УДК 004.652.4:004.75

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ГІБРИДНОГО ПІДХОДУ ДОСТУПУ ДО POSTGRESQL У NODE.JS ЗАСТОСУНКАХ

Бобирь І.О.
e-mail: ivan.bobyr.cpe@nure.ua
Науковий керівник – д.т.н., проф. Смеляков К.С.
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This paper presents an experimental comparison of database access approaches in Node.js applications using PostgreSQL. The study evaluates raw SQL, ORM, and a hybrid approach that combines ORM with optimized SQL queries. Benchmark tests were performed on a dataset of about one million records with multiple iterations and a warm-up phase. The results show that the hybrid approach can achieve performance close to raw SQL in complex queries while preserving the development convenience of ORM. However, for simple operations its advantage may be minimal or absent.

У сучасних серверних застосунках на платформі Node.js доступ до реляційних баз даних зазвичай реалізується за допомогою одного з двох підходів: використання ORM або прямого виконання SQL-запитів. ORM-інструменти значно спрощують розробку, забезпечуючи абстракцію над реляційною моделлю даних і дозволяючи працювати з базою даних через об’єктну модель. Водночас такий рівень абстракції може створювати додаткові накладні витрати під час генерації SQL-запитів і впливати на продуктивність системи, особливо у складних сценаріях доступу до даних [1]. З іншого боку, використання raw SQL дозволяє отримати максимальний контроль над запитами та потенційно досягти кращих показників продуктивності, однак ускладнює підтримку та розвиток програмного коду [2].

У зв’язку з цим у практиці розробки все частіше застосовується гібридний підхід, який поєднує переваги обох стратегій. У межах такої архітектури ORM використовується для стандартних CRUD-операцій, тоді як для складніших або критичних з точки зору продуктивності запитів застосовуються оптимізовані SQL-реалізації. Подібний підхід дозволяє зберегти зручність розробки та водночас уникнути суттєвих втрат продуктивності у найбільш навантажених частинах системи [3].

Для оцінки ефективності різних підходів було проведено експериментальне дослідження у середовищі, що моделює типове серверне навантаження. Тестування виконувалося на базі даних обсягом близько одного мільйона записів із використанням бенчмаркового стенду. Для кожного сценарію виконувалося 200 ітерацій після попереднього прогріву

системи, а навантаження формувалося із використанням середніх значень конкурентності запитів, характерних для типових веб-застосунків.

Для початкового порівняння було обрано прості операції читання даних, що є найбільш поширеними у серверних системах. Зокрема розглядалися два типові сценарії: отримання одного запису за ідентифікатором (findUserById) та отримання сторінки результатів із посторінковою навігацією (listUsers_paged). Аналіз цих операцій дозволяє оцінити вплив архітектурного підходу доступу до бази даних на продуктивність у найпоширеніших CRUD-сценаріях.

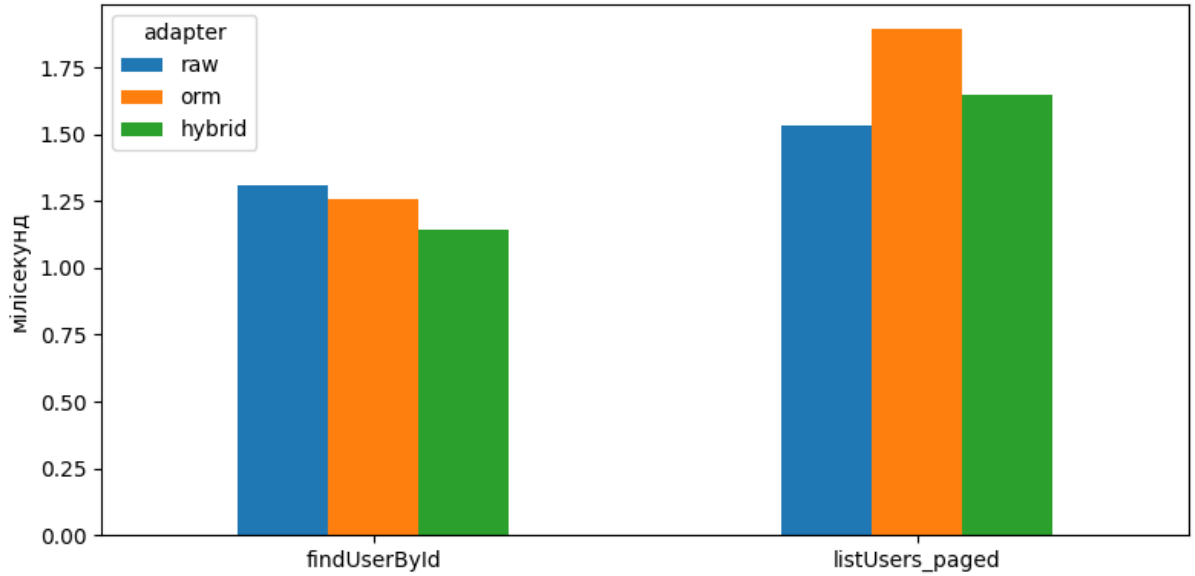

Рисунок 1 – Порівняння латентності (p95) для простої операції читання findUserById

Отримані результати показують, що для простих операцій читання різниця між досліджуваними підходами є незначною. Для сценарію findUserById значення p95 становить приблизно 1.31 мс для raw-підходу, 1.26 мс для ORM та 1.14 мс для гібридного підходу. Для сценарію listUsers_paged значення p95 складають відповідно 1.54 мс, 1.89 мс та 1.65 мс. Таким чином, результати експерименту демонструють, що у типових CRUD-операціях вибір архітектури доступу до даних практично не впливає на латентність системи. Гібридний підхід демонструє продуктивність, співставну з прямим SQL та ORM, що підтверджує можливість його використання без ризику суттєвого погіршення часу відповіді застосунку.

Наступним етапом дослідження було проведено оцінку продуктивності підходів доступу до даних у більш складному сценарії. На відміну від типових CRUD-операцій, складні запити часто включають об’єднання кількох таблиць, додаткову обробку результатів та формування більш складних структур даних на стороні застосунку. У таких випадках накладні витрати рівня доступу до даних можуть впливати на загальну продуктивність системи значно сильніше.

Для перевірки цього припущення було використано сценарій getTopOrdersWithItems, який передбачає виконання запиту з використанням операцій JOIN та обробкою пов’язаних записів. У рамках експерименту порівнювалися три підходи доступу до PostgreSQL: прямий SQL (raw), використання ORM та гібридний підхід. Як і в попередньому експерименті, основною метрикою для оцінки продуктивності було обрано 95-й перцентиль часу відповіді (p95).

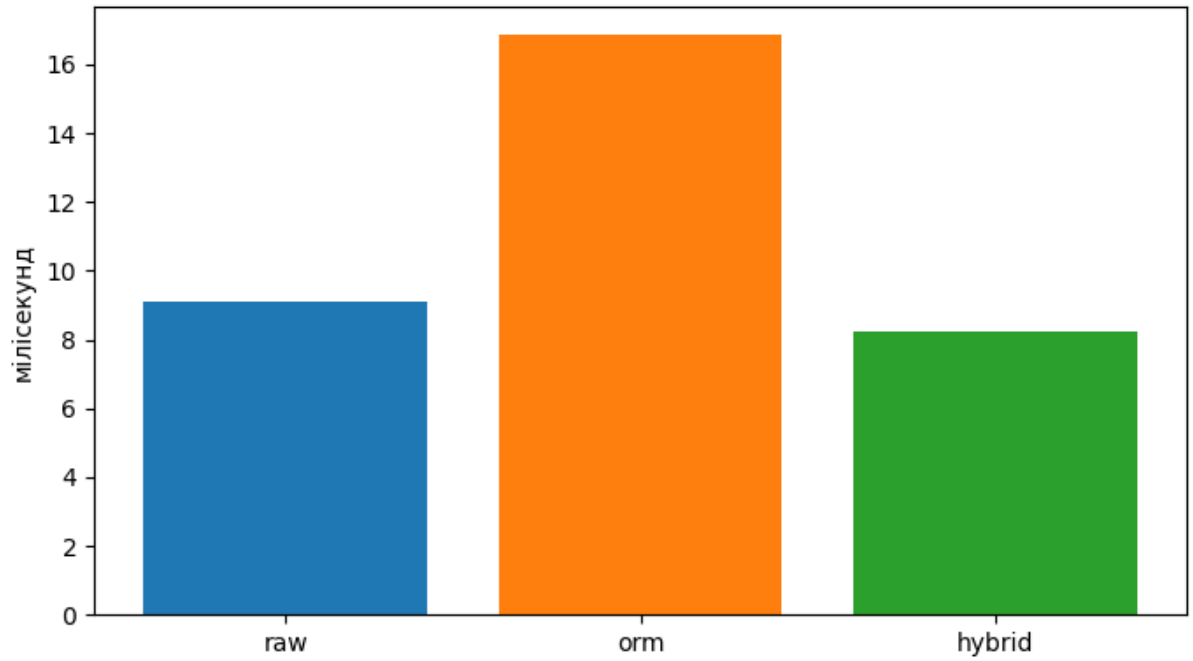

Рисунок 2 – Порівняння латентності (p95) для складнішого запиту з об’єднанням таблиць getTopOrdersWithItems

Отримані результати демонструють суттєву різницю між досліджуваними підходами. Для складного запиту getTopOrdersWithItems значення p95 для ORM становить приблизно 16.8 мс, тоді як для прямого SQL цей показник дорівнює близько 9.1 мс. Гібридний підхід демонструє ще кращий результат – близько 8.2 мс.

Таким чином, при виконанні складніших запитів накладні витрати ORM стають значно помітнішими та призводять до зростання латентності. Гібридний підхід у цьому випадку дозволяє використовувати оптимізований SQL для критичних операцій, що забезпечує продуктивність, близьку до прямого доступу до бази даних, при збереженні переваг використання ORM у решті частини системи.

Наступний експериментальний приклад демонструє сценарій посторінкового отримання списку користувачів (listUsers_paged). Подібні запити є типовими для прикладних інформаційних систем і використовуються для формування сторінок зі списками об’єктів у веб-інтерфейсах або API. На відміну від більш складних аналітичних чи

багатотабличних запитів, така операція має відносно просту структуру та зазвичай добре оптимізується механізмами ORM.

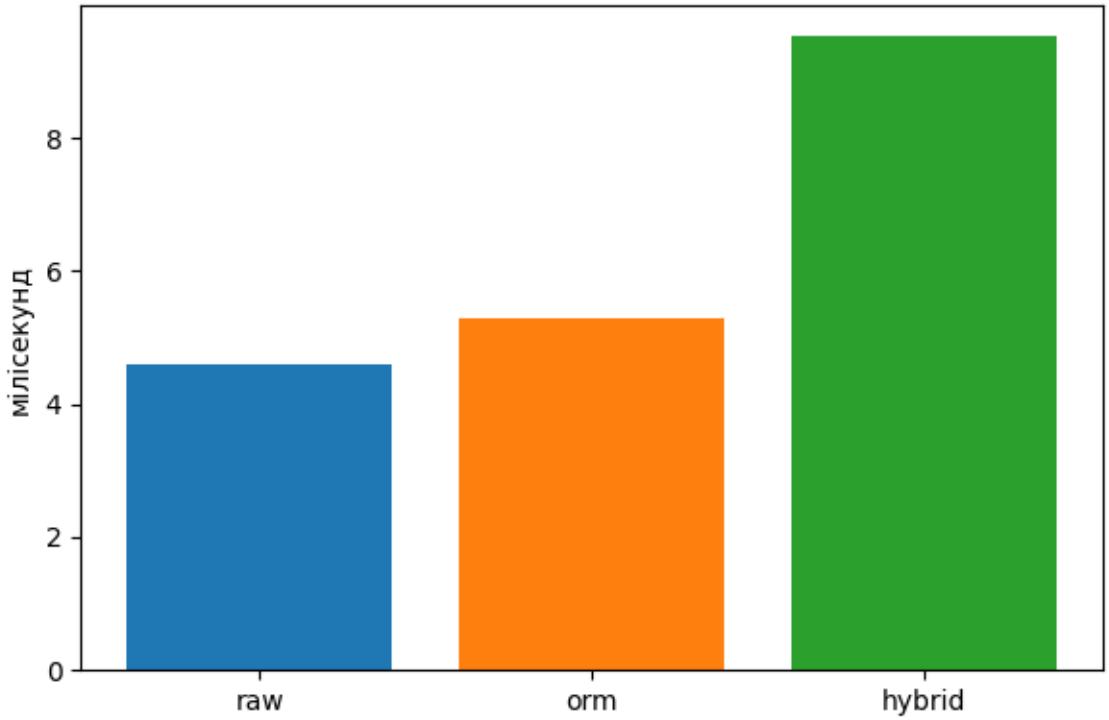

Рисунок 3 – Порівняння латентності (p95) для посторінкової вибірки даних listUsers_paged

Результати показують, що у цьому сценарії гібридний підхід не демонструє переваги над іншими варіантами доступу до бази даних. Навпаки, значення p95 для нього є помітно вищим, ніж для raw-SQL та ORM. Якщо для прямого SQL-запиту p95 становить близько 4.59 мс, а для ORM – приблизно 5.28 мс, то для гібридної реалізації цей показник зростає до 9.52 мс. Таким чином, у даному випадку гібридна реалізація показує майже вдвічі більшу латентність.

Подібний результат свідчить про те, що переваги гібридного підходу проявляються не в усіх типах операцій. Для відносно простих CRUD-запитів ORM часто забезпечує достатньо ефективну генерацію SQL та не створює значних накладних витрат. У таких умовах додаткова логіка, яка використовується у гібридній архітектурі, може навіть дещо погіршувати результати. Це підтверджує, що використання гібридного підходу доцільно насамперед у більш складних сценаріях доступу до даних, тоді як для простих операцій читання виграш може бути відсутнім.

Проведене експериментальне дослідження дозволяє зробити висновок, що ефективність підходів доступу до бази даних значною мірою залежить від характеру виконуваних запитів. Отримані результати підтверджують, що для простих операцій читання різниця між підходами може бути незначною або навіть відсутньою, тоді як у більш складних запитах оптимізовані SQL-реалізації здатні демонструвати помітно кращі

результати. Подібні висновки узгоджуються з результатами інших досліджень у галузі, де зазначається, що використання ORM спрощує розробку програмного забезпечення, але може створювати додаткові накладні витрати при трансляції об’єктної моделі у реляційну структуру бази даних [4].

У цьому контексті гібридний підхід може розглядатися як компромісна архітектурна стратегія. Він дозволяє використовувати ORM для типових CRUD-операцій, зберігаючи переваги швидкої розробки та підтримки коду, і водночас застосовувати оптимізовані SQL-запити у критичних з точки зору продуктивності сценаріях. Така модель поєднання різних механізмів доступу до даних часто рекомендується у практиці розробки серверних застосунків, оскільки дає можливість балансувати між продуктивністю системи та зручністю розробки. Отримані у роботі експериментальні результати підтверджують доцільність такого підходу та демонструють, що вибір конкретної технології доступу до бази даних повинен здійснюватися з урахуванням типу запитів та вимог до продуктивності системи.

Список використаних джерел:

1. Yusmita J. C., Wijaya J. M., Siswanto R. R. Optimizing Database Access Strategy: A Performance Analysis Comparison of Raw SQL and Prisma ORM // *Procedia Computer Science*. 2025. URL: https://doi.org/10.1016/j.procs.2025.09.061 (дата звернення: 03.03.2026).

2. Starić E. Preliminary Study on Effects of Object-Relational Mapping on Database Performance // *Proceedings of the MIPRO Conference*. 2024. URL: https://doi.org/10.1109/MIPRO60963.2024.10569842 (дата звернення: 04.03.2026).

3. Colley D. Identifying New Directions in Database Performance Tuning // *Procedia Computer Science*. 2017. URL: https://doi.org/10.1016/j.procs.2017.11.036 (дата звернення: 03.03.2026).

4. Yusmita J. C. A Performance Analysis Comparison of Raw SQL and Prisma ORM on PostgreSQL Databases // Procedia Computer Science. 2025. URL: https://www.sciencedirect.com/science/article/pii/S187705092502722X (дата звернення: 05.03.2026).