Технічні науки

УДК 004.4

Голубович П.О.
магістрант
Львівський національний університет
"Львівська політехніка"
Грицюк Ю. І.
доктор технічних наук, професор
кафедри програмного забезпечення
Львівський національний університет
"Львівська політехніка"

Голубович П.О.
студент
Львовский национальный университет
"Львовская политехника"
Грыцюк Ю. И.
доктор технических наук, профессор
кафедры программного обеспечения
Львовский национальный университет
"Львовская политехника"

Holubovych P. O.
student
Lviv National University "Lviv Polytechnic"
Gryciuk Yu. I.
Doctor of Engineering, Professor,
Professor of Software Department
Lviv National University "Lviv Polytechnic"

ПОРІВНЯННЯ ШВИДКОДІЇ НАЯВНИХ ЗАСОБІВ ОБ'ЄКТНО-РЕЛЯЦІЙНОГО ВІДОБРАЖЕННЯ НА БАЗІ ДАНИХ ПРОЕКТУ У СФЕРІ ТЕЛЕКОМУНІКАЦІЙ

СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ СУЩЕСТВУЮЩИХ СРЕДСТВ ОБЬЕКТНО-РЕЛЯЦИОННОГО ОТОБРАЖЕНИЯ НА БАЗЕ ДАННЫХ ПРОЕКТА В СФЕРЕ ТЕЛЕКОММУНИКАЦИЙ

COMPARING THE PERFORMANCE OF EXISTING OBJECT-RELATIONAL MAPPING ASSETS ON THE DATABASE USED IN PROJECT IN TELECOMMUNICATIONS BUSINESS AREA

Анотація: *Наведено порівняння швидкодії наявних засобів об'єктно-реляційного відображення на базі даних проекту у сфері телекомунікацій, а також графічне порівняння швидкодії на різних типах запитів.*

Ключові слова: *SQL Server, бази даних, об'єктно-реляційне відображення, швидкодія, Ado.Net. Entity Framework, NHibernate, LINQ2SQL, програмне забезпечення.*

Аннотация: *Освещено сравнения быстродействия существующих средств объектно-реляционного отображения на базе данных проекта в сфере телекоммуникаций. Приведено графическое сравнение быстродействия на различных типах запросов.*

Ключевые слова: *SQL Server, базы данных, объектно-реляционное отображение, быстродействие, Ado.Net. Entity Framework, NHibernate, LINQ2SQL, программное обеспечение.*

Summary: *Covering performance comparison of existing object-relational mapping on the database used in project in telecommunications business area. Demonstrated graphical comparison of performance on different types of requests.*

Key words: *SQL Server, databases, object-relational mapping, performance, Ado.Net. Entity Framework, NHibernate, LINQ2SQL, software.*

Вступ

Об'єктно-реляційне відображення (ORM) – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи "віртуальну об'єктну базу даних" [1]. У об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу. Необхідно знайти рішення, що перетворить такі об'єкти у форму, в якій вони можуть бути збережені у файлах або базах даних, і які в подальшому можна буде легко витягнути, зі збереженням властивостей об'єктів і відносин між ними.

Для вирішення проблеми зберігання даних існують так звані реляційні системи управління базами даних. Використання реляційної бази даних для зберігання об'єктно-орієнтованих даних приводить до семантичного провалу, змушуючи програмістів писати програмне забезпечення, яке має вміти як обробляти дані в об'єктно-орієнтованому вигляді, так і вміти зберегти ці дані в реляційній формі. Ця постійна потреба в перетворенні між двома різними формами даних не тільки значно знижує продуктивність, але й створює

труднощі для програмістів, оскільки обидві форми даних накладають обмеження одна на одну.

Проблема, яка вирішується об'єктно-реляційним відображенням – усунення семантичного провалу у прикладних програмах [2]. Розроблено безліч пакетів, що уникають потреби в перетворенні об'єктів для зберігання в реляційних базах даних. Однак, використання таких технологій призводить до зниження швидкодії та збільшення використання пам’яті. ORM позбавляє потреби програміста писати велику кількість настанов коду програми, часто одноманітного і схильного до помилок, тим самим значно підвищуючи швидкість процесу розробки ПЗ.

Виклад основного матеріалу

Спробуємо порівняти швидкодію наявних засобів об’єктно-реляційного відображення на базі даних проекту у сфері телекомунікацій.

Тест перший. Для тестування було використано базу даних з проекту в області телекомунікацій, розроблюваного на базі практики. Зокрема таблиці Users, Nodes, NodeVersions і Jobs. Для кожного із способів доступу до даних зроблено 8 простих тестів, які наведено в табл. 1

Табл. 1. Опис проведених тестів

Тест	Опис
Ініціалізація	Створення контексту, читання стрічки підключення.
Отримання всіх замовлень	Проста вибірка всіх рядків з таблиці Nodes
Багаторазове отримання всіх замовлень	Вибірка всіх рядків з таблиці Nodes 3 рази
Отримання всіх продуктів одного замовника	Один складний запит через 3 таблиці
Багаторазове отримання всіх продуктів одного замовника	Виконання складного запиту через 3 таблиці тричі.
Отримання всіх продуктів одного замовника (складна версія)	Покрокове отримання того ж результату, що і в попередньому випадку.
Багаторазове отримання всіх продуктів одного замовника (складна версія)	Покрокове отримання того ж результату, що і в попередньому випадку тричі.
Комбіновані запити: отримання замовлень і продуктів замовника	Синтетичний тест, що емулює різні запити
Навантаження	Всі попередні запити послідовно

Тести були проведені над такими засобами доступу: Ado.Net, LINQ to SQL, Entity Framework, NHibernate.

Спробуємо проаналізувати результати першого тесту.

Табл. 2. Результати виконання тестів у мс

Test	Ado.Net	LINQ to SQL	EF	NH
Initialization	726	3773	3414	19833
Nodes	6047	8320	36437	51373
NodesMultiple	9046	10545	36944	57419
UserJobs	5273	11319	44887	40053
UserJobsMultiple	6047	13593	45428	55920
UserJobsComplex	6772	22445	49113	45326
UserJobsComplexMultiple	9820	24719	51208	53646
Mixed	6772	14319	42758	57419
All	9046	24912	55974	74786

З результатів можна констатувати значне відставання EF і NH від L2S, а також те, наскільки швидко L2S працює з базою – в середньому всього тільки у 2 рази повільніше. При найближчому розгляді можна зробити висновок, що NH працює повільніше, ніж EF, принаймні, в запитах, які повертають невелику кількість даних. NH повільніше матеріалізує об'єкти, ніж EF (запит по Nodes – найпростіший, але в той же час найбільший за кількістю даних). При цьому ініціалізація контексту у EF відбуватися навіть швидше, як у L2S, а NH значно відстає за цим показником.

Також важливим є те, що всі ORM відмінно справляються з кешуванням повторних запитів: 3 запити виконуються не набагато довше, ніж один. Пов'язано це, перш за все, з кешуванням не тільки матеріалізованих об'єктів, чим суттєво економлять час, але і у випадку EF – дерева проміжних запитів Tree). Однакові і навіть схожі запити виконуються набагато швидше. Найбільш показовим є останній тест: при нагрузці один загальний тест тільки у незначній мірі перевищує найповільніший зі своїх компонентів. Він дає можливість побачити, що система добре розподіляє ресурси між запитами.

L2S зав'язаний тільки на один SQL Server і, не будує ніяких проміжних CQT, щоб спілкуватися з великою кількістю провайдерів, як це робить EF. По-друге, EF – не просто ORM, здатна працювати з різними базами даних і на

льоту підміняти їх без зміни коду. EF та NH також володіють різними просунутими функціями маппінга, яких немає в L2S. EF краще за NH працює зі складними записами. Так як Ado.Net не використовує жодних проміжків, то він залишається найшвидшим засобом організації доступу до баз даних.

Тест другий. Спосіб порівняння вибрано за таким принципом. Є написаний код для отримання продуктів по замовнику. Це не складний запит, але він є типовим для багатьох застосунків. При цьому існує два варіанти отримання даних: одним запитом з оператором Join (тест UserJobs) або за допомогою простих операцій одержання спочатку замовника, потім його замовлень, потім деталей замовлення і вже в кінці – продуктів, тобто за допомогою кількох запитів (тест UserJobsComplex).

Для збору метрик було використано SQL Server Profiler. Він дозволяє запускати тести, витягнути SQL-запити та проаналізувати актуальний план виконання. Проведено 4 тести. У першому порівнюються запити з UserJobs для всіх чотирьох способів доступу до даних. У наступних трьох порівняно попарно запити для тесту UserJobs і для тесту UserJobsComplex для кожного з ORM окремо.

Мета першого тесту – порівняти якість генерованого SQL-коду порівняно з конкурентами і найпростішим SQL-запитом, написаним вручну і отримують ці ж дані. Мета інших тестів – з'ясувати, який спосіб отримання даних, через один або декілька запитів, вигідніше і наскільки.

При проведені першого тесту було отримано наступні результати:

- L2S згенерував два запити замість одного. Перший запит витягує користувача, а другий вже виконує реальну роботу. Цей же LINQ-запит в EF перетворюється в один SQL-запит. L2S буде тестуватися з двома запитами, хоча перший запит порівняно з другим виконується практично миттєво.
- NH показав наступні результати: очікувано, що написаний HQL-запит виллється в один SQL-запит. NH згенерував і виконав 17 запитів.

Проаналізувавши запити, відмічено, що це запити до таблиці Supplier, яка в тесті не бере участь, але зате є в відображенні.

- Навіть незважаючи на те, що на вході LINQ-запити в L2S і EF ідентичними, згенеровані SQL-запити відрізняються. Більше того, вони відрізняються не тільки текстом, але і продуктивністю.

Спробуємо проаналізувати результати другого тесту. У першому тесті отримано наступні результати за допомогою утиліти Query cost: класичний ADO.NET – 14%; LINQ 2 SQL – 2 + 46 = 48%; Entity Framework – 18%; Active Record (NHibernate) – 20%.

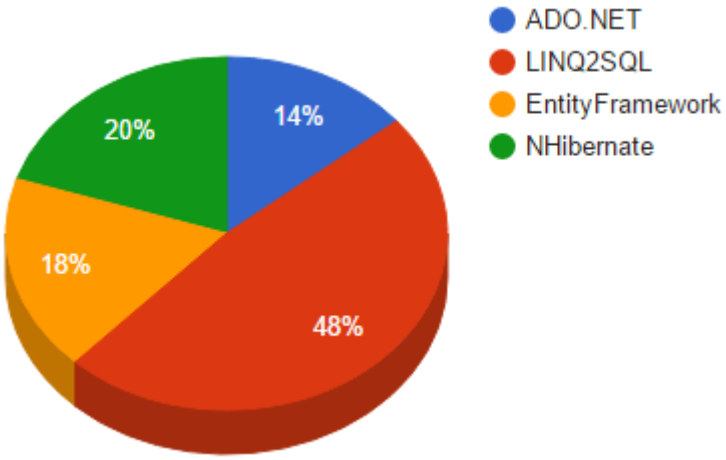

Рис. 1. Порівняння Query cost запитів, згенерованих системами ОРВ

У другому тесті порівняно складний запит з Join і набір простих запитів для L2S: складний запит з Join – 28%, набір простих запитів – 72%.

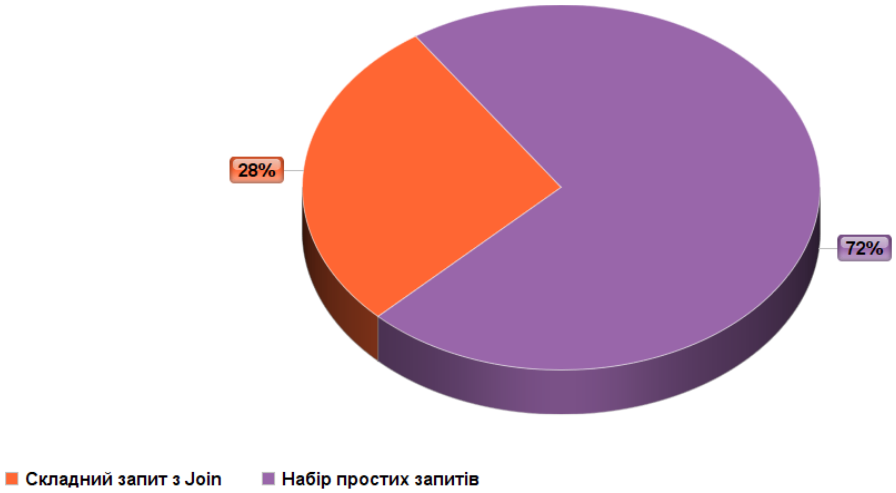

Рис. 2. Порівняння Query cost запитів, згенерованих L2S.

Один запит ефективніше, проте, в цілому, можна сказати, що з точки зору бази даних втрати на віддалене виконання не є великими. Третій тест був присвячений EF: складний запит з Join – 12%, набір простих запитів – 88%. Результати схожі до попереднього тесту.

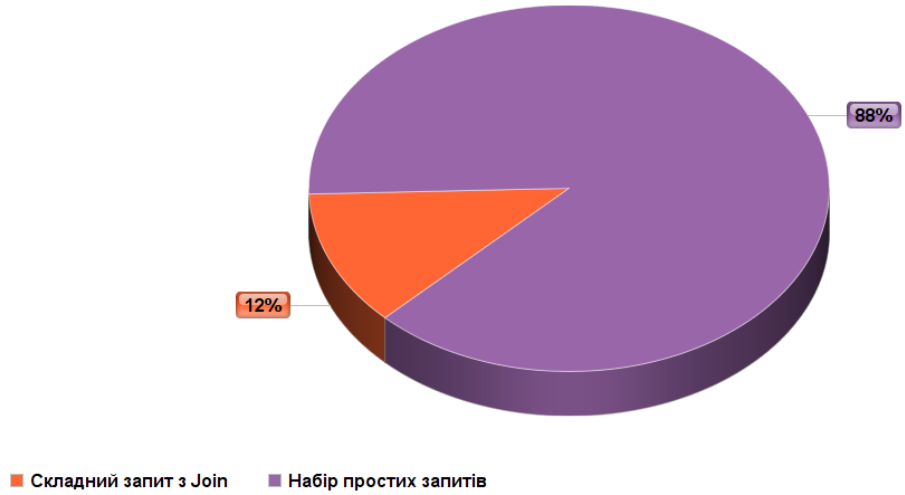

Рис. 3. Порівняння Query cost запитів, згенерованих EF

В четвертому тесті порівняно запити для NH: складний запит з Join – 6%; набір простих запитів – 94%. В NH результати відрізняються найбільше.

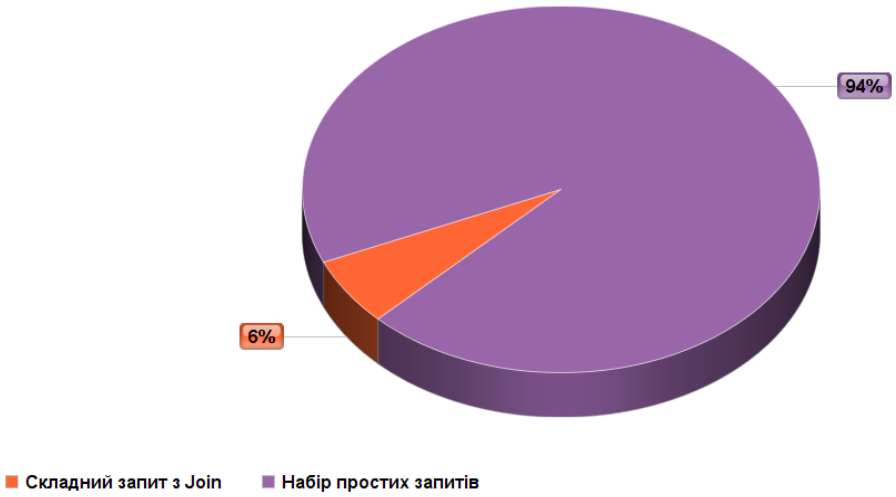

Рис. 4. Порівняння Query cost запитів, згенерованих NH

Висновок: Запит L2S виявився істотно повільніше запиту EF. EF побудував значно ефективніші запити співставні з контрольним запитом SQL. Його план виконання в точності збігся з планом виконання контрольного

запиту. План виконання NH теж непоганий. Також варто відзначити, що середній час виконання цих запитів набагато менше результуючого часу, що враховує додаткові витрати на побудову запиту і матеріалізацію результату (в середньому десь 50-150 мс згідно з профайлером). І тут вже L2S безсумнівно випереджає EF і AR / NH з великим відривом.

Для проектів, що потребують найкращої швидкодії найкращим вибором є Ado.Net, для проектів низької складності, пов'язаних з SQL Server, найкращим вибором є LINQ2SQL. Для проектів підвищеної складності найкращим вибором є EF, через велику кількість функціональних можливостей та зручність використання.

Література

1. What is Object/Relational Mapping? Hibernate Overview. JBOSS Hibernate. Режим доступа: http://hibernate.org/orm/what-is-an-orm/
2. Torsten Stanienda, Douglas Barry, "Solving the Java Object Storage Problem", Computer, vol. 31, issue 11, pp. 33-40, November 1998, doi:10.1109/2.730734