of-trigger-frameworks>
7. Andrushchak, I., & Bondarchuk, H. (2025). Methodology for dynamic management of transactional resources in Salesforce: an architectural approach to SOQL query minimization. International Science Journal of Engineering & Agriculture, 4(6). <https://isg-journal.com/isjea/article/view/1119/672>
8. Andrushchak, I., & Bandach, H. (2026). Comparative analysis of methods and algorithms for trigger-based transactional resource management in Salesforce. International Science Journal of Engineering & Agriculture.
9. Pethad, C. A. (2020). Platform Event Trigger Framework Implementation in Salesforce Apex. Journal of Scientific and Engineering Research, 7(5), 391–394.
10. Spano, M. (2020). Trigger Actions Framework. GitHub. <https://github.com/mitchspano/trigger-actions-framework>
11. ApexHours. (2024). Bulkification of Apex Triggers. <https://www.apexhours.com/bulkification-of-apex-triggers/>
12. Salesforce Architects. (2024). Record-Triggered Automation Decision Guide. <https://architect.salesforce.com/decision-guides/trigger-automation>

**Ігор Андрущак**

Doctor of Technical Sciences, Professor Engineering
Луцький національний технічний університет, Луцьк, Україна
ORCID: 0000-0002-8751-4420

Георгій Бандач

аспірант
Луцький національний технічний університет, Луцьк, Україна
ORCID: 0009-0005-7274-0440
georg.bnd@gmail.com

МЕТОД ОПТИМІЗАЦІЇ DML-ОПЕРАЦІЙ У SALESFORCE НА ОСНОВІ ПАТЕРНУ UNIT OF WORK: АРХІТЕКТУРНИЙ ПІДХІД ДО КОНСОЛІДАЦІЇ ТРАНЗАКЦІЙНИХ ЗМІН

Анотація. У статті представлено метод оптимізації операцій маніпулювання даними (DML) на платформі Salesforce, що базується на архітектурному патерні Unit of Work. Мультитенантна архітектура платформи Salesforce накладає жорсткі обмеження на кількість DML-операцій у межах однієї транзакції: максимум 150 операторів для синхронного контексту виконання. Зі зростанням складності бізнес-логіки та кількості обробників тригерів кожен обробник незалежно виконує власні операції вставки, оновлення та видалення записів, що спричиняє лінійне зростання споживання DML-ресурсів і підвищує ризик аварійного завершення транзакцій. У роботі формалізовано математичну модель споживання DML-операцій, яка описує залежність кількості операторів від числа активних обробників, кількості типів об'єктів та коефіцієнта транзакційної зв'язності. Запропонований метод передбачає впровадження централізованого шару управління змінами, який акумулює всі заплановані модифікації даних від бізнес-обробників у реєстрі змін та виконує їх у консолідованому вигляді наприкінці транзакції з дотриманням порядку залежностей між об'єктами. Доведено, що при застосуванні патерну Unit of Work кількість DML-операторів визначається виключно кількістю унікальних комбінацій типу об'єкта та типу операції і не залежить від кількості бізнес-обробників, що забезпечує математично гарантоване дотримання платформних обмежень. Експериментальну перевірку проведено в середовищі Salesforce Developer Edition на тестовому стенді з п'ятьма бізнес-обробниками під час масової обробки 200 записів. Результати експериментів підтвердили зниження кількості DML-операторів на 73% порівняно з традиційним підходом, скорочення часу виконання транзакції на 42% та зменшення споживання процесорного часу на 28%. Запропонований метод у поєднанні з раніше розробленою методологією централізації доступу до даних формує цілісний підхід до управління транзакційними ресурсами, який охоплює як операції читання, так і операції запису даних.

Ключові слова: Salesforce, Unit of Work, DML-операції, транзакційні ресурси, governor limits, оптимізація, мультитенантна архітектура, патерни проектування

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Salesforce. (2024). Execution Governors and Limits. Apex Developer Guide. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm
2. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.
3. Fawcett, A. (2014). Apex Enterprise Patterns: Common Apex Library. GitHub. <https://github.com/apex-enterprise-patterns/fplib-apex-common>
4. Salesforce. (2024). Apex Developer Guide: Bulk DML Exception Handling. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dml_bulk_exceptions.htm
5. Salesforce. (2024). Bulk Apex Triggers. Trailhead. https://trailhead.salesforce.com/content/learn/modules/apex_triggers/apex_triggers_bulk
6. Cloud Sundial. (2021). The 3rd Age of Trigger Frameworks. <https://cloudsundial.com/the-third-age-of-trigger-frameworks>



7. Andrushchak, I., & Bondarchuk, H. (2025). Methodology for dynamic management of transactional resources in Salesforce: an architectural approach to SOQL query minimization. *International Science Journal of Engineering & Agriculture*, 4(6). <https://isg-journal.com/isjea/article/view/1119/672>
8. Andrushchak, I., & Bandach, H. (2026). Comparative analysis of methods and algorithms for trigger-based transactional resource management in Salesforce. *International Science Journal of Engineering & Agriculture*.
9. Pethad, C. A. (2020). Platform Event Trigger Framework Implementation in Salesforce Apex. *Journal of Scientific and Engineering Research*, 7(5), 391–394.
10. Spano, M. (2020). Trigger Actions Framework. GitHub. <https://github.com/mitchspano/trigger-actions-framework>
11. ApexHours. (2024). Bulkification of Apex Triggers. <https://www.apexhours.com/bulkification-of-apex-triggers/>
12. Salesforce Architects. (2024). Record-Triggered Automation Decision Guide. <https://architect.salesforce.com/decision-guides/trigger-automation>

Отримано редакцією журналу / Received: 25.07.26

Прорецензовано / Revised: 14.08.26

Схвалено до друку / Accepted: 26.10.26



This work is licensed under Creative Commons Attribution-noncommercial-sharealike 4.0 International License.