PERFORMANCE OPTIMIZATION OF ORACLE SCM/WMS WORKFLOWS USING SET-ORIENTED DATA ACCESS: AN INDUSTRY CASE STUDY

ABSTRACT

Enterprise Oracle SCM/WMS applications frequently experience performance degradation due to inefficient UI-driven data access patterns. In many production environments, row-wise database queries executed inside iterative UI logic result in excessive database round trips, poor scalability, and prolonged client escalations. This paper presents an industry case study conducted at **Intellinum Private Limited**, focusing on performance optimization of a Pick Confirmation workflow in Oracle WMS. The legacy implementation relied on executing aggregation queries per pick line, leading to severe response delays. The proposed solution replaces this approach with a set-oriented bulk aggregation strategy combined with in-memory key-based lookup. Experimental results demonstrate a significant improvement, reducing execution time from 2.55 minutes to 6 seconds for 185 pick lines, while preserving functional correctness. The study confirms that architectural restructuring of data access patterns is a highly effective approach for resolving performance bottlenecks in enterprise ERP systems.

---

1. INTRODUCTION

Oracle Supply Chain Management (SCM) and Warehouse Management System (WMS) platforms play a critical role in enterprise operations such as inventory control, manufacturing execution, and outbound logistics. These systems handle high transaction volumes and demand responsive user interfaces to ensure operational efficiency.

Despite robust backend capabilities, performance issues frequently arise in custom enterprise interfaces due to inefficient data access

patterns. In particular, UI-driven orchestration of database queries often leads to severe scalability issues when handling large datasets.

This paper addresses a real production performance issue observed in a Pick Confirmation screen within an Oracle WMS environment. The issue resulted in a prolonged client escalation and persisted despite multiple optimization attempts. The objective of this study is to analyze the root cause and present a scalable, production-ready solution that resolves the performance issue without altering core Oracle services.

---

2. LITERATURE REVIEW

Performance optimization in enterprise systems has been widely studied, with several works highlighting the impact of inefficient data access patterns on system scalability. One commonly cited issue is the N+1 query problem, where repeated database queries are executed within iterative application logic.

Enterprise architecture literature emphasizes the importance of separating data access from presentation logic and leveraging set-oriented database operations. Relational databases are optimized for bulk aggregation using grouping operations, whereas UI layers are ill-suited for repetitive database interactions.

While caching and indexing are frequently proposed solutions, these techniques often fail to address fundamental architectural flaws caused by row-wise data processing. This paper builds upon established database optimization principles and applies them to a real Oracle WMS production workflow.

---

3. METHODOLOGY

The optimization methodology adopted in this study focuses on eliminating row-wise database access patterns and introducing a scalable, set-oriented execution model. The approach consists of four structured steps:

1. **Legacy Workflow Analysis**
   The existing Pick Confirmation workflow was examined to identify database access frequency and UI execution flow. It was observed that on-hand quantity calculations were performed inside iterative UI loops.

2. **Root Cause Identification**
   Aggregation queries were executed for each pick line and, in some cases, for each associated lot. This resulted in repeated database round trips and non-linear performance growth.

3. **Set-Oriented Data Retrieval Design**
   All required inventory identifiers—Inventory Item, Subinventory, Locator, and Lot—were extracted in advance to form a unique composite key set.

4. **In-Memory Lookup During UI Rendering**
   A single bulk aggregation query was executed to retrieve on-hand quantities for all required keys. The results were stored in an in-memory map, enabling constant-time lookup during UI table construction.

Algorithm 1: Optimized On-Hand Quantity Retrieval

Input: PickLines

Output: PickLineTableData

Initialize empty set UniqueKeys

Initialize empty map OnHandMap

For each PickLine in PickLines:Extract (Item, Subinventory, Locator, Lot)

Add combination to UniqueKeys

Execute single bulk aggregation query using UniqueKeys

Store aggregated results in OnHandMap For each PickLine in PickLines:

Lookup on-hand quantity from OnHandMap

This algorithm replaces row-wise database queries with a set-oriented aggregation strategy. By executing a single database operation and performing in-memory lookup during UI rendering, the approach minimizes database round trips and ensures predictable performance.

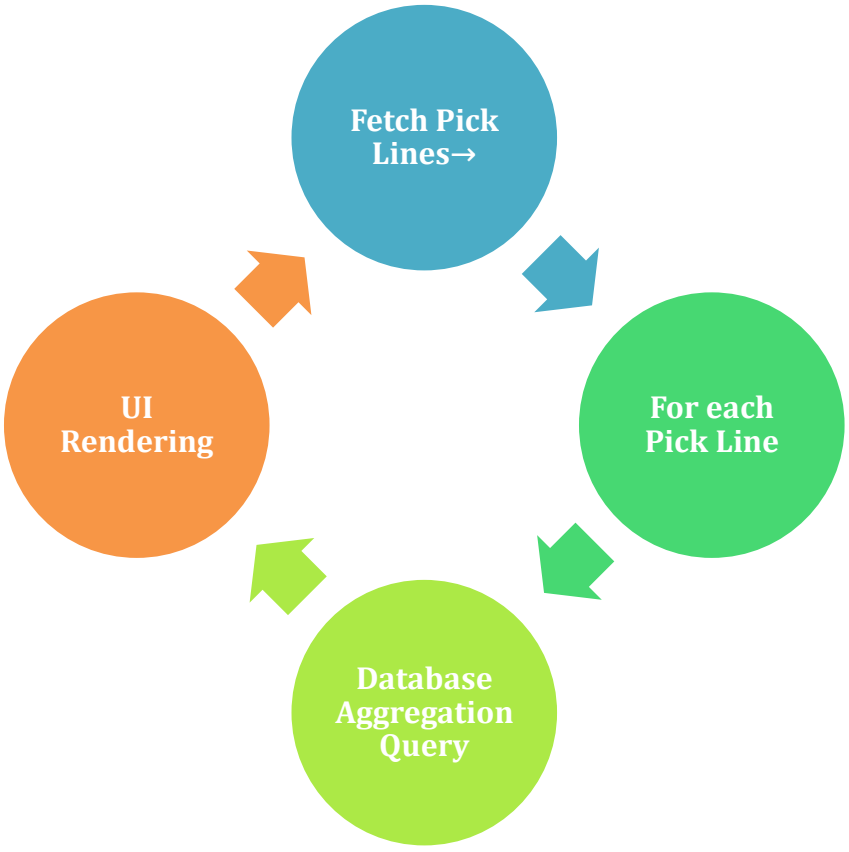
---

4. EXPERIMENTAL RESULTS

4.1 Workflow Comparison

Legacy Workflow:

UI Action



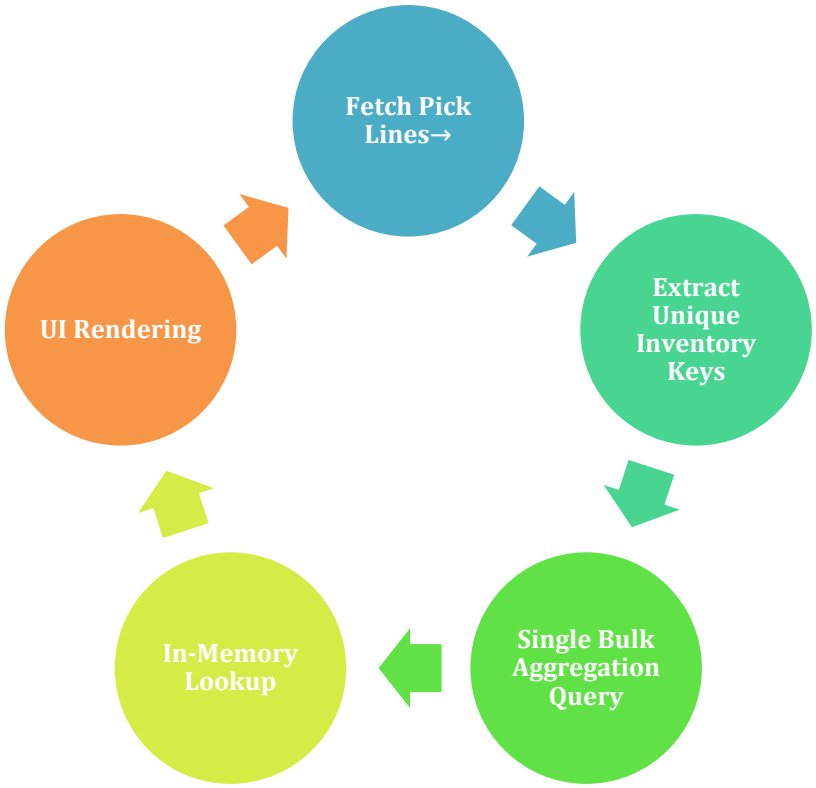
Optimized Workflow:

```
SELECT item_id, subinventory, locator_id, lot_number,
SUM(onhand_quantity)
FROM onhand_inventory_table
GROUP BY item_id, subinventory, locator_id, lot_number;
```

Explanation:

The query illustrates a generic set-oriented aggregation pattern used to compute on-hand quantities for all required inventory combinations in a single operation. Actual table names, schemas, and identifiers are anonymized to preserve confidentiality.

UI Action:



The optimized workflow eliminates repeated database calls and decouples UI rendering from database access.

---

5. TIME COMPLEXITY ANALYSIS

Legacy Implementation:

for each pick_line → execute database query

Let:

- n = number of pick lines
- l = average number of lots per pick line

The legacy implementation executes database aggregation queries inside nested loops. The effective time complexity can be expressed as:

$$\mathbf{O(n \times l \times DB)}$$

Where DB represents the cost of a database round trip and aggregation execution.

As data volume increases, execution time grows rapidly, leading to poor scalability.

Optimized Implementation:

execute database query once → for each pick_line: perform in-memory lookup

In the optimized workflow:

- All inventory keys are collected once
- A single database aggregation query is executed
- UI rendering uses in-memory lookup

The effective complexity becomes:

O(n) + O(DB)

This ensures predictable and stable performance regardless of the number of pick lines.

Complexity Comparison

Aspect	Legacy	Optimized
Database calls	O(n × l)	O(1)
UI processing	High	Linear
Scalability	Poor	Stable

---

6. OBSERVATION

The performance of both implementations was evaluated using identical production data consisting of 185 pick lines.

Metric	Legacy Implementation	Optimized Implementation
Number of Pick Lines	185	185
Database Queries	185+	1
Execution Time	2.55 minutes	6 seconds
Time Complexity	O(n × l)	O(n)
UI Responsiveness	Poor	High

The legacy workflow exhibited non-linear performance degradation as data volume increased. In contrast, the optimized solution maintained stable execution time dominated by a single database operation and efficient in-memory processing.

The optimization successfully resolved a long-standing production escalation that had remained unresolved for several months.

---

7. CONCLUSION

This paper presented an industry case study addressing a real production performance issue in an Oracle WMS Pick Confirmation workflow. The study demonstrated that UI-driven row-wise database aggregation is a major contributor to performance bottlenecks in high-volume enterprise systems.

By redesigning the data access strategy using set-oriented bulk aggregation and in-memory lookup, the execution time was reduced from 2.55 minutes to 6 seconds for the same workload. Time complexity analysis confirmed the transition from a database-bound multiplicative execution model to a linear and scalable workflow.

The findings emphasize that many ERP performance issues originate from architectural design choices rather than platform limitations. The proposed approach is generalizable and can be applied to other Oracle SCM/WMS workflows experiencing similar challenges.