ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ / INFORMATICS AND INFORMATION PROCESSES

DOI: https://doi.org/10.60797/IRJ.2024.148.145

ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ТЕХНОЛОГИЙ ОБЪЕКТНО-РЕЛЯЦИОННОГО ОТОБРАЖЕНИЯ ПРИ ВЗАИМОДЕЙСТВИИ С MICROSOFT SQL SERVER

Научная статья

Ёкубджонов Д.И.[1], Гибадуллин Р.Ф.[2,] *
[2] ORCID : 0000-0001-9359-911X;
[1, 2] Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ, Казань, Российская Федерация

* Корреспондирующий автор (landwatersun[at]mail.ru)

Аннотация

В условиях стремительного роста объемов данных и увеличения требований к производительности современных информационных систем, выбор оптимальной технологии для работы с базами данных становится ключевым аспектом разработки программного обеспечения. В данной статье проводится сравнительный анализ двух широко используемых в экосистеме .NET технологий объектно-реляционного отображения – Dapper и Entity Framework. Исследование фокусируется на оценке производительности этих инструментов при взаимодействии с Microsoft SQL Server в контексте выполнения операций вставки, выборки, обновления и удаления данных. В статье детально рассматриваются критерии оценки, такие как время отклика, потребление системных ресурсов и масштабируемость. Результаты тестирования демонстрируют значительные различия в производительности между Dapper и Entity Framework, что позволяет разработчикам сделать осознанный выбор технологии в зависимости от специфики задач и требований к производительности.

Ключевые слова: ORM, Dapper, Entity Framework, производительность, Microsoft SQL Server, объектно-реляционное отображение, время отклика, потребление ресурсов, масштабируемость, .NET, разработка программного обеспечения, базы данных.

A PERFORMANCE STUDY OF OBJECT-RELATIONAL MAPPING TECHNOLOGIES WHEN INTERACTING WITH MICROSOFT SQL SERVER

Research article

Yoqubjonov D.I.[1], Gibadullin R.F.[2,] *
[2] ORCID : 0000-0001-9359-911X;
[1, 2] Kazan National Research Technical University named after A.N. Tupolev – KAI, Kazan, Russian Federation

* Corresponding author (landwatersun[at]mail.ru)

Abstract

In conditions of rapid growth of data volumes and increasing requirements to the performance of modern information systems, the choice of optimal technology for working with databases becomes a key aspect of software development. This article provides a comparative analysis of two widely used object-relational mapping technologies in the .NET ecosystem – Dapper and Entity Framework. The study focuses on evaluating the performance of these tools when interacting with Microsoft SQL Server in the context of performing data insert, select, update and delete operations. The paper discusses in detail the evaluation criteria such as response time, system resource consumption and scalability. The test results demonstrate significant performance differences between Dapper and Entity Framework, allowing developers to make an informed choice of technology depending on the specific tasks and performance requirements.

Keywords: ORM, Dapper, Entity Framework, performance, Microsoft SQL Server, object-relational mapping, response time, resource consumption, scalability, .NET, software development, databases.

Введение

Современные информационные системы сталкиваются с возрастающими требованиями к эффективности обработки и доступу к данным, что делает взаимодействие с базами данных ключевым элементом их архитектуры [1], [2], [3]. В условиях экспоненциального роста объемов данных и ужесточения требований к производительности, правильный выбор технологии для работы с базами данных приобретает стратегическое значение. Одним из эффективных решений является применение технологий объектно-реляционного отображения (ORM, англ. Object-Relational Mapping), которые позволяют разработчикам взаимодействовать с базами данных посредством объектно-ориентированных подходов. Это не только упрощает написание кода, но и способствует его лучшей поддерживаемости и масштабируемости. В рамках данного исследования будет проведен сравнительный анализ производительности двух широко используемых технологий ORM в экосистеме .NET – Dapper [4] и Entity Framework [5] – при работе с Microsoft SQL Server [6]. Исследование направлено на оценку эффективности данных инструментов в условиях различных сценариев взаимодействия с базой данных, что позволит определить оптимальный выбор ORM технологии в зависимости от специфики задач и требований к производительности.

В современном мире IT объемы данных стремительно растут, требуя создания высокопроизводительных и масштабируемых систем управления базами данных. Бизнесы зависят от эффективности информационных систем,

обеспечивающих быстрый доступ к данным и их обработку. Выбор технологии для работы с базами данных становится ключевой задачей разработки программного обеспечения (ПО).

Технологии объектно-реляционного отображения упрощают взаимодействие с базами данных, предоставляя абстракцию над реляционной моделью данных и позволяя работать в объектно-ориентированном стиле. Это упрощает разработку и уменьшает вероятность ошибок, связанных с ручным написанием SQL-запросов. Однако ORM может влиять на производительность приложения, поэтому важно тщательно анализировать и тестировать производительность различных ORM-технологий для обоснованного выбора.

Entity Framework и Dapper являются двумя популярными ORM-технологиями в экосистеме .NET. Entity Framework предлагает мощные возможности по автоматизации работы с базами данных, но может быть медленнее и требовать больше ресурсов по сравнению с более легковесным Dapper, который предоставляет минимальную абстракцию над SQL и позволяет напрямую выполнять запросы к базе данных с минимальными накладными расходами. В условиях ограниченных ресурсов и высоких требований к производительности, особенно для высоконагруженных систем, выбор между этими технологиями становится критически важным.

Кроме того, современные приложения часто требуют от разработчиков обеспечения высокой производительности на всех уровнях стека, от базы данных до пользовательского интерфейса. Неправильный выбор ORM может привести к значительным потерям производительности, увеличению времени отклика системы и, как следствие, ухудшению пользовательского опыта. В условиях конкуренции и стремительного развития технологий, это может негативно сказаться на бизнес-показателях компании.

Важность исследования производительности технологий ORM [7] также обусловлена тенденцией к миграции приложений в облако, где оптимизация ресурсов и сокращение эксплуатационных расходов играют ключевую роль. Облачные сервисы предоставляют возможность масштабирования, но также требуют тщательного управления ресурсами и затратами. Оптимизация работы с базой данных с помощью правильного выбора ORM может существенно повлиять на эффективность использования облачных ресурсов и снижение расходов.

Таким образом, предоставление разработчикам и архитекторам программного обеспечения объективной и актуальной информации о производительности и эффективности технологий ORM является крайне актуальным, поскольку позволяет оптимизировать разработку и эксплуатацию информационных систем, улучшить производительность приложений, сократить эксплуатационные расходы и повысить удовлетворенность пользователей.

Работа направлена на сравнительный анализ производительности и нагрузки на процессор двух .NET технологий ORM – Dapper и Entity Framework, для определения наиболее эффективного подхода при работе с Microsoft SQL Server.

Практическую ценность работы заключается в предоставлении разработчикам и архитекторам программного обеспечения конкретных данных и рекомендаций по выбору технологии ORM при работе с Microsoft SQL Server. Результаты исследования помогут выбрать наиболее производительное решение, минимизировать нагрузку на процессор и оптимизировать время отклика приложений, что особенно важно в условиях больших объемов данных и высоких требований к производительности.

Анализ производительности ORM-технологий

В данном разделе статьи рассматривается анализ производительности ORM-технологий для среды .NET и SQL Server. Рассматривается процесс выбора ключевых ORM-инструментов, а также формулирование критериев, по которым будет проводиться сравнение производительности этих технологий.

Введём определения, связанные с процедурой анализа производительности ORM-технологий, важные в контексте разрабатываемой методологии:

1. Ключевые ORM-технологии для .NET и SQL Server – инструменты для объектно-реляционного отображения, используемые в среде .NET для взаимодействия с базой данных SQL Server. К ним относятся Entity Framework и Dapper. Эти технологии выбраны для исследования на основе их популярности и широкого использования в промышленной разработке.

2. Время отклика – один из основных критериев оценки производительности ORM-технологий, представляющий собой время, затрачиваемое на выполнение запросов к базе данных. Измеряется в миллисекундах и позволяет оценить быстродействие технологии при выполнении операций чтения и записи данных.

3. Потребление ресурсов – критерий, характеризующий использование системных ресурсов, таких как оперативная память и процессорное время, при работе с ORM-технологией. Оценка потребления ресурсов позволяет определить эффективность использования вычислительных мощностей.

Существует множество ORM-инструментов для объектно-ориентированного программирования: для Java используются Hibernate, TopLink и OpenJPA; для Python – Django, Peewee и SQLAlchemy; для PHP – RedBeanPHP, Doctrine и Propel. В среде .NET широко используются Dapper и Entity Framework. В данном исследовании проводится сравнение характеристик этих ORM-инструментов. Мы выбрали Dapper и Entity Framework для более детального анализа, так как они представляют два подхода к ORM в .NET экосистеме: Dapper как легковесный и высокопроизводительный микро ORM, и Entity Framework как полнофункциональный и гибкий инструмент, предоставляющий широкий спектр возможностей для работы с базами данных.

Критерии оценки

Для оценки производительности различных ORM инструментов, таких как Dapper и Entity Framework (EF), необходимо определить четкие и объективные критерии. Эти критерии помогут оценить эффективность каждого инструмента в различных сценариях использования. В данной работе используются следующие основные критерии: время обработки запроса, потребление ресурсов и масштабируемость. Используются библиотеки BenchmarkDotNet и EtwProfiler [8].

Время обработки является критическим показателем, который отражает скорость выполнения операций. Для оценки времени обработки в данной работе рассматриваются следующие аспекты:

CRUD [9] операции: измерение времени, необходимого для выполнения операций создания (Create), чтения (Read), обновления (Update) и удаления (Delete).

1. Create: время, затраченное на вставку новой записи в базу данных.
2. Read: время, необходимое для чтения данных из базы.
3. Update: время, затраченное на обновление существующей записи.
4. Delete: время, необходимое для удаления записи из базы данных.

Для выполнения этих измерений используется BenchmarkDotNet – высокоточная библиотека для проведения микро-бенчмарков на платформе .NET. Она предназначена для измерения производительности и оптимизации кода. С помощью BenchmarkDotNet можно легко измерить время выполнения, потребление памяти и другие метрики производительности для различных методов и алгоритмов.

Основные возможности BenchmarkDotNet:

1. Точность и надёжность: BenchmarkDotNet минимизирует влияние на измерения внешних факторов, таких как другие процессы, путем выполнения бенчмарков в изолированном контексте. Он автоматически выполняет множество итераций и вычисляет статистически значимые результаты.
2. Автоматическое управление средой: библиотека автоматически настраивает окружение для тестов, например, контролирует работу сборщика мусора, что позволяет получать более стабильные результаты.
3. Поддержка различных платформ: BenchmarkDotNet поддерживает несколько .NET платформ, включая .NET Framework, .NET Core, Mono и CoreRT.
4. Широкий спектр метрик: помимо времени выполнения, BenchmarkDotNet позволяет измерять потребление памяти, количество генерируемого мусора и другие важные параметры.
5. Гибкость конфигурации: библиотека предлагает множество настроек и атрибутов для настройки поведения тестов, включая параметры прогрева, количество итераций и другие.

Для того чтобы добавить BenchmarkDotNet в свой проект, нужно выполнить следующие шаги:

1. Установить пакет NuGet BenchmarkDotNet в проект, выполнив команду Install-Package BenchmarkDotNet.
2. Создать класс, который будет содержать тесты производительности, и пометить его атрибутом [MemoryDiagnoser] и [RankColumn].
3. Запустить тесты, выполнив команду BenchmarkRunner.Run<MyBenchmarks>().

Нагрузка на процессор: измеряется использование CPU во время выполнения операций. Для более точной оценки нагрузки на процессор в операциях обновления (Update) и удаления (Delete) применяется EtwProfiler.

EtwProfiler – это профайлер, который использует Event Tracing for Windows (ETW) для сбора данных о производительности. Этот профайлер помогает разработчикам получать подробные сведения о производительности своего кода, такие как использование процессора, I/O операции, события памяти и многое другое.

После выполнения тестов BenchmarkDotNet создаст ETL файл (Event Trace Log). Этот файл содержит данные о производительности, собранные ETW.С помощью программы Windows Performance Analyzer (WPA) можно открыть и проанализировать собранные данные ЦП.

Разработка тестовых сценариев

Основная цель тестирования – сравнить производительность EF и Dapper при выполнении следующих операций с базой данных:

1. Вставка:
1.1. Вставка одиночной записи.
1.2. Вставка множества записей.
2. Выборка:
2.1. Выборка всех записей.
2.2. Выборка записей по ID.
2.3. Выборка записей с фильтрацией (по имени, дате и т.д.).
3. Поиск записей по заданным критериям, используя различные операторы сравнения (равно, содержит, начинается с и т.д.).
4. Обновление:
4.1. Обновление отдельных записей.
4.2. Обновление нескольких записей.
5. Удаление:
5.1. Удаление отдельных записей.
5.2. Удаление нескольких записей.

Производить замеры будем с помощью библиотеки BenchmarkDotNet, которая будет настроена с дополнительными диагностическими инструментами для отслеживания использования памяти и потоков, а также для профилирования ETW (Event Tracing for Windows) [10].

Основные компоненты и настройки:

1. MemoryDiagnoser: отслеживает потребление памяти во время выполнения тестов, позволяя выявить потенциальные утечки памяти и оценить эффективность использования ресурсов.
2. ThreadingDiagnoser: анализирует работу потоков, помогая выявить узкие места, связанные с параллельным выполнением кода.
3. EtwProfiler: профилирует события ETW. Это позволяет получить детальную информацию о работе операционной системы и выявить скрытые проблемы, влияющие на производительность.

Разработанные тесты позволяют всесторонне оценить производительность различных операций с базой данных с использованием EF и Dapper. Эти тесты помогут определить наиболее эффективные технологии и подходы для различных сценариев работы с данными в .NET приложениях.

Для тестирования производительности используется база данных, содержащая несколько таблиц. Эти таблицы имитируют типичные сценарии работы с данными в реальных приложениях, такие как управление пользователями, заказами и продуктами.

Для генерации данных в таблицах используется библиотека Bogus. Bogus – это мощная библиотека для .NET, позволяющая легко создавать фейковые данные для тестирования. Преимущества использования библиотеки Bogus включают:

1. Простота использования и настройка.
2. Возможность генерации данных различных типов, таких как имена, адреса, даты.
3. Поддержка сложных структур данных и зависимостей между ними.
4. Генерация данных, соответствующих различным культурам и локальным привязанностям.

Проект представляет собой набор тестовых сценариев для оценки производительности операций вставки, выборки, обновления и удаления данных при использовании двух подходов к доступу к данным: Entity Framework и Dapper. Вот краткое описание каждого файла тестирования:

1. Program.cs: этот файл содержит точку входа в приложение. Он настраивает и запускает тестовые сценарии с использованием библиотеки BenchmarkDotNet.
2. InsertTest.cs: этот файл содержит тесты для оценки производительности операций вставки данных. Он сравнивает подходы EF и Dapper как для одиночной вставки, так и для вставки множества записей.
3. SelectTest.cs: в этом файле проводятся тесты для оценки производительности операций выборки данных. Он сравнивает различные способы выборки записей, такие как поиск по ID, фильтрация по имени.
4. SearchTest.cs: этот файл содержит тесты для оценки производительности операций поиска данных. Он сравнивает различные способы поиска записей по критериям, таким как совпадение, начало строки, содержание.
5. FunctionsTest.cs: здесь проводятся тесты для оценки производительности дополнительных функций, предоставляемых EF и Dapper, таких как подсчет записей и постраничный запрос.
6. UpdateTest.cs: этот файл содержит тесты для оценки производительности операций обновления данных. Он сравнивает различные способы обновления записей через EF и Dapper.
7. DeleteTest.cs: здесь проводятся тесты для оценки производительности операций удаления данных. Он сравнивает различные способы удаления записей через EF и Dapper.

С помощью данных тестов мы определим, какой из двух подходов (EF или Dapper) будет наиболее эффективным для конкретных операций с базой данных в проекте. Использование BenchmarkDotNet обеспечивает надежное и объективное сравнение производительности различных реализаций кода.

Для проведения тестов производительности между Entity Framework и Dapper был разработан набор тестовых сценариев, оценивающих основные операции с базой данных. Тестирование проводилось с использованием библиотеки BenchmarkDotNet, настроенной для детального отслеживания использования памяти, потоков и профилирования ETW (Event Tracing for Windows).

Тесты проводились на машине со следующими характеристиками:

1. Процессор: Intel Core i5-8365U.
2. Оперативная память: 16 GB DDR4.
3. Операционная система: Windows 11.

Использовалась среда разработки Visual Studio 2022, а в качестве базы данных – Microsoft SQL Server 2019.

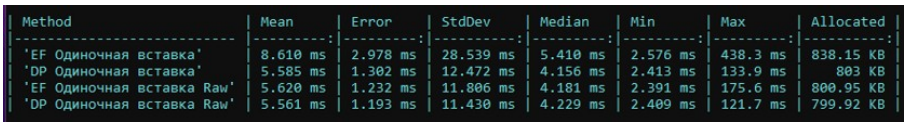
Для получения детальной информации о производительности использовались следующие инструменты:

1. MemoryDiagnoser: для отслеживания использования памяти.
2. ThreadingDiagnoser: для отслеживания работы потоков.
3. EtwProfiler: для профилирования событий ETW.

Были проведены тесты на вставку одной и множества записей в базу данных. Для каждой операции были разработаны тесты как для EF, так и для Dapper. Вот пример кода для тестирования вставки одной записи (листинг 1).

```
[Benchmark(Description = "EF Одиночная вставка")]
public async Task InsertEF()
{
   var student = StudentDataProvider.GetStudentEF();
   await context.AddAsync(student);
   await context.SaveChangesAsync();
}
 [Benchmark(Description = "DP Одиночная вставка")]
public async Task InsertDP()
{
   var student = StudentDataProvider.GetStudentDP();
   await connection.InsertAsync(student);
}
```

Результат выполнения операции ввода вставки в таблицу базы данных на рисунке 1.



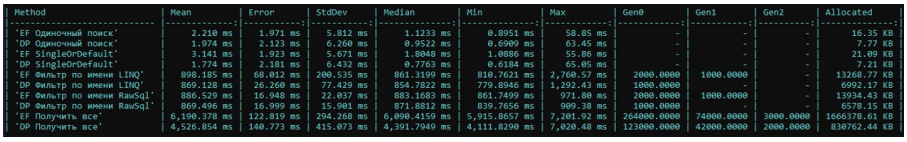
Method	Mean	Error	StdDev	Median	Min	Max	Allocated
'EF Одиночная вставка'	8.610 ms	2.978 ms	28.539 ms	5.410 ms	2.576 ms	438.3 ms	838.15 KB
'DP Одиночная вставка'	5.585 ms	1.302 ms	12.472 ms	4.156 ms	2.413 ms	133.9 ms	803 KB
'EF Одиночная вставка Raw'	5.620 ms	1.232 ms	11.806 ms	4.181 ms	2.391 ms	175.6 ms	800.95 KB
'DP Одиночная вставка Raw'	5.561 ms	1.193 ms	11.430 ms	4.229 ms	2.409 ms	121.7 ms	799.92 KB

Рисунок 1 - Результат сценария InsertTest
DOI: https://doi.org/10.60797/IRJ.2024.148.145.1

Для тестирования выборки данных были разработаны тесты, которые включают одиночный выбор по ID, фильтрацию по имени и получение всех записей. Пример кода одиночного выбора по ID (листинг 2).

```
[Benchmark(Description = "EF Одиночный поиск")]
public async Task EF_Select_Student_By_Id_Linq()
{
   int id = GetRandomId();
   await context.Students.FindAsync(id);
}
 [Benchmark(Description = "DP Одиночный поиск")]
public async Task DP_Select_Student_By_Id_Linq()
{
   int id = GetRandomId();
   await connection.GetAsync<Student>(id);
}
```

Результат выполнения операции выбора из таблицы базы данных указано на рисунке 2.



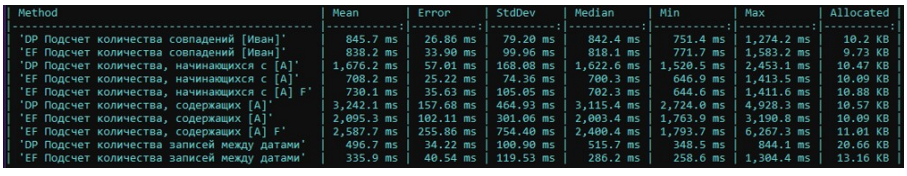
Method	Mean	Error	StdDev	Median	Min	Max	Gen0	Gen1	Gen2	Allocated
'EF Одиночный поиск'	2.210 ms	1.971 ms	5.812 ms	1.1233 ms	0.8951 ms	58.85 ms	-	-	-	16.35 KB
'DP Одиночный поиск'	1.974 ms	2.123 ms	6.260 ms	0.9522 ms	0.6909 ms	63.45 ms	-	-	-	7.77 KB
'EF SingleOrDefault'	3.141 ms	1.923 ms	5.671 ms	1.8048 ms	1.0886 ms	55.86 ms	-	-	-	21.09 KB
'DP SingleOrDefault'	1.774 ms	2.181 ms	6.432 ms	0.7763 ms	0.6184 ms	65.05 ms	-	-	-	7.21 KB
'EF Фильтр по имени LINQ'	898.185 ms	68.012 ms	200.535 ms	861.3199 ms	810.7621 ms	2,760.57 ms	2000.0000	1000.0000	-	13268.77 KB
'DP Фильтр по имени LINQ'	869.128 ms	26.260 ms	77.429 ms	854.7822 ms	779.8946 ms	1,292.43 ms	1000.0000	-	-	6992.17 KB
'EF Фильтр по имени RawSql'	886.529 ms	16.948 ms	22.037 ms	883.1683 ms	861.7499 ms	971.80 ms	2000.0000	1000.0000	-	13934.43 KB
'DP Фильтр по имени RawSql'	869.406 ms	16.999 ms	15.901 ms	871.8812 ms	839.7656 ms	909.38 ms	1000.0000	-	-	6578.15 KB
'EF Получить все'	6,190.378 ms	122.819 ms	294.268 ms	6,090.4159 ms	5,915.8657 ms	7,201.92 ms	264000.0000	74000.0000	3000.0000	1666378.61 KB
'DP Получить все'	4,526.854 ms	140.773 ms	415.073 ms	4,391.7949 ms	4,111.8290 ms	7,020.48 ms	123000.0000	42000.0000	2000.0000	830762.44 KB

Рисунок 2 - Результат сценария SelectTest
DOI: https://doi.org/10.60797/IRJ.2024.148.145.2

Для операций поиска были разработаны тесты на подсчет записей, начинающихся с определенной буквы, содержащих определенную букву и записей в диапазоне дат (листинг 3).

```
[Benchmark(Description = "DP Подсчет количества записей между датами")]
public async Task SearchDateTimeDP()
{
   await connection.CountAsync<Student>(i => i.BirthDate >= startDateTime && i.BirthDate <= endDateTime);
}
[Benchmark(Description = "EF Подсчет количества записей между датами")]
public async Task SearchDateTimeEF()
{
   await context.Students.CountAsync(i => i.BirthDate >= startDateTime && i.BirthDate <= endDateTime);
}
```

Результат выполнения операции поиска из таблицы базы данных указано на рисунке 3.



Method	Mean	Error	StdDev	Median	Min	Max	Allocated
'DP Подсчет количества совпадений [Иван]'	845.7 ms	26.86 ms	79.20 ms	842.4 ms	751.4 ms	1,274.2 ms	10.2 KB
'EF Подсчет количества совпадений [Иван]'	838.2 ms	33.90 ms	99.96 ms	818.1 ms	771.7 ms	1,583.2 ms	9.73 KB
'DP Подсчет количества, начинающихся с [A]'	1,676.2 ms	57.01 ms	168.08 ms	1,622.6 ms	1,520.5 ms	2,453.1 ms	10.47 KB
'EF Подсчет количества, начинающихся с [A]'	708.2 ms	25.22 ms	74.36 ms	700.3 ms	646.9 ms	1,413.5 ms	10.09 KB
'EF Подсчет количества, начинающихся с [A] F'	730.1 ms	35.63 ms	105.05 ms	702.3 ms	644.6 ms	1,411.6 ms	10.88 KB
'DP Подсчет количества, содержащих [A]'	3,242.1 ms	157.68 ms	464.93 ms	3,115.4 ms	2,724.0 ms	4,928.3 ms	10.57 KB
'EF Подсчет количества, содержащих [A]'	2,095.3 ms	102.11 ms	301.06 ms	2,003.4 ms	1,763.9 ms	3,190.8 ms	10.09 KB
'EF Подсчет количества, содержащих [A] F'	2,587.7 ms	255.86 ms	754.40 ms	2,400.4 ms	1,793.7 ms	6,267.3 ms	11.01 KB
'DP Подсчет количества записей между датами'	496.7 ms	34.22 ms	100.90 ms	515.7 ms	348.5 ms	844.1 ms	20.66 KB
'EF Подсчет количества записей между датами'	335.9 ms	40.54 ms	119.53 ms	286.2 ms	258.6 ms	1,304.4 ms	13.16 KB

Рисунок 3 - Результат сценария SearchTest
DOI: https://doi.org/10.60797/IRJ.2024.148.145.3

Были проведены тесты на обновление записей как с использованием EF, так и с использованием Dapper (листинг 4).

```
[Benchmark(Description = "DP Обновление записи")]
public async Task UpdateSingleDP()
{
   var user = GetRandomStudent();
   user.FirstName = user.FirstName.ToUpper();
```

```
    await connection.UpdateAsync(user);
}
 [Benchmark(Description = "EF Обновление записи")]
public async Task UpdateSingleEF()
{
    var user = GetRandomStudent();
    user.FirstName = user.FirstName.ToUpper();
    context.Update(user);
    await context.SaveChangesAsync();
}
```

Результат выполнения операции обновление записей таблицы в базы данных указано на рисунке 4.

Method	Mean	Error	StdDev	Median	Min	Max	Allocated
'DP Обновление записи'	1.629 ms	1.3759 ms	4.057 ms	1.1443 ms	0.9615 ms	41.63 ms	6.86 KB
'EF Обновление записи'	2.688 ms	3.3671 ms	9.928 ms	1.5941 ms	1.3950 ms	100.84 ms	45.13 KB
'DP Обновление записи Raw'	1.206 ms	0.9023 ms	2.660 ms	0.9209 ms	0.7795 ms	27.49 ms	5.53 KB
'EF Обновление записи Raw'	1.126 ms	0.4923 ms	1.452 ms	0.9539 ms	0.8175 ms	15.45 ms	6.85 KB

Рисунок 4 - Результат сценария UpdateTest
DOI: https://doi.org/10.60797/IRJ.2024.148.145.4

Были проведены тесты на удаление записей (листинг 5).

```
[Benchmark(Description = "DP Удаление")]
public async Task DeleteSingleDP()
{
    var student = GetRandomStudent();
    await connection.DeleteAsync(student);
}
 [Benchmark(Description = "EF Удаление")]
public async Task DeleteSingleEF()
{
    var student = GetRandomStudent();
    context.Students.Remove(student);
    await context.SaveChangesAsync();
}
```

Результат выполнения операции удаление записей таблицы в базы данных указано на рисунке 5.

Method	Mean	Error	StdDev	Median	Min	Max	Allocated
'DP Удаление'	1.763 ms	1.140 ms	4.826 ms	1.0982 ms	0.9579 ms	44.59 ms	5.47 KB
'EF Удаление'	2.599 ms	2.703 ms	11.443 ms	1.4034 ms	1.2298 ms	124.46 ms	15.27 KB
'DP Удаление Raw'	1.649 ms	1.143 ms	4.840 ms	1.0332 ms	0.8147 ms	62.32 ms	4.98 KB
'EF Удаление Raw'	1.864 ms	1.610 ms	6.818 ms	0.9911 ms	0.7270 ms	79.03 ms	6.28 KB

Рисунок 5 - Результат сценария DeleteTest
DOI: https://doi.org/10.60797/IRJ.2024.148.145.5

Расшифровка столбцов таблицы результатов:

1. Method: наименования методов тестирования.
2. Mean: среднее время выполнения операций в миллисекундах.
3.Error: возможное отклонение среднего значения.
4. StdDev: стандартное отклонение времени выполнения.
5. Median: медианное время выполнения.
6. Min: минимальное время выполнения.
7. Max: максимальное время выполнения.
8. Allocated: объем памяти, выделенный для выполнения операций.
9. insertRowCount: количество строк, которые были вставлены в таблицу.

В ходе проведения тестов с использованием библиотеки BenchmarkDotNet было использовано EtwProfiler для получения результата анализа нагрузку на процессор. Данные профилирования выводились в папку *\src\ConsoleApp\bin\Release\net8.0\BenchmarkDotNet.Artifacts*. Результирующие файлы:

1. DeleteSingleDP.etl.
2. DeleteSingleEF.etl.
3. DeleteSingleDPRaw.etl.
4. DeleteSingleEFRaw.etl.
5. UpdateSingleDP.etl.
6. UpdateSingleEF.etl.
7. UpdateSingleDPRaw.etl.

8. UpdateSingleEFRaw.etl.

На рисунках с 6 по 13 представлены результаты тестов, демонстрирующие нагрузку процессора при выполнении операций удаления и обновления записей с использованием Dapper и Entity Framework:

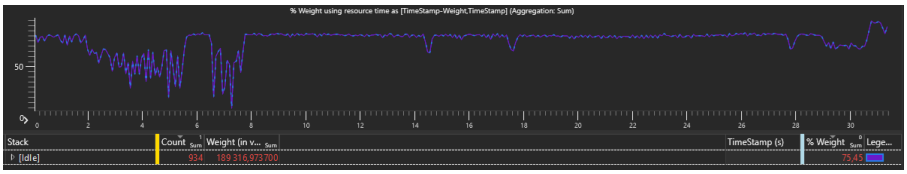


Рисунок 6 - Результат сценария DeleteSingleDP
DOI: https://doi.org/10.60797/IRJ.2024.148.145.6

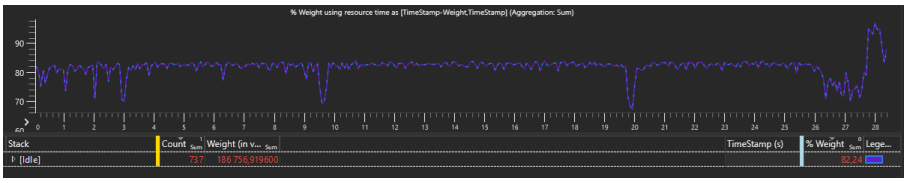


Рисунок 7 - Результат сценария DeleteSingleEF
DOI: https://doi.org/10.60797/IRJ.2024.148.145.7

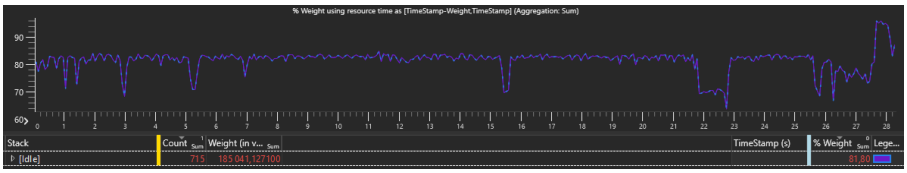


Рисунок 8 - Результат сценария DeleteSingleDPRaw
DOI: https://doi.org/10.60797/IRJ.2024.148.145.8

Рисунок 9 - Результат сценария DeleteSingleEFRaw
DOI: https://doi.org/10.60797/IRJ.2024.148.145.9

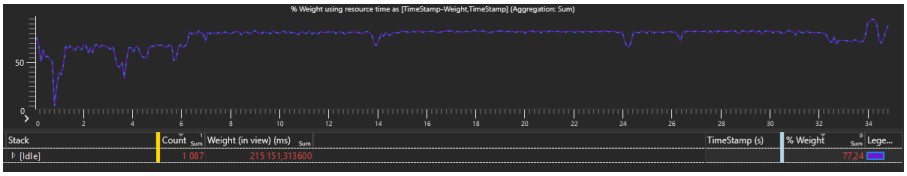


Рисунок 10 - Результат сценария UpdateSingleDP
DOI: https://doi.org/10.60797/IRJ.2024.148.145.10

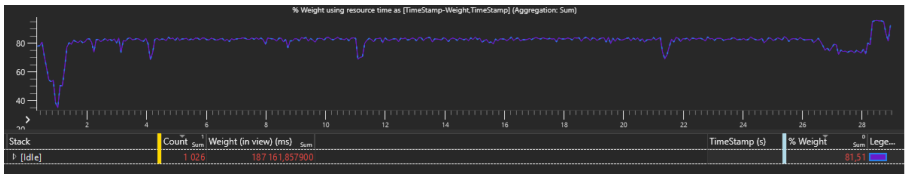


Рисунок 11 - Результат сценария UpdateSingleEF
DOI: https://doi.org/10.60797/IRJ.2024.148.145.11

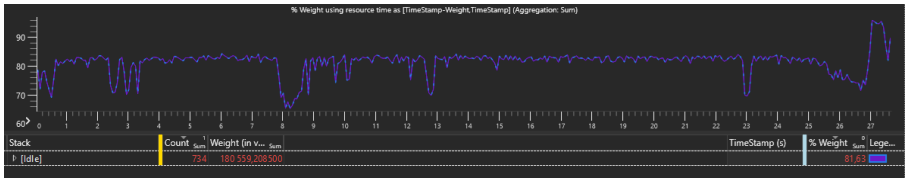


Рисунок 12 - Результат сценария UpdateSingleDPRaw
DOI: https://doi.org/10.60797/IRJ.2024.148.145.12

Рисунок 13 - Результат сценария UpdateSingleEFRawpng
DOI: https://doi.org/10.60797/IRJ.2024.148.145.13

Как видно из результатов, при использовании Dapper нагрузка на процессор в среднем по операциям нагрузка на процессор ниже по сравнению с использованием Entity Framework. Это связано с несколькими ключевыми факторами:

1. Более низкий уровень абстракции: Dapper является микро ORM и работает напрямую с SQL-запросами, что позволяет избежать накладных расходов, связанных с обработкой запросов на более высоком уровне абстракции, как это делает EF.

2. Меньше генерации SQL: Dapper использует заранее подготовленные SQL-запросы или непосредственно переданные строки SQL, что уменьшает нагрузку на процессор при генерации запросов.

3. Оптимизация выполнения запросов: Dapper использует простую и эффективную обработку результатов запросов, что уменьшает общую нагрузку на процессор по сравнению с более сложной обработкой данных в EF.

4. Минимизация внутренней логики: Dapper фокусируется на минималистичной логике взаимодействия с базой данных, избегая сложных механизмов трекинга изменений и прочих дополнительных функций, которые присутствуют в EF.

Эти особенности делают Dapper более производительным инструментом для выполнения операций, таких как вставка, обновление и удаление записей в базе данных, особенно в сценариях, где важна производительность и минимальная нагрузка на процессор.

Разработка рекомендаций

Выбор между Dapper и Entity Framework зависит от множества факторов, таких как требования к производительности, сложность бизнес-логики, удобство разработки, а также опыт и предпочтения команды разработчиков. Рассмотрим более детально случаи, когда целесообразно использовать каждый из этих инструментов.

Dapper, благодаря своей легковесности, показывает значительно меньшую нагрузку на процессор по сравнению с EF в тестах. Это делает его оптимальным выбором для приложений, где производительность является критически важным фактором. Высокая скорость выполнения Dapper достигается за счет отсутствия накладных расходов, связанных с генерацией SQL-запросов и отслеживанием изменений, характерных для EF. Этот инструмент предоставляет разработчикам возможность писать собственные SQL-запросы, обеспечивая полный контроль над их выполнением, что особенно важно при оптимизации сложных запросов и использовании специфичных для базы данных функций. В отличие от EF, который генерирует SQL-запросы на основе LINQ [11], Dapper использует заранее подготовленные SQL-запросы, что минимизирует накладные расходы и ускоряет выполнение операций. Это делает его идеальным для выполнения простых операций CRUD (вставка, обновление, удаление, выборка). Если приложение в основном выполняет такие операции, использование Dapper может быть более эффективным и рациональным выбором. Дополнительно, Dapper легко интегрируется в проект и требует минимального объема кода для выполнения базовых операций с базой данных.

Для обработки больших объемов данных, где требуется высокая производительность, Dapper предлагает значительные преимущества. Его архитектура позволяет более эффективно масштабировать приложение, так как он работает ближе к низкоуровневым механизмам доступа к данным, обеспечивая минимальные накладные расходы и более быструю обработку данных.

С другой стороны, Entity Framework предоставляет более высокий уровень абстракции, что позволяет разработчикам работать с объектами домена вместо написания SQL-запросов вручную. Это существенно упрощает разработку, делает код более читаемым и легко поддерживаемым. EF особенно полезен в тех случаях, когда требуется автоматическое управление сложными отношениями между сущностями, такими как один-к-одному, один-ко-многим и многие-ко-многим. Он обеспечивает встроенные механизмы отслеживания изменений, которые автоматически генерируют соответствующие SQL-запросы для сохранения изменений в базе данных. Кроме того, EF поддерживает ленивую загрузку данных, что позволяет загружать связанные данные только по мере необходимости, оптимизируя использование памяти и повышая производительность приложения.

Использование LINQ в EF упрощает написание и чтение запросов, поскольку LINQ-запросы компилируются в SQL-запросы, обеспечивая интеграцию с базой данных без необходимости написания SQL-кода. Автоматическая генерация SQL-запросов на основе выражений LINQ и модели данных снижает вероятность ошибок и ускоряет процесс разработки.

Если команда разработчиков уже обладает значительным опытом работы с EF, его использование может быть более продуктивным, так как существующие знания и наработки могут сократить время на обучение и внедрение новых решений. Кроме того, EF предоставляет удобные средства для автоматического управления миграциями базы данных, что упрощает процесс обновления схемы базы данных при изменении модели данных.

Заключение

Проведенное исследование подтвердило значимость правильного выбора ORM технологии для обеспечения высокой производительности приложений, работающих с базами данных. Сравнительный анализ Dapper и Entity Framework продемонстрировал, что каждая из этих технологий обладает своими преимуществами и недостатками, которые могут быть критическими в зависимости от специфики проекта.

Dapper показал высокую производительность, особенно в операциях с минимальными накладными расходами на процессор и память. Это делает его предпочтительным выбором для приложений, где критически важны скорость выполнения запросов и низкие затраты на ресурсы, таких как высоконагруженные системы с большими объемами данных. Его простота и эффективность при выполнении CRUD операций делают его оптимальным выбором для сценариев, где требуется минимизация времени отклика системы.

С другой стороны, Entity Framework предоставляет более высокий уровень абстракции и автоматизации, что значительно упрощает разработку сложных приложений с богатой бизнес-логикой. EF особенно полезен в проектах, где требуется работа с объектами домена и сложными отношениями между сущностями. Возможности автоматического управления миграциями базы данных и интеграция с LINQ позволяют ускорить разработку и повысить читаемость и поддерживаемость кода.

Таким образом, выбор между Dapper и Entity Framework должен основываться на конкретных требованиях проекта. Dapper подходит для случаев, когда производительность и минимальные накладные расходы являются приоритетными, тогда как Entity Framework является предпочтительным для проектов, где важны удобство разработки, автоматизация и поддержка сложных отношений между данными.

Результаты данного исследования могут быть полезны для разработчиков и архитекторов, которые принимают решения о выборе ORM технологии для своих проектов, обеспечивая оптимальный баланс между производительностью, удобством разработки и поддерживаемостью кода.

Конфликт интересов

Не указан.

Рецензия

Белашова Е.С., Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ, Казань, Российская Федерация

DOI: https://doi.org/10.60797/IRJ.2024.148.145.14

Conflict of Interest

None declared.

Review

Belashova E.S., Kazan National Research Technical University named after A.N. Tupolev – KAI, Kazan, Russian Federation

DOI: https://doi.org/10.60797/IRJ.2024.148.145.14

Список литературы / References

1. Савельев А.Г. Анализ эффективности использования графических процессоров для ускорения обработки SQL-запросов / А.Г. Савельев, И.С. Вершинин, Р.Ф. Гибадуллин // Информационные технологии и нанотехнологии (ИТНТ-2017) : сборник трудов III международной конференции и молодежной школы, Самара, 25—27 апреля 2017 года / Самарский национальный исследовательский университет имени академика С.П. Королева. — Самара: Новая техника, 2017. — С. 1672–1675.

2. Ямалеева Г.Н. Оптимизация исполнения SQL-запросов к базам данных под управлением MySQL / Г.Н. Ямалеева, М.Ю. Перухин, Р.Ф. Гибадуллин // Информационные технологии и математическое моделирование (ИТММ-2017) : Материалы XVI Международной конференции имени А.Ф. Терпугова, Казань, 29 сентября — 03 октября 2017 года. — Казань: Издательство научно-технической литературы, 2017. — Т. 2. — С. 239–241.

3. Гибадуллин Р.Ф. Ускорение обработки SQL-запросов к базам данных на GPU посредством аппаратно-программной платформы NVIDIA CUDA / Р.Ф. Гибадуллин, А.Г. Савельев, М.Ю. Перухин // Вестник Технологического университета. — 2016. — Т. 19. — № 20. — С. 110–116.

4. Отинчиев А.К. Использование Dapper C# в программировании / А.К. Отинчиев, Л.Г. Касенова // Актуальные вопросы технических наук : материалы V Международной научной конференции, Санкт-Петербург, 20—23 февраля 2019 года. — Санкт-Петербург: Свое издательство, 2019. — С. 5–8.

5. Макаров О.С. Краткий обзор технологии Entity Framework / О.С. Макаров // Наукосфера. — 2020. — № 7. — С. 125–128.

6. Оганнесян Д.А. Работа с хранилищами данных в Microsoft SQL Server / Д.А. Оганнесян, Е.И. Чигарина // Перспективные информационные технологии (ПИТ 2019) : Труды Международной научно-технической конференции, Самара, 24—26 июня 2019 года / Под ред. С.А. Прохорова. — Самара: Самарский научный центр РАН, 2019. — С. 81–83.

7. Судник О.А. Использование технологии ORM в работе с базами данных на языках программирования высокого уровня / О.А. Судник, Н.И. Белодед // Технологические инновации и научные открытия : Сборник научных статей по материалам XV Международной научно-практической конференции, Уфа, 17 мая 2024 года. — Уфа: Вестник науки, 2024. — С. 605–608.

8. Khan O.M.A. C# 7 and. NET Core 2.0 High Performance: Build highly performant, multi-threaded, and concurrent applications using C# 7 and. NET Core 2.0 / O.M.A. Khan. — Packt Publishing Ltd, 2018.

9. Chakraborty S. CRUD Operation on WordPress Database Using C# SQL Client / S. Chakraborty, P.S. Aithal // International Journal of Case Studies in Business, IT, and Education. — 2023. — P. 138–149. — DOI: 10.47992/ijcsbe.2581.6942.0313.

10. Assi M.J. Root Cause Analysis And Improvement In Windows System Based On Windows Performance Toolkit WPT / M.J. Assi, A.A. Fahad, B. Al-Sarray // Iraqi Journal of Science. — 2022. — P. 5046–5057. — DOI: 10.24996/ijs.2022.63.11.39.

11. Кузнецов А.С. Исследование технологии доступа к данным LINQ to SQL / А.С. Кузнецов, И.Ю. Балашова // Информационные ресурсы и системы в экономике, науке и образовании : Сборник статей VII Международной научно-практической конференции, Пенза, 27—28 апреля 2017 года. — Пенза: Приволжский Дом знаний, 2017. — С. 41–45.

Список литературы на английском языке / References in English

1. Savel'ev A.G. Analiz jeffektivnosti ispol'zovanija graficheskih processorov dlja uskorenija obrabotki SQL-zaprosov [Analysis of the efficiency of using graphics processors to accelerate SQL-queries processing] / A.G. Savel'ev, I.S. Vershinin, R.F. Gibadullin // Informacionnye tehnologii i nanotehnologii (ITNT-2017) : sbornik trudov III mezhdunarodnoj konferencii i molodezhnoj shkoly, Samara, 25—27 aprelja 2017 goda [Information technologies and nanotechnologies (ITNT-2017): proceedings of the III International Conference and Youth School, Samara, 25-27 April 2017] / Samara National Research University named after Academician S.P. Korolev. — Samara: New Technology, 2017. — P. 1672–1675. [in Russian]

2. Jamaleeva G.N. Optimizacija ispolnenija SQL-zaprosov k bazam dannyh pod upravleniem MySQL [Optimization of SQL-queries execution to databases under MySQL control] / G.N. Jamaleeva, M.Ju. Peruhin, R.F. Gibadullin // Informacionnye tehnologii i matematicheskoe modelirovanie (ITMM-2017) : Materialy XVI Mezhdunarodnoj konferencii imeni A.F. Terpugova, Kazan', 29 sentjabrja — 03 oktjabrja 2017 goda [Information Technologies and Mathematical Modelling (ITMM-2017): Proceedings of the XVI International Conference named after A.F. Terpugov, Kazan, September 29 – October 03, 2017]. — Kazan: Publishing House of Scientific and Technical Literature, 2017. — Vol. 2. — P. 239–241. [in Russian]

3. Gibadullin R.F. Uskorenie obrabotki SQL-zaprosov k bazam dannyh na GPU posredstvom apparatno-programmnoj platformy NVIDIA CUDA [Acceleration of SQL-queries processing to databases on GPU by means of NVIDIA CUDA hardware-software platform] / R.F. Gibadullin, A.G. Savel'ev, M.Ju. Peruhin // Vestnik Tehnologicheskogo universiteta [Bulletin of Technological University]. — 2016. — Vol. 19. — № 20. — P. 110–116. [in Russian]

4. Otinchiev A.K. Ispol'zovanie Dapper C# v programmirovanii [The use of Dapper C# in programming] / A.K. Otinchiev, L.G. Kasenova // Aktual'nye voprosy tehnicheskih nauk : materialy V Mezhdunarodnoj nauchnoj konferencii, Sankt-Peterburg, 20—23 fevralja 2019 goda [Topical issues of technical sciences: proceedings of the V International Scientific Conference, Saint-Petersburg, 20-23 February 2019]. — St. Petersburg: Own Publishing House, 2019. — P. 5–8. [in Russian]

5. Makarov O.S. Kratkij obzor tehnologii Entity Framework [A brief overview of Entity Framework technology] / O.S. Makarov // Naukosfera [Sciencesphere]. — 2020. — № 7. — P. 125–128. [in Russian]

6. Ogannesjan D.A. Rabota s hranilishhami dannyh v Microsoft SQL Server [Working with data warehouses in Microsoft SQL Server] / D.A. Ogannesjan, E.I. Chigarina // Perspektivnye informacionnye tehnologii (PIT 2019) : Trudy Mezhdunarodnoj nauchno-tehnicheskoj konferencii, Samara, 24—26 ijunja 2019 goda [Perspective Information Technologies (PIT 2019): Proceedings of the International Scientific and Technical Conference, Samara, 24-26 June 2019] / Ed.by S.A. Prohorov. — Samara: Samara Scientific Centre of RAS, 2019. — P. 81–83. [in Russian]

7. Sudnik O.A. Ispol'zovanie tehnologii ORM v rabote s bazami dannyh na jazykah programmirovanija vysokogo urovnja [Use of ORM technology in working with databases in high-level programming languages] / O.A. Sudnik, N.I. Beloded // Tehnologicheskie innovacii i nauchnye otkrytija : Sbornik nauchnyh statej po materialam XV Mezhdunarodnoj nauchno-prakticheskoj konferencii, Ufa, 17 maja 2024 goda [Technological innovations and scientific discoveries: Collection of scientific articles on the materials of XV International Scientific and Practical Conference, Ufa, 17 May 2024]. — Ufa: Bulletin of Science, 2024. — P. 605–608. [in Russian]

8. Khan O.M.A. C# 7 and. NET Core 2.0 High Performance: Build highly performant, multi-threaded, and concurrent applications using C# 7 and. NET Core 2.0 / O.M.A. Khan. — Packt Publishing Ltd, 2018.

9. Chakraborty S. CRUD Operation on WordPress Database Using C# SQL Client / S. Chakraborty, P.S. Aithal // International Journal of Case Studies in Business, IT, and Education. — 2023. — P. 138–149. — DOI: 10.47992/ijcsbe.2581.6942.0313.

10. Assi M.J. Root Cause Analysis And Improvement In Windows System Based On Windows Performance Toolkit WPT / M.J. Assi, A.A. Fahad, B. Al-Sarray // Iraqi Journal of Science. — 2022. — P. 5046–5057. — DOI: 10.24996/ijs.2022.63.11.39.

11. Kuznecov A.S. Issledovanie tehnologii dostupa k dannym LINQ to SQL [Research of LINQ to SQL data access technology] / A.S. Kuznecov, I.Ju. Balashova // Informacionnye resursy i sistemy v jekonomike, nauke i obrazovanii : Sbornik statej VII Mezhdunarodnoj nauchno-prakticheskoj konferencii, Penza, 27—28 aprelja 2017 goda [Information resources and systems in economics, science and education: Collection of articles of the VII International Scientific and Practical Conference, Penza, 27-28 April 2017]. — Penza: Volga Region House of Knowledge, 2017. — P. 41–45. [in Russian]