Comparing the Performance of ORMs
การเปรียบเทียบประสิทธิภาพของ ORMs

Jakkapan Attala[1*], and Chaiyaporn Khemapatpapan[2]
จักรพันธ์ อัตลา[1*], และ ชัยพร เขมะภาตะพันธ์[2]

Received: 17 June 2025;
Revised: 15 July 2025;
Accepted: 30 July 2025;
Published: 18 August 2025;

Abstract

Modern application development requires efficient data access, and Object-Relational Mapping (ORM) tools simplify SQL operations by enabling developers to interact with databases using object-oriented paradigms. This study compares the performance of three widely used ORMs Prisma, TypeORM, and Sequelize when used with PostgreSQL in a Docker-based TypeScript environment. The experiments involved four types of table relationships: Single Table, One-to-One, One-to-Many, and Many-to-Many. Performance was evaluated based on response time, memory usage, and CPU utilization. The findings suggest that no single ORM outperforms the others in every aspect, as each tool has distinct advantages and limitations depending on the use case. This study aims to provide developers with insights into selecting the most appropriate ORM for their projects, thereby enhancing the efficiency of database management in modern application development.

Keywords: Object-Relational Mapping, Prisma, TypeORM, Sequelize, Performance Evaluation

บทคัดย่อ

การพัฒนาแอปพลิเคชันยุคใหม่ต้องการการเข้าถึงข้อมูลอย่างมีประสิทธิภาพ โดยเครื่องมือ Object-Relational Mapping (ORM) มีบทบาทสำคัญในการการเขียนคำสั่งในรูปแบบของโครงสร้างข้อมูลแบบเชิงโครงสร้าง ด้วยการจัดการฐานข้อมูลผ่านแนวคิดเชิงวัตถุ งานวิจัยนี้เปรียบเทียบประสิทธิภาพของเครื่องมือ ORM ที่ได้รับความนิยมและใช้งานเครื่องเหล่านี้ ได้แก่ Prisma, TypeORM และ Sequelize ซึ่งใช้งานร่วมกับ PostgreSQL ภายใต้สภาพแวดล้อม Docker และภาษา TypeScript โดยครอบคลุมความสัมพันธ์ของตาราง 4 ประเภท ได้แก่ Single Table, One-to-One, One-to-Many และ Many-to-Many การประเมินประสิทธิภาพพิจารณาจากเวลาในการตอบสนอง การใช้หน่วยความจำ และหน่วยประมวลผลกลาง ผลการทดลองจึงได้พบว่ายังไม่มีเครื่องมือใดที่ดีที่สุดในทุก ๆ ด้าน เนื่องจากแต่ละเครื่องมือมีจุดเด่นและข้อจำกัดแตกต่างกัน งานวิจัยนี้จึงมุ่งหวังที่จะให้ข้อมูลเชิงลึกแก่ผู้พัฒนา

[1,*] Student, Program in Computer Engineering, College of Engineering and Technology, Dhurakij Pundit University, Bangkok 10210, Thailand; นักศึกษา สาขาวิชาคอมพิวเตอร์ศึกษา วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์ มหาวิทยาลัยธุรกิจบัณฑิตย์, กรุงเทพมหานคร 10210 ประเทศไทย; Email: 66130030@dpu.ac.th

[2] Assistant Professor, Dr., Program in Computer Engineering, College of Engineering and Technology, Dhurakij Pundit University, Bangkok 10210, Thailand; ผู้ช่วยศาสตราจารย์ ดร. สาขาวิชาคอมพิวเตอร์ศึกษา วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์ มหาวิทยาลัยธุรกิจบัณฑิตย์, กรุงเทพมหานคร 10210 ประเทศไทย; Email: chaiyaporn@dpu.ac.th

*Corresponding authors: Jakkapan Attala (66130030@dpu.ac.th)

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

ในการเลือกใช้ ORM ให้เหมาะสมกับความต้องการของโครงการ และช่วยเพิ่มประสิทธิภาพในการจัดการฐานข้อมูลในกระบวนการพัฒนาแอปพลิเคชันสมัยใหม่
คำสำคัญ: การแมปเชิงวัตถุกับฐานข้อมูลเชิงสัมพันธ์, พริซม่า, ไทป์ออาร์เอ็ม, ซีควอไลซ์, การวัดประสิทธิภาพ

1. บทนำ (Introduction)

การจัดการฐานข้อมูลเชิงสัมพันธ์ (Relational Database) มีบทบาทสำคัญในการพัฒนาแอปพลิเคชันสมัยใหม่ โดยเฉพาะเมื่อมีการใช้ภาษาโปรแกรมเชิงวัตถุ (Object-Oriented Programming: OOP) ซึ่งแนวทางการเข้าถึงข้อมูลแบบดั้งเดิมผ่าน SQL อาจสร้างความยุ่งยากเมื่อต้องดูแลโค้ดจำนวนมากในระยะยาว ด้วยเหตุนี้ เทคโนโลยีการแมปเชิงวัตถุกับฐานข้อมูลเชิงสัมพันธ์ (Object-Relational Mapping: ORM) จึงถูกพัฒนาขึ้นเพื่อเชื่อมโยงข้อมูลจากฐานข้อมูลกับโครงสร้างเชิงวัตถุของโปรแกรมได้โดยตรง ช่วยลดความซับซ้อนในการเขียนคำสั่ง SQL และเพิ่มความปลอดภัยของข้อมูลผ่านแนวคิด Type Safety (Prisma, n.d.) อย่างไรก็ตาม แม้ว่า ORM จะช่วยให้นักพัฒนาสามารถจัดการฐานข้อมูลได้สะดวกขึ้น แต่งานวิจัยที่ผ่านมา เช่น ของ Chen et al. (2016) พบว่า ORM มีแนวโน้มที่จะเกิดการเข้าถึงข้อมูลซ้ำซ้อน (Redundant Data Access) ซึ่งส่งผลกระทบต่อประสิทธิภาพของระบบโดยรวม ทั้งในด้านเวลาในการประมวลผลและการใช้ทรัพยากร เช่น หน่วยความจำและ CPU นอกจากนี้ ORM แต่ละตัวมีแนวทางการทำงานแตกต่างกัน เช่น การจัดการแบบ Active Record หรือ Data Mapper ซึ่งส่งผลต่อพฤติกรรมของระบบและความยืดหยุ่นในการเขียนโปรแกรม

ดังนั้นจึงมีความจำเป็นที่จะต้องช่วยให้ผู้พัฒนาเข้าใจข้อดีข้อเสียของแต่ละ ORM และสามารถตัดสินใจเลือกใช้เทคโนโลยีที่เหมาะสมที่สุดในการพัฒนาแอปพลิเคชัน ซึ่งจะทำให้ลดต้นทุนและเพิ่มประสิทธิภาพในการดำเนินงานออกแบบระบบในระยะยาว สำหรับเครื่องมือที่เลือกมาศึกษา ได้แก่ Prisma, TypeORM และ Sequelize เนื่องจากเป็นเครื่องมือที่ได้รับความนิยมสูงในปัจจุบัน

ในงานวิจัยนี้จึงมีวัตถุประสงค์เพื่อเปรียบเทียบประสิทธิภาพของเครื่องมือ ORM (Object-Relational Mapping) ที่ได้รับความนิยมในระบบที่ใช้ภาษา TypeScript ได้แก่ Prisma TypeORM และ Sequelize ซึ่งใช้งานร่วมกับฐานข้อมูล PostgreSQL ภายใต้สภาพแวดล้อม Docker โดยครอบคลุมความสัมพันธ์ของตาราง 4 ประเภท ได้แก่ ตารางเดี่ยว (Single Table) หนึ่งต่อหนึ่ง (One-to-One) หนึ่งต่อหลาย (One-to-Many) และหลายต่อหลาย (Many-to-Many) การเปรียบเทียบจะพิจารณาจากตัวชี้วัดหลัก ได้แก่ เวลาในการตอบสนอง (Response Time) การใช้หน่วยความจำ (Memory Usage) และหน่วยประมวลผลกลาง (Central Processing Unit: CPU) เพื่อให้เห็นถึงข้อจำกัดของ ORM แต่ละประเภท และเป็นแนวทางในการเลือกใช้ ORM ที่เหมาะสมกับลักษณะของโครงการ โดยเฉพาะในยุคที่การพัฒนาแอปพลิเคชันต้องให้ความสำคัญกับทั้งประสิทธิภาพและความเร็ว

2. วัตถุประสงค์งานวิจัย (Research Objectives)

เพื่อวัดและเปรียบเทียบประสิทธิภาพการทำงานของเครื่องมือ ORM ทั้งสาม ได้แก่ Prisma, TypeORM และ Sequelize

3. การทบทวนวรรณกรรมและทฤษฎีที่เกี่ยวข้อง (Literature Review)

ในการพัฒนาแอปพลิเคชันสมัยใหม่ที่มีการใช้ฐานข้อมูลขนาดใหญ่และซับซ้อน การเลือกเครื่องมือในการเข้าถึงและจัดการข้อมูลจึงเป็นสิ่งสำคัญ โดยเฉพาะอย่างยิ่งการใช้เทคนิค Object–Relational Mapping (ORM) ซึ่งเป็นแนวทางที่ช่วยให้สามารถเชื่อมโยงข้อมูลจากฐานข้อมูลเชิงสัมพันธ์เข้ากับแนวคิดเชิงวัตถุในภาษาโปรแกรมได้สะดวกมากยิ่งขึ้น บทนี้จะนำเสนอแนวคิดที่เกี่ยวข้องและงานวิจัยที่ผ่านมา ซึ่งมีผลต่อการกำหนดแนวทางของงานวิจัยฉบับนี้

3.1 แนวคิดและทฤษฎีเกี่ยวกับ Object–Relational Mapping (ORM)

Object–Relational Mapping (ORM) คือ เทคนิคที่เชื่อมโยงระหว่างแนวคิดของฐานข้อมูลเชิงสัมพันธ์ Relational Database (Monster Connect, n.d.) และการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming: OOP) โดย ORM ทำหน้าที่แปลงข้อมูลจากรูปแบบตารางให้สามารถใช้งานในรูปแบบของคลาสและอ็อบเจกต์ในภาษาโปรแกรมได้โดยตรง แนวคิดนี้ช่วยลดการเขียน SQL แบบดั้งเดิม และสามารถจัดการข้อมูลได้ผ่านคำสั่งที่คล้ายกับโค้ดเชิงวัตถุ เช่น การ

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

ใช้ method และ property และ ORM มีจุดเด่นในการลดความซับซ้อนของการดึงและบันทึกข้อมูล สนับสนุนความปลอดภัยของข้อมูลผ่าน Type Safety (Prisma, n.d.b) และเพิ่มความเร็วในการพัฒนา อย่างไรก็ตาม ORM ก็มีข้อจำกัด เช่น ปัญหา N+1 Query (Vaja & Rahevar, 2016) การใช้ทรัพยากรสูง และความไม่ยืดหยุ่นในการสร้าง Query ที่ซับซ้อน

3.2 งานวิจัยที่เกี่ยวข้อง

Chen et al. (2016) ได้ศึกษาและประเมินผลกระทบต่อประสิทธิภาพการทำงานของการเข้าถึงข้อมูลซ้ำซ้อน (Redundant Data Access) สำหรับแอปพลิเคชันที่พัฒนาโดยใช้กรอบการทำงานที่มักเกิดในระบบที่ใช้ ORM โดยพบว่าปัญหานี้ส่งผลให้ระบบเกิดความล่าช้าและใช้ทรัพยากรเกินความจำเป็น นอกจากนี้ Vaja & Rahevar (2016) ได้เสนอการใช้ In-Memory Caching ร่วมกับ ORM เพื่อเพิ่มประสิทธิภาพ โดยสามารถลดการดึงข้อมูลซ้ำซ้อนจากฐานข้อมูล และลดเวลาในการเข้าถึงข้อมูล โดยที่ Marchuk et al. (2023) ได้ทำการเปรียบเทียบการเข้าถึงฐานข้อมูลผ่าน ORM, REST API, และ GraphQL โดยพบว่า ORM แม้จะสะดวกในการพัฒนาแต่ก็มีค่าใช้ทรัพยากรสูงในบางสถานการณ์ โดยเฉพาะในระบบที่มีการทำงานแบบ Real-time และในงานวิจัยของ Güvercin & Avenoglu, 2022) ได้วิเคราะห์ประสิทธิภาพของ ORM บน .NET เวอร์ชัน 6 และพบว่า ORM แต่ละตัวมีจุดเด่นที่แตกต่างกัน เช่น Entity Framework Core ทำงานได้ดีใน CRUD ทั่วไป แต่ช้าลงเมื่อ Query มีความซับซ้อน จะเห็นว่าแม้จะมีงานวิจัยที่วิเคราะห์ประสิทธิภาพของ ORM มาบ้างแล้วแต่ส่วนมากเน้นเฉพาะแพลตฟอร์มหรือภาษาเฉพาะ เช่น .NET หรือ Java งานวิจัยนี้จึงมีบทบาทเติมเต็มในบริบทของการเปรียบเทียบ ORM ที่ใช้งานร่วมกับ TypeScript และ PostgreSQL โดยออกแบบการทดลองเชิงปริมาณครอบคลุมทุกประเภทความสัมพันธ์ตาราง และใช้ตัวชี้วัดประสิทธิภาพหลากหลาย เช่น Response Time, Memory Usage และ CPU Utilization

3.3 เทคโนโลยีที่เกี่ยวข้อง

ในปัจจุบัน ORM ที่ได้รับความนิยมในระบบที่ใช้ภาษา TypeScript ได้แก่ Prisma, TypeORM และ Sequelize ซึ่งมีลักษณะเฉพาะดังนี้

1. Prisma (Prisma, n.d.b) เป็น ORM รุ่นใหม่ที่เน้นความปลอดภัยของประเภทข้อมูล (Type Safety) มี Prisma Schema, Prisma Client และ Prisma Migrate เป็นเครื่องมือหลัก และเหมาะกับระบบที่ต้องการควบคุมโครงสร้างข้อมูลอย่างเข้มงวด

2. TypeORM (TypeORM., n.d.) ซึ่งรองรับทั้ง Active Record และ Data Mapper Pattern ใช้แนวคิด Entity-based Modeling และรองรับการใช้ Decorator ร่วมกับ TypeScript ใน TypeORM แต่ละตารางในฐานข้อมูล จะถูกแมปเป็น Entity (Class) แต่ละคอลัมน์ในฐานข้อมูล จะถูกกำหนดเป็น Property ของ Class ใช้ Decorators ในการกำหนดโครงสร้าง เช่น @Entity(), @Column(), @PrimaryGeneratedColumn()

3. Sequelize (Renovate, 2025) เป็น ORM ที่พัฒนาบน Node.js รองรับหลายฐานข้อมูล มีลักษณะใช้งานง่าย และมีระบบ Model Associations ที่รองรับความสัมพันธ์ของข้อมูลง่ายขึ้นโดยใช้ JavaScript หรือ TypeScript Model ใน Sequelize เป็นตัวแทนของ ตาราง (Table) ในฐานข้อมูลซึ่งช่วยให้เราสามารถจัดการข้อมูลได้โดยไม่ต้องเขียน SQL โดยตรง แต่ใช้เมธอดของ Sequelize แทนแนวคิดหลักของ Sequelize Model ซึ่งช่วยให้การจัดการ

4. กรอบแนวคิดงานวิจัย (Conceptual Framework)

ในการวิจัยนี้มุ่งเน้นการเปรียบเทียบประสิทธิภาพของเครื่องมือ Object-Relational Mapping (ORM) ที่ใช้งานร่วมกับภาษา TypeScript และฐานข้อมูล PostgreSQL โดยเลือกศึกษาจากเครื่องมือยอดนิยมสามประเภท ได้แก่ Prisma, TypeORM และ Sequelize (Prisma, n.d.a; Renovate, 2025; TypeORM, n.d.) ซึ่งเป็นเครื่องมือที่ได้รับความนิยมในการพัฒนาแอปพลิเคชันด้วยภาษา TypeScript ร่วมกับฐานข้อมูล PostgreSQL

งานวิจัยนี้มีกรอบแนวคิดในการวิจัยโดยศึกษาออกแบบเพื่อวิเคราะห์ความสัมพันธ์ระหว่างตัวแปร ดัง Figure 1.

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

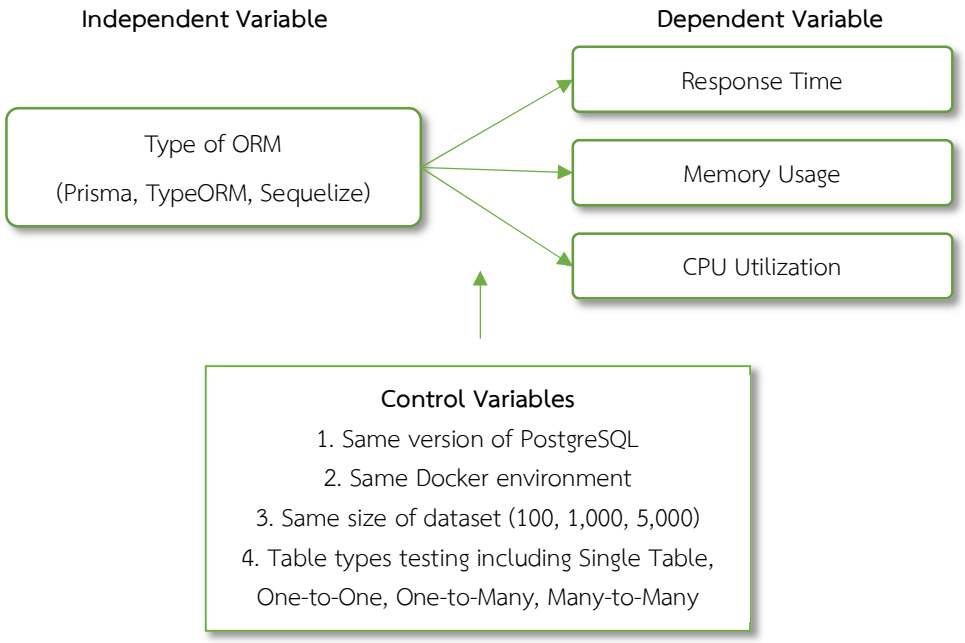

Figure 1. Conceptual Framework.

จาก Figure 1. อธิบายดังนี้

1. ตัวแปรต้น (Independent Variable) ประกอบด้วย ประเภทของ ORM ได้แก่ Prisma, TypeORM และ Sequelize

2. ตัวแปรตาม (Dependent Variables) ประกอบด้วย ประสิทธิภาพของระบบในด้านต่าง ๆ ได้แก่

1) เวลาในการตอบสนอง (Response Time)

2) การใช้หน่วยความจำ (Memory Usage)

3) การใช้หน่วยประมวลผลกลาง (CPU Utilization)

การวิเคราะห์ตัวแปรตาม โดยใช้การวิเคราะห์ความแปรปรวนทางเดียว (One-Way ANOVA) เพื่อวิเคราะห์ความแตกต่างของผลลัพธ์ทางสถิติระหว่างกลุ่ม (ORM แต่ละประเภท)

3. ตัวแปรควบคุม (Control Variables) ได้แก่

1) ใช้ฐานข้อมูล PostgreSQL เวอร์ชั่นเดียวกันในทุกการทดลอง

2) ทดสอบในสภาพแวดล้อม Docker เดียวกัน

3) ใช้ชุดข้อมูลขนาดเดียวกัน (100, 1,000, 5,000)

4) ดำเนินการทดสอบกับรูปแบบความสัมพันธ์ของตารางทั้ง 4 ประเภท ได้แก่ Single Table, One-to-One, One-to-Many และ Many-to-Many

จากองค์ประกอบทั้งหมดดังกล่าวนี้ได้กำหนด สมมติฐานของการศึกษา ซึ่งประกอบด้วยประเด็นสำคัญ 3 ด้าน ดังนี้

1. เครื่องมือ ORM แต่ละประเภทส่งผลต่อประสิทธิภาพของระบบที่แตกต่างกันอย่างมีนัยสำคัญ

2. Prisma ซึ่งรองรับ Schema แบบ Declarative และมี Type Safety สูงจะช่วยลดข้อผิดพลาดในระหว่างการพัฒนา

3. ORM ที่ใช้กับ TypeScript มีแนวโน้มที่จะช่วยให้การพัฒนาแอปพลิเคชันมีประสิทธิภาพมากกว่าการใช้ SQL ตรง

5. วิธีดำเนินงานวิจัย (Research Methodology)

ในการศึกษานี้เราใช้ Docker เป็นหลักในการจัดเตรียมสภาพแวดล้อมทดสอบ เพื่อให้สามารถควบคุม Version ของซอฟต์แวร์และการตั้งค่าได้อย่างแม่นยำ รวมถึงลดความซับซ้อนในการติดตั้งและกำหนดค่า PostgreSQL และ ORM แต่ละตัว

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

ได้แก่ Prisma, TypeORM และ Sequelize การตั้งค่าสภาพแวดล้อมจะดำเนินการผ่าน Docker Compose ซึ่งช่วยให้สามารถจัดการคอนเทนเนอร์หลายตัวได้สะดวก โดยสภาพแวดล้อมจะประกอบไปด้วย

1. PostgreSQL 17 รันบน Docker
2. Node.js + TypeScript สำหรับรันโค้ด
3. Prisma, TypeORM, Sequelize ทดสอบ CRUD Operations และเปรียบเทียบการทำงานสำหรับการแปลงข้อมูล

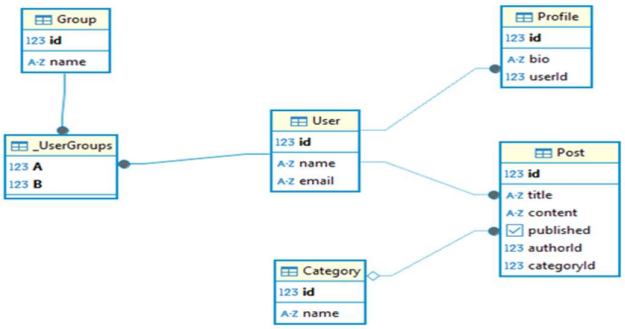

Figure 2. ER table relationships used in testing.

Figure 3. Flow chart of the overall testing process.

จาก Figure 3. วิธีการจะแบ่ง ออกเป็น 3 ส่วนหลัก ๆ คือ

1. การตั้งค่าสภาพแวดล้อม และการกำหนดตั้งค่า ORM แต่ละส่วนโดยมีรูปแบบความสัมพันธ์ของตารางเดียวกันตามรูป Figure 2. และการตั้งค่าสภาพแวดล้อมในรูปแบบเดียวเพื่อให้ได้ข้อมูลที่ตรงมากที่สุด และการเชื่อมต่อ Database ให้เรียบร้อย

2. กระบวนการทดสอบ: การดำเนินการพื้นฐานในฐานข้อมูล (Create, Read, Update, Delete: CRUD) โดยจะมีการเขียน Script test ด้วย Jest (Zaytsev, 2025) ซึ่งคือ framework สำหรับการทดสอบของ JavaScript/TypeScript ซึ่งจะอยู่ที่โฟลเดอร์ tests/* อิงจาก Figure 4. แบ่งตามประเภทของ ORM จะเขียนทดสอบข้อมูลจำนวน 5,000 ข้อมูลและการเก็บข้อมูลในกระบวนการทดสอบโดยจะใช้ API ของตัว NodeJS (Node.js, n.d.) เพื่อเข้าถึงการใช้งานหน่วยความจำ Memory Usage และ CPU Usage โดยผ่านตัวแปลที่ชื่อว่า Process เป็น Global Object ที่ให้เข้าถึงข้อมูลและควบคุมกระบวนการปัจจุบัน (Current process) ภายในสภาพแวดล้อม NodeJS

3. การวิเคราะห์และรวบรวมข้อมูลโดยใช้สคริปต์ ‘runWithMetrics.ts’ จากนั้นจะได้ไฟล์ผลลัพธ์ออกมาเป็นรูปแบบ Json ซึ่งจะบันทึกเวลา ‘queryTimeMs’, หน่วยความจำก่อนและหลัง (‘memoryUsed’), และ CPU usage ของแต่ละไฟล์ทดสอบต่อ 30 ครั้งต่อ 1 กรณีทดสอบจากนั้นจะรวบรวมข้อมูลทั้งหมดและคำนวณสถิติค่าเฉลี่ยพื้นฐาน ตัวอย่างผลลัพธ์ ดังนี้

```
{
    “testFile”: “tests/typeorm/singleTb.test.ts”,
    “queryTimeMs”: 106727,
    “memoryUsed”: “159.78 -> 122.92 MB”,
   “cpuUsed”: “282.00 ms user / 46.00 ms system”,
}
```


Citation: Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

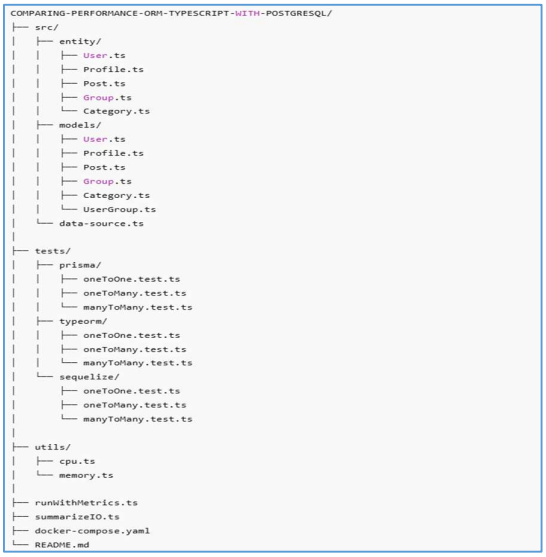

Figure 4. Code and folders used for testing.

จาก Figure 4. สามารถอธิบายแต่ละโฟลเดอร์ ดังนี้

1. src/entity/
   1.1 โฟลเดอร์นี้ใช้เก็บ Entity ของ TypeORM ซึ่งเป็นคลาสที่แมปกับตารางใน PostgreSQL
   1.2 แต่ละไฟล์ เช่น User.ts, Post.ts กำหนดความสัมพันธ์ เช่น One-to-One หรือ One-to-Many
2. src/models/
   2.1 ใช้สำหรับเก็บ Model ของ Sequelize
   2.2 โครงสร้างคล้ายกับ entity แต่ใช้แนวทาง OOP ของ Sequelize
   2.3 UserGroup.ts ใช้สำหรับความสัมพันธ์แบบ Many-to-Many (ผ่าน Table)
3. src/data-source.ts
   3.1 ใช้กำหนดการเชื่อมต่อฐานข้อมูลของ TypeORM
4. tests/
   4.1 โฟลเดอร์นี้รวมไฟล์ทดสอบที่ใช้เปรียบเทียบ ORM ทั้ง 3 ตัวแต่ละโฟลเดอร์ย่อย (prisma, typeorm, sequelize) มีชุดทดสอบแยกตามความสัมพันธ์ เช่น oneToMany.test.ts, manyToMany.test.ts สำหรับวัด Performance ของแต่ละ ORM
5. utils/
   5.1 รวมฟังก์ชันวัด การใช้ CPU (cpu.ts) และหน่วยความจำ (memory.ts) ขณะรันทดสอบ
6. runWithMetrics.ts
   6.1 สคริปต์หลักที่ใช้รันการทดสอบแต่ละ ORM พร้อมจับเวลา/หน่วยความจำ/CPU
7. docker-compose.yaml
   7.1 ใช้สำหรับรัน PostgreSQL ภายใน Docker เพื่อให้ทุก ORM เชื่อมต่อกับฐานข้อมูลเดียวกัน

README.md

ทั้งนี้สามารถศึกษาเพิ่มเติมจากคู่มือการใช้งานโปรเจกต์พร้อมอธิบายการรัน การติดตั้ง และวัตถุประสงค์ และสามารถเข้าถึงโค้ดเพิ่มเติมได้ ที่ GitHub Repository (Attala, 2025)

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

6. ผลการวิจัย (Results)

6.1 ผลการวัดประสิทธิภาพและเปรียบเทียบการทำงานของ ORMs แต่ละตัว

การทดสอบประสิทธิภาพ ORM ทั้ง 3 ตัวและเปรียบเทียบข้อมูลโดยจะรวบรวมข้อมูลจากการทดสอบซึ่งจะนำเสนอโดยรูปแบบ CURD และความสัมพันธ์ของตาราง โดยจะมีการสรุปข้อมูลในการทดสอบและเปรียบเทียบที่ทดสอบขนาด 5,000 ข้อมูล ดัง Table 1-3. และ Figure 5-7.

Table 1. Response Time Summary Table

Relationship	Operation	Prisma (ms)	TypeORM (ms)	Sequelize (ms)
Single Table	Create	28247	46817	29464
	Read	78	126	89
	Update	29266	28825	29422
	Delete	76	83	72
One-To-One	Create	130379	58037	62647
	Read	277	121	235
	Update	47329	28791	42964
	Delete	323	95	95
One-To-Many	Create	70741	83378	95406
	Read	199	181	557
	Update	55661	59296	59669
	Delete	2936	2820	3666
Many-To-Many	Create	63739	69485	84870
	Read	135	114	425
	Update	84769	87757	78830
	Delete	84	86	93

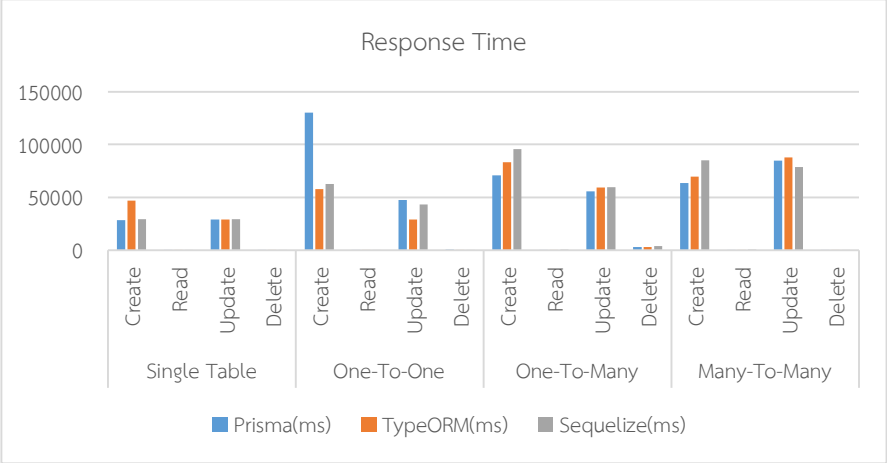

Figure 5. Chart Response Time

จากการทดสอบ Response Time ตาม Table 1. และ Figure 5. สามารถแบ่งเป็น 4 กรณีศึกษาหลัก (Single Table, One-To-One, One-To-Many, Many-To-Many) โดยวัดเวลาที่ใช้ในแต่ละ Operation (Create, Read, Update, Delete) โดยมีการสรุปภาพรวม ดังนี้

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

1. Prisma เด่นในการสร้าง (Create) และอัพเดต (Update) ข้อมูลแบบ Single Table, One-To-Many, Many-To-Many (เร็วกว่า TypeORM 6–18 %) และ เสียเปรียบกับ TypeORM/Sequelize ใน One-To-One (ช้ากว่า~ 40–70 %)

2. TypeORM เด่นในการอ่าน (Read) ข้อมูลที่มีความสัมพันธ์เชิงซ้อน (One-To-One, Many-To-Many) (เร็วกว่า Prisma 15–56 %) และ ช้ากว่า Prisma มากในงาน Create แบบ Single Table (ช้ากว่า) และ One-To-Many/M-to-M

3. Sequelize ประสิทธิภาพกลาง ๆ โดยภาพรวมช้ากว่า Prisma และ TypeORM ในหลายกรณีโดดเด่นเพียงบางจุดเช่น Update ในMany-To-Many (เร็วกว่า Prisma 7 %)

Table 2. Memory Usage Summary

Relationship	Operation	Prisma (MB)	TypeORM (MB)	Sequelize (MB)
Single Table	Create	23.91	10.35	16.16
	Read	1.05	1.32	2.24
	Update	24.71	5	10
	Delete	0	0.05	0.1
One-To-One	Create	24.64	16.16	22.32
	Read	0.57	2.24	8.42
	Update	3.08	8.35	8.98
	Delete	0	0	0
One-To-Many	Create	28.36	24.18	0.33
	Read	4.15	2.66	1.28
	Update	29.39	29.91	1.28
	Delete	0.04	0.07	0
Many-To-Many	Create	24.95	20.95	2.45
	Read	0.61	0.87	1.72
	Update	36.57	11.22	34.85
	Delete	0	0	0.06

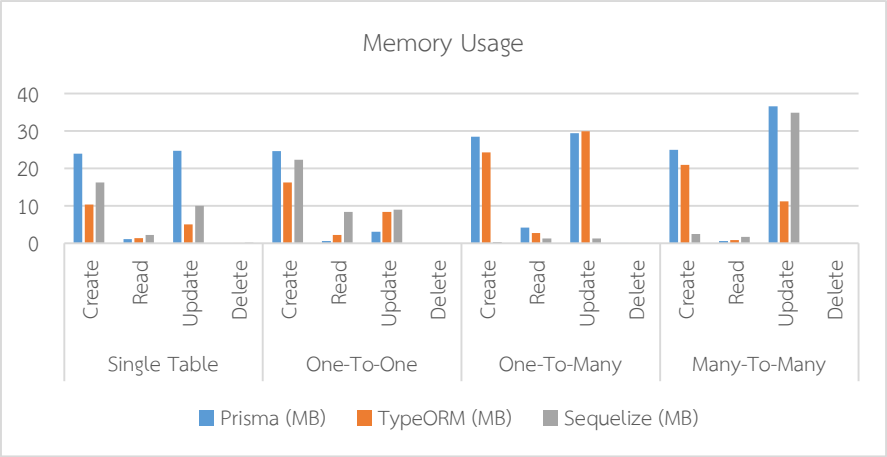

Figure 6. Chart Memory Usage

Citation: Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

จากทดสอบ Memory Usage ตาม Table 2. และ Figure 6. สามารถแบ่งเป็น 4 กรณีศึกษาหลัก (Single Table, One-To-One, One-To-Many, Many-To-Many) โดยวัดการใช้หน่วยความจำ ที่ใช้ในแต่ละ Operation (Create, Read, Update, Delete) โดยมีการสรุปภาพรวม ดังนี้

1. Prisma ใช้หน่วยความจำสูงที่สุดในการ CreateและUpdate ของ Single Table, One-To-Many, Many-To-Many ใน One-To-One จะกิน memory น้อยกว่าคู่แข่งเล็กน้อย (เมื่อเทียบกับ Sequelize) แต่ Read/Updateกลับใช้สูงกว่าอย่างมาก

2. TypeORM กิน memory น้อยกว่า Prisma ในการ Create ทุกกรณี และใช้ memory สูงกว่า Prisma มากเมื่อ อ่านข้อมูลเชิงสัมพันธ์ (One-To-One, Many-To-Many)

3. Sequelize มี memory spike สูงสุดเมื่ออ่าน One-To-One และ อ่าน Many-To-Many ทำงานมีประสิทธิภาพในการใช้หน่วยความจำมากใน One-To-Many (Create/Update ลดลงกว่า 95%)

Table 3. Summary of CPU Utilization comparison

Relationship	Operation	Prisma (ms)	TypeORM (ms)	Sequelize (ms)
Single Table	Create	9500	15500	9203
	Read	47	141	94
	Update	10563	9891	8765
	Delete	15	0	16
One-To-One	Create	38421	20484	21735
	Read	266	234	297
	Update	17328	9469	16734
	Delete	16	16	15
One-To-Many	Create	19235	31532	34453
	Read	218	203	594
	Update	20578	19203	18515
	Delete	16	31	0
Many-To-Many	Create	19640	26328	30078
	Read	94	110	500
	Update	26735	34202	32140
	Delete	0	0	0

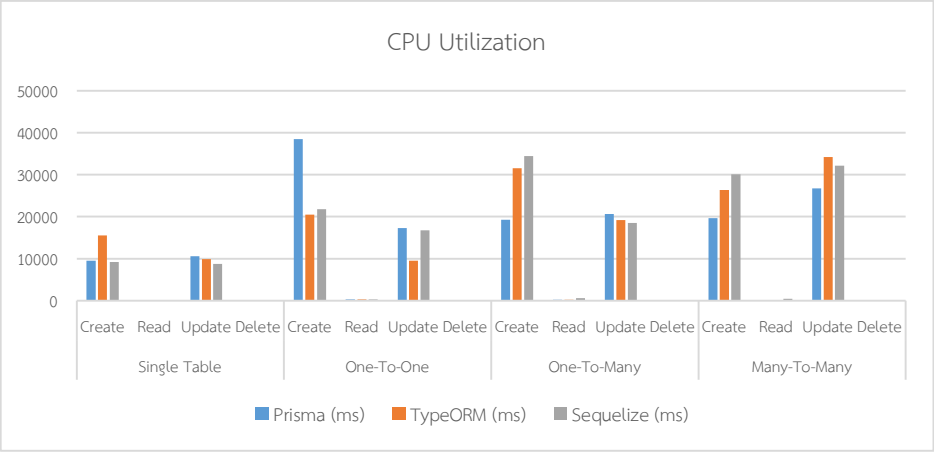

Figure 7. Chart CPU Utilization comparison

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

จากตารางการทดสอบการใช้ CPU Utilization ตาม Table 3. และ Figure 7. สามารถแบ่งเป็น 4 กรณีศึกษาหลัก (Single Table, One-To-One, One-To-Many, Many-To-Many) โดยวัดการใช้ CPU ที่ใช้ในแต่ละ Operation (Create, Read, Update, Delete) โดยมีการสรุปภาพรวม ดังนี้

1. Prisma Single Table ใช้ CPU ต่ำสุดทั้ง Create/Update แต่ Read และ Delete ก็อยู่ในระดับต่ำ. ความสัมพันธ์ซับซ้อน (One-To-One / M-to-M): CPU พุ่งสูงสุด เช่น Create One-To-One (38,421) และ Update M-to-M (26,735)

2. TypeORM Create ใช้ CPU สูงกว่า Prisma มากใน Single Table (+63.2 %) และ One-To-Many (+ 63.9 %) การ Read/Update: ค่อนข้างสมดุล เมื่อเทียบกับ Prisma บางกรณีอาจกิน CPU น้อยกว่า เช่น Update One-To-Many –6.7%

3. Sequelize Read ความสัมพันธ์ซับซ้อนบานปลาย เช่น Read Many-To-Many +431.9 % และ Delete บางกรณีประหยัด CPU (เช่น One-To-Many Delete = 0)

6.2 ผลการเปรียบเทียบกระบวนการแปลงคำสั่ง Object เป็น SQL

งานวิจัยนี้เพื่อให้เข้าใจเชิงลึกถึงพฤติกรรมของ ORM ในกระบวนการทำงานจริง จึงได้เปรียบเทียบโครงสร้างคำสั่ง SQL ที่ถูกสร้างโดย ORM ทั้งสาม ได้แก่ Prisma, TypeORM, และ Sequelize โดยเน้นที่กรณี One-to-One Relationship ในการทำงานแบบ Create ผลลัพธ์ดังแสดง Figure 8. จากภาพจะเห็นได้ว่า

1. Prisma มีลักษณะการสร้าง SQL ที่แบ่งขั้นตอนอย่างชัดเจน เช่น BEGIN, INSERT, และ COMMIT พร้อมกับใช้ RETURNING เพื่อรับค่าที่ถูกสร้าง เช่น id. จุดเด่นของ Prisma คือการควบคุมธุรกรรม (Transaction) โดยอัตโนมัติระหว่างตารางที่มีความสัมพันธ์กัน ซึ่งช่วยให้การทำงานปลอดภัยและลดความซับซ้อนของโค้ดฝั่งผู้พัฒนา

2. TypeORM ใช้รูปแบบ SQL แบบ explicit โดยต้องเริ่ม START TRANSACTION และสิ้นสุดที่ COMMIT. การส่งค่าผ่าน parameter เช่น $1, $2 ทำให้สามารถตรวจสอบและควบคุม flow ได้ละเอียดมาก เหมาะกับระบบที่ต้องการความยืดหยุ่นและการควบคุมเชิงลึกในระดับโค้ด

3. Sequelize สร้าง SQL ที่กระชับ และมักจัดการธุรกรรมแบบอัตโนมัติ (implicit) เว้นแต่ระบุ manual transaction โดยตรง คืนค่าหลายฟิลด์พร้อมกันในคำสั่งเดียว ทำให้เหมาะสำหรับการใช้งานที่ต้องการความเร็วในการพัฒนาและผลลัพธ์ที่ครบถ้วนทันที

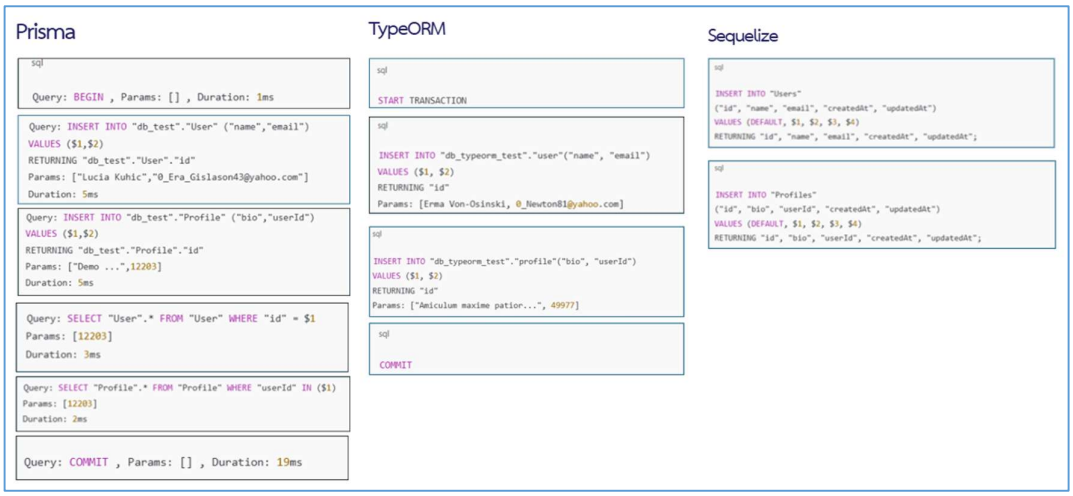

Figure 8. SQL Query Comparison Across Prisma, TypeORM, and Sequelize.

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

7. สรุปและอภิปรายผลการวิจัย (Conclusion and Discussion)

7.1 สรุปผลการวิจัย (Conclusion)

การวิจัยนี้มีวัตถุประสงค์เพื่อเปรียบเทียบประสิทธิภาพของเครื่องมือ Object–Relational Mapping (ORM) ได้แก่ Prisma, TypeORM และ Sequelize ร่วมกับภาษา TypeScript และฐานข้อมูล PostgreSQL โดยทำการทดสอบกับรูปแบบความสัมพันธ์ของตารางที่หลากหลาย ได้แก่ Single Table, One-to-One, One-to-Many และ Many-to-Many พร้อมประเมินผลจาก 3 ตัวชี้วัดหลัก ได้แก่ เวลาในการตอบสนอง (Response Time), การใช้หน่วยความจำ (Memory Usage) และการใช้ CPU (CPU Utilization) จากผลการทดลองสามารถสรุปได้ว่า:

1. ไม่มี ORM ใดที่มีประสิทธิภาพดีที่สุดในทุกด้าน แต่ละตัวมีจุดแข็งและข้อจำกัดของตนเอง
2. TypeORM มีความสมดุลระหว่างประสิทธิภาพและการใช้ทรัพยากร เหมาะกับระบบที่มีความสัมพันธ์ซับซ้อน
3. Prisma เหมาะสำหรับงานที่ต้องการความปลอดภัยของชนิดข้อมูลและความเร็วในการพัฒนา โดยเฉพาะกรณี Single Table และ One-to-Many แต่ใช้ทรัพยากรสูงกว่า
4. Sequelize ใช้หน่วยความจำน้อย เหมาะสำหรับโครงการขนาดเล็กถึงกลางที่ต้องการพัฒนาเร็ว

ผลการวิจัยจึงช่วยยืนยันว่า การเลือกใช้ ORM ควรคำนึงถึงลักษณะงานและข้อจำกัดของระบบ มากกว่าความนิยมเพียงอย่างเดียว

7.2 อภิปรายผลการวิจัย (Discussion)

เมื่อพิจารณาผลการทดลองร่วมกับวัตถุประสงค์การวิจัยและวรรณกรรมที่เกี่ยวข้อง จะพบดังนี้

1. TypeORM แสดงผลลัพธ์ที่น่าสนใจในกรณีการอ่านข้อมูลจากความสัมพันธ์ที่ซับซ้อน เช่น One-to-One และ Many-to-Many ซึ่งสอดคล้องกับแนวทางของ Repository Pattern และ Query Builder ที่ให้ความยืดหยุ่นในการออกแบบคำสั่ง SQL (สอดคล้องกับ Güvercin & Avenoglu (2022)
2. Prisma ให้ความเร็วในกรณี Insert และ Update โดยเฉพาะ Single Table ซึ่งสอดคล้องกับคุณสมบัติ Type Safety และระบบ Transaction อัตโนมัติ ทำให้เหมาะกับทีมพัฒนาสมัยใหม่ (Prisma, n.d.b)
3. Sequelize แม้จะด้อยกว่าด้านประสิทธิภาพในบางกรณี แต่ก็เหมาะกับโครงการที่เน้นการพัฒนาเร็วและมีข้อจำกัดด้านทรัพยากร เช่น RAM ต่ำ ซึ่งตรงกับข้อเสนอในงานวิจัยของ Marchuk et al. (2023)

การเปรียบเทียบในครั้งนี้จึงช่วยยืนยันอีกครั้งว่า ไม่มี ORM ใดดีที่สุดในทุกมิติ แต่ การตัดสินใจควรอิงกับลักษณะของข้อมูล ความซับซ้อนของระบบ และทรัพยากรที่มีอยู่ เป็นหลัก

8. ข้อเสนอแนะงานวิจัย (Recommendation)

8.1 ข้อเสนอแนะเชิงเทคนิค (Technical Recommendation)

1. ควรศึกษาการใช้งาน ORM ควบคู่กับเทคนิคการ Cache เพื่อเพิ่มประสิทธิภาพการเข้าถึงข้อมูล

จากผลการทดลองพบว่า ORM บางตัว (เช่น Sequelize) มีการใช้หน่วยความจำต่ำ แต่มีเวลาในการอ่านข้อมูลสูงในบางกรณี โดยเฉพาะในความสัมพันธ์แบบ Many-to-Many ซึ่งมีค่า queryTimeMs และ CPU Usage สูงอย่างชัดเจน

ดังนั้น ควรพิจารณาใช้ระบบ In-Memory Cache เช่น Redis หรือ Node.js memory cache เพื่อเก็บข้อมูลที่มีการเรียกซ้ำบ่อย ลดการโหลดจากฐานข้อมูลโดยตรง ซึ่งจะช่วยลด latency และภาระ CPU ได้อย่างมีนัยสำคัญ งานวิจัยของ Vaja & Rahevar (2016) ได้แสดงให้เห็นว่า Caching สามารถลด Redundant Data Access ได้อย่างมีประสิทธิภาพ

2. การหลีกเลี่ยงปัญหา N+1 Query ด้วยการ Optimize Query หรือใช้ Query Builder อย่างเหมาะสม

จากผลทดสอบ ORM ที่สนับสนุน Query Builder อย่าง TypeORM พบว่ามีประสิทธิภาพสูงในการอ่านข้อมูลที่มีความสัมพันธ์ซับซ้อน เนื่องจากสามารถรวม Query ได้อย่างยืดหยุ่น ลดจำนวน Query ซ้ำซ้อน

ดังนั้นควรใช้ eager loading, join queries, หรือ tools เช่น Prisma’s include, Sequelize’s include, หรือ TypeORM’s leftJoinAndSelect เพื่อรวม Query แทนการ query ทีละ entity

ข้อเสนอแนะนี้สอดคล้องกับผลการวิจัยของ Chen et al. (2016) ที่ระบุว่า ORM มักมีแนวโน้มเกิด Redundant Data Access หากไม่ได้จัดการ Query อย่างมีประสิทธิภาพ

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

8.2 ข้อเสนอแนะเชิงการประยุกต์ใช้ (Practical Recommendation)

1. นักพัฒนาและองค์กรควรพิจารณาเลือกลักษณะของ ORM ให้เหมาะสมกับประเภทของระบบ เช่น 1. หน่วยความจำหลัก (Eng: RAM – Random Access Memory) เช่น Embedded หรือ Serverless ควรเลือกใช้ Sequelize ซึ่งผลการทดลองแสดงว่าใช้หน่วยความจำน้อยกว่าตัวอื่นอย่างมีนัยสำคัญในหลายกรณี 2. ระบบที่มีความซับซ้อนของข้อมูลและต้องการ Query แบบยืดหยุ่น เช่น ระบบ Dashboard หรือ Back-office ขนาดใหญ่ ควรเลือก TypeORM ที่รองรับ Repository Pattern และ Query Builder ได้ดี 3.ระบบที่ต้องการความแม่นยำของ schema และ Type Safety สูง เช่น ระบบที่เน้นความปลอดภัยของข้อมูลหรือทีมพัฒนาที่ใช้ TypeScript อย่างเข้มข้น ควรเลือก Prisma

2. ควรมีการประเมินความต้องการของระบบในด้านทรัพยากร จากการทดลองแสดงให้เห็นว่า ORM บางตัวใช้ CPU สูงกว่าอีกตัวในกรณีเดียวกันมากถึง 60% ดังนั้นองค์กรควรวัด Resource Budget เช่น CPU/memory limit ของระบบให้ชัดเจนก่อนเลือก ORM เพื่อให้เหมาะสมกับระบบจริง

8.3 ข้อเสนอแนะสำหรับการวิจัยในอนาคต (Recommendations for Future Research)

1. ควรขยายผลการศึกษาไปยัง ORM อื่น ๆ เช่น Kysely, Drizzle ORM, MikroORM หรือ Objection.js เพื่อให้เห็นแนวโน้มการเปลี่ยนแปลงของ performance ตามยุคสมัย โดยเฉพาะ ORM สมัยใหม่ที่เน้น “type-safe query builder” แทน traditional ORM pattern

2. ควรทดสอบร่วมกับฐานข้อมูลอื่น เช่น MySQL, MariaDB หรือ MongoDB ORM บางตัวอาจมี performance ต่างกันอย่างชัดเจนเมื่อใช้งานร่วมกับ database อื่น จึงควรเปรียบเทียบข้ามระบบเพื่อตอบโจทย์งานที่ใช้ stack แตกต่างกัน

3. ศึกษา performance ของ ORM ในสภาพแวดล้อม cloud-native เช่น Kubernetes หรือ Serverless Architecture เนื่องจากสภาพแวดล้อมเหล่านี้มีข้อจำกัดด้านทรัพยากรและความคาดเดาได้น้อย การเข้าใจ performance pattern ของ ORM ภายใต้เงื่อนไขนี้จะมีประโยชน์อย่างมาก

9. องค์ความรู้จากงานวิจัย (Contributions)

งานวิจัยนี้ช่วยให้เข้าใจถึงข้อแตกต่างด้านประสิทธิภาพของ ORM แต่ละตัว (Prisma, TypeORM, Sequelize) ได้ชัดเจนขึ้น ทั้งในแง่เวลาในการตอบสนอง การใช้หน่วยความจำ และ CPU โดยทดสอบกับตารางที่มีความสัมพันธ์ต่างกันและควบคุมตัวแปรอย่างเป็นระบบ นอกจากนี้ ยังได้ใช้วิธีการวัดผลตามหลักสถิติ เช่น การวิเคราะห์ค่าเฉลี่ย และ One-Way ANOVA เพื่อให้เปรียบเทียบได้อย่างมีนัยสำคัญ งานนี้ยังเติมเต็มจากงานวิจัยก่อนหน้า ซึ่งส่วนใหญ่ยังไม่ครอบคลุมการทดสอบ ORM ร่วมกับ TypeScript และ PostgreSQL ในระดับความสัมพันธ์ข้อมูลหลายรูปแบบ สุดท้ายงานวิจัยนี้เสนอแนะแนวทางการเลือกใช้ ORM ให้เหมาะกับลักษณะงานจริง เช่น หากเน้นความเร็วและ type safety อาจเลือก Prisma แต่ถ้าต้องการ query ยืดหยุ่นควรเลือก TypeORM เป็นต้น

10. เอกสารอ้างอิง (References)

Attala, J. (2025). *Comparing-Performance-ORM-typescript-with-PostgreSQL* [Dockerfile]. https://github.com/Jakkapan-a/Comparing-Performance-ORM-typescript-with-PostgreSQL.

Chen, T.-H., Shang, W., Jiang, Z. M., Hassan, A. E., Nasser, M., & Flora, P. (2016). Finding and Evaluating the Performance Impact of Redundant Data Access for Applications that are Developed Using Object-Relational Mapping Frameworks. *IEEE Transactions on Software Engineering, 42*(12), 1148-1161. https://doi.org/10.1109/TSE.2016.2553039.

Güvercin, A. E., & Avenoglu, B. (2022). Performance Analysis of Object-Relational Mapping (ORM) Tools in .Net 6 Environment. *Bilişim Teknolojileri Dergisi, 15*(4), 453–465. https://doi.org/10.17671/gazibtd.1059516.

Marchuk, Y., Dyyak, I., & Makar, I. (2023, September 26-28). Performance Analysis of Database Access: Comparison of Direct Connection, ORM, REST API and GraphQL Approaches. *2023 IEEE 13th International Conference on Electronics and Information Technologies*, 174–176. https://doi.org/10.1109/ELIT61488.2023.10310748.

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.

Monster Connect. (n.d.). *Relational Database*. https://monsterconnect.co.th/relational-database. (In Thai)

Node.js. (n.d.). *Process*. https://nodejs.org/api/process.html?utm_source=chatgpt.com.

Prisma. (n.d.a). *Introspection for PostgreSQL in a TypeScript Project*. https://www.prisma.io/docs/getting-started/setup-prisma/add-to-existing-project/relational-databases/introspection-typescript-postgresql.

Prisma. (n.d.b). *Type Safety*. https://www.prisma.io/docs/orm/prisma-client/type-safety.

Renovate. (2025). *Model Basics*. Sequelize. https://sequelize.org/docs/v6/core-concepts/model-basics.

TypeORM. (n.d.). *Entities*. https://typeorm.io/docs/entity/entities.

Vaja, D. D., & Rahevar, M. (2016, December 19-21). Improve Performance of ORM Caching Using In-Memory Caching. *2016 International Conference on Computing, Analytics and Security Trends*, 112-115. https://doi.org/10.1109/CAST.2016.7914950.

Zaytsev, S. (2025). *Architecture*. https://jestjs.io/docs/architecture.

Citation:
Attala, J., & Khemapatpapan, C. (2025). Comparing the Performance of ORMs. *Journal of Computer and Creative Technology, 3*(2), 201-213. https://doi.org/10.14456/jcct.2025.16.