開志専門職大学 情報学部
2025 年度 ICT 活用総合実習 最終報告要旨

Go 言語における ORM と Raw SQL の 定量的パフォーマンス比較研究

中西 悠元

学籍番号： 20122055
指導教員： 中西 悠元

1. はじめに

研究背景

1.1 研究背景

Go 言語による Web アプリケーション開発において、データベース操作は重要な要素である。実装方法として、ORM ツール（GORM）による抽象化されたアプローチと、Raw SQL による直接的なアプローチが存在する。GORM は約 39,300 の GitHub スターを獲得し、型安全性、開発速度向上、移植性などの利点を提供する一方、パフォーマンスのオーバーヘッドが懸念される。しかし、この問題の多くは定性的な経験に基づいており、具体的な数値データに裏付けられた定量的比較研究は不足している。

1.2 研究目的

本研究では以下の 4 点を目的とする：

- 定量的パフォーマンス測定：実行時間、メモリ使用量、スループットの数値化
- 操作パターン別の性能特性：基本的な CRUD 操作と複雑なクエリにおける性能特性の明確化
- スケーラビリティ分析：1,000 件から 10,000 件までのデータ規模での性能変化の分析
- 判断基準の提示：定量的データに基づく技術選択の判断基準の提供

2. 研究対象とシステム構成

2.1 テスト環境

ハードウェア環境

- MacBook Pro (2020, 13-inch) / Apple M1 チップ / 16 GB メモリ

ソフトウェア環境

- Go 1.24.4 / GORM v1.30.2 / MySQL 8.0 (Docker) / gofakeit v6.28.0

2.2 データモデル

一般的なブログシステムを模した User テーブルと Post テーブルを使用。User ： Post = 1 ： N（CASCADE 削除設定）。

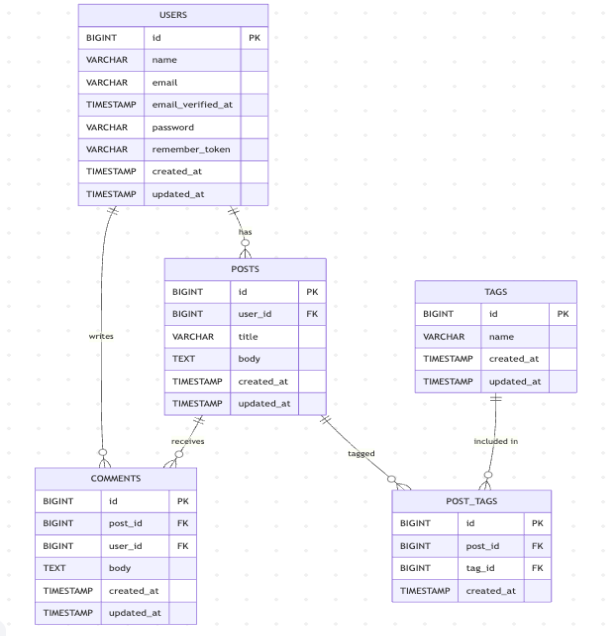

図 1. システム ER 図

2.3 測定項目

各テストケースにおいて、平均実行時間、中央値、標準偏差、P95/P99 パーセンタイル、変動係数（CV）、メモリ使用量、Operations per second（ops/sec）、速度比、メモリ効率比を測定した

3. 実験方法

3.1 テストパターン

実務で頻出する 9 種類のデータベース操作パタ

ーンについてベンチマークを実施：

基本 CRUD 操作（4 種類）

・Create（一括挿入）、Read（ID 取得）、Update（一括更新）、Delete（物理削除）

複雑なクエリパターン（5 種類）

・Pagination（LIMIT/OFFSET）、N+1 問題の最適化（Preload vs JOIN）、複数テーブル JOIN、集計クエリ（GROUP BY）、IN 句 with 大量 ID

3.2 データ規模と測定手順

1,000 件から 10,000 件まで 1,000 件刻みで 10 段階のデータ規模でテストを実施。各テストケースは、環境準備→データベース初期化→データ準備→ウォームアップ（3 回）→本測定（10 回、各回の前に GC 実行）→統計処理の手順で厳密に実施した。

4. 実験結果

4.1 基本クラッドの実行結果

10,000 件のデータにおける各操作の測定結果を以下に示す。

操作	GORM 時間	Raw SQL 時間	速度比	GORM メモリ	Raw SQL メモリ
Create	12,831 ms	8,323 ms	1.54 倍	887.23 MB	69.34 MB
Read	4,030 ms	3,333 ms	1.21 倍	609.13 MB	168.78 MB
Update	13,494 ms	8,027 ms	1.68 倍	802.56 MB	46.49 MB
Delete	7,906 ms	3,416 ms	2.31 倍	636.32 MB	29.70 MB

表 1. 基本クラッド操作まとめ表

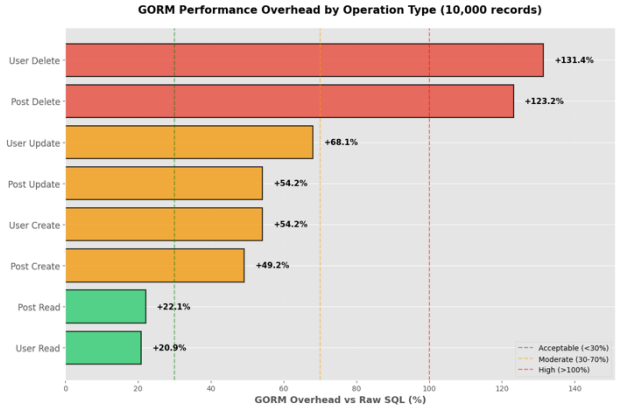

図 2. 基本 CRUD のオーバーヘッド

Raw SQL は全操作で 1.21～2.31 倍高速であった。特に Delete 操作では 2.31 倍という最大の速度差が観測された。メモリ使用量では、Raw SQL が GORM の 3.6～21.4 分の 1 という驚異的な効率を示した。

4.2 複雑なクエリパターンの結果

操作パターン	GORM 時間	Raw SQL 時間	速度比	主な特徴
Pagination	3,085 ms	3,049 ms	1.01 倍	ほぼ同等
N+1 問題最適化	10.66 ms	3.40 ms	**3.14 倍**	最大差
複数 JOIN	30.77 s	22.98 s	1.34 倍	中程度の差
集計クエリ	3,283 ms	3,352 ms	**0.98 倍**	GORM 優位
IN 句 with 大量 ID	32.12 ms	24.22 ms	1.33 倍	中程度の差

表 2. 複雑クエリ操作まとめ表

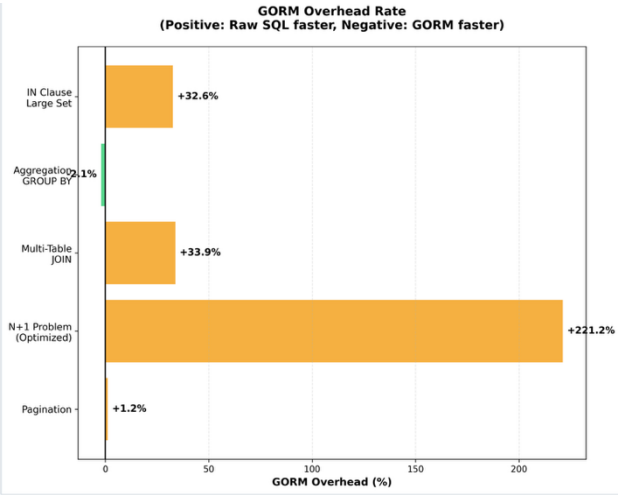

図 3. 複雑クエリのオーバーヘッド

N+1 問題の最適化では 3.14 倍という最大の速度差を示した。GORM の Preload 機能は 2 クエリアプローチを採用する一方、Raw SQL は単一 JOIN で全データを取得するため、この差が生じた。集計クエリでは唯一 GORM が優位を示した。

4.3 スケーラビリティ分析

データ量の増加（1,000 件→10,000 件）に対して、両者ともにほぼ線形にスケールすることが確認された。Create 操作では、理論的な 10 倍に対して約 9 倍（GORM：9.2 倍、Raw SQL：9.0 倍）となり、固定オーバーヘッドの存在が示された。メモリ使用量も線形に増加したが、GORM と Raw SQL の比率は一定（約 12.8 倍）を維持した。

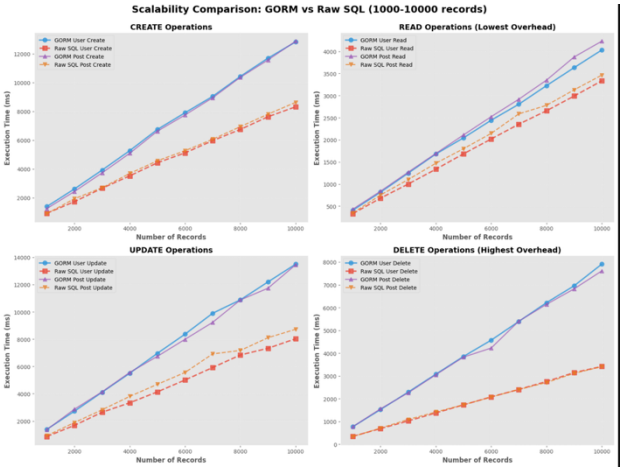

図 4. 基本 CRUD のスケーラビリティ

5. 考察

5.1 パフォーマンス差の要因分析

書き込み操作のオーバーヘッド

GORM はリフレクション（10〜15 倍の時間コスト）、ライフサイクルフック、バリデーション、動的 SQL 生成のオーバーヘッドを伴う。Raw SQL はプリペアドステートメントの繰り返し実行により効率的に動作する。

削除操作の大きな差

GORM のソフトデリート機構（DELETE→UPDATE 変換）、削除前の存在確認（クエリ数 2 倍）、関連レコードチェックが大きなオーバーヘッドとなる。

読み取り操作の小さな差

読み取り操作は重いフック処理が少なく、データベース側の処理が約 80%を占めるため、GORM のオーバーヘッドが目立ちにくい。

N+1 問題での大きな差

GORM の 2 クエリアプローチ（親 3ms+子 7ms＝10ms）に対し、Raw SQL の単一 JOIN クエリ（3ms）という根本的なアプローチの違いが、最大の性能差を生んだ。

5.2 メモリ効率の分析

Raw SQL が GORM に対して 1.0〜21.4 倍のメモリ効率を示した主要因は、構造体のパディングとメタデータ保持、中間データ構造の生成、ガベージコレクション圧力（GORM：平均 47 回、Raw SQL：平均 3 回）である。コンテナ環境では、メモリ制限 1GB での 10,000 件操作時、GORM は約 900MB（限界ぎりぎり）に対し、Raw SQL は約 70MB（十分な余裕）となり、実用上大きな差となる。

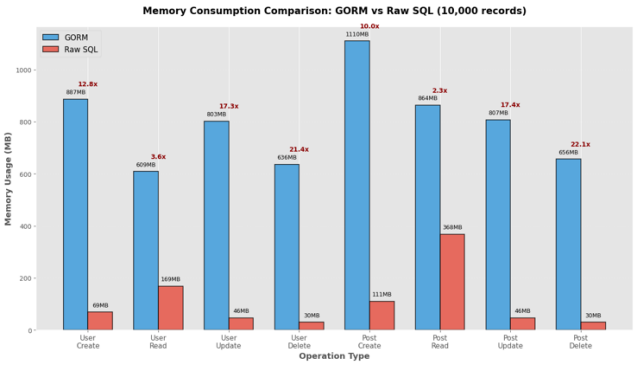

図 5. 基本 CRUD のメモリ使用率

5.3 実務における技術選択の指針

Raw SQL 推奨条件

- 高スループット要件（≥1,000 ops/sec）
- メモリ制約（<500MB）
- 大量データ処理（≥10,000 件/操作）
- 厳格な応答時間 SLA（<100ms）
- バッチ処理、コスト最適化

GORM 許容条件

- 開発速度重視（MVP フェーズ、プロトタイピング）
- 複雑なビジネスロジック（フック、バリデーション、ソフトデリート）
- 小〜中規模データ（<10,000 件/操作）
- 緩いパフォーマンス要件（≥100ms 許容）

・ データベース移植性、単純なクエリ（Pagination、集計）

推奨アプローチ：初期は GORM で開発し、成長フェーズでボトルネックを特定し、高負荷操作を Raw SQL で最適化するハイブリッド戦略が実務では最も効果的である。

5.4 研究の限界

本研究は MySQL 8.0 に限定され、データ規模は 10,000 件まで、単一スレッドでの測定であり、GORM のデフォルト設定を使用した。より大規模なデータ、並行処理、他のデータベース、GORM のチューニングにより結果は変化する可能性がある。

6. 結論

主要な発見

Raw SQL は全操作で 1.0〜3.1 倍のパフォーマンス優位性を示した（N+1 問題：3.14 倍、Delete：2.31 倍が最大）。メモリ効率では 1.0〜21.4 倍の差を示し、コンテナ環境での優位性が明確となった。単純なクエリでは差は最小（1.0〜1.3 倍）、複雑なクエリでは差が拡大（1.3〜3.1 倍）し、データ量増加に対して両者ともに線形にスケールした。

実務への示唆

高スループット、メモリ制約、大量データ処理、厳格な SLA では Raw SQL を推奨する。開発速度重視、複雑なロジック、小〜中規模データ、緩い要件では GORM が許容される。実務では、初期は GORM で開発し、ボトルネックを特定後に Raw SQL で最適化するハイブリッドアプローチが最も効果的である。

学術的貢献

Go 言語 ORM に関する包括的な定量的データ（9 操作パターン、10 段階データ規模）、Apple Silicon 環境での性能評価、メモリ効率の詳細分析、統計的信頼性を確保した実務志向の判断基準を提供した。

今後の展望

PostgreSQL 等の他データベースでの検証、より大規模データ（100,000 件以上）での評価、並行処理の影響分析、他 ORM ツール（ent、SQLBoiler）との比較が今後の課題である。

最終的な推奨

技術選択は、パフォーマンスだけでなく、チームスキル、保守性、拡張性、ビジネス要件など多角的な要因を考慮すべきである。本研究の定量的データは、その判断材料として活用されることを期待する。