УДК 004.624

ОПТИМИЗАЦИЯ BULK-ОПЕРАЦИЙ В СИСТЕМАХ МИГРАЦИИ ДАННЫХ НА C#

Мифтахутдинов И.Р., студент гр. 4411, IV курс, Кремлёва Э.Ш., к.т.н., доцент
Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ, г. Казань

Аннотация

Массовая миграция данных является неотъемлемой частью процессов модернизации и интеграции информационных систем. При этом эффективность используемых методов вставки напрямую влияет на производительность всей системы. В данной статье рассматриваются различные подходы к bulk-вставке данных в контексте языка программирования C# и СУБД PostgreSQL. Особое внимание уделяется сравнению производительности ORM-инструментов и нативных механизмов, таких как команда COPY, реализуемая через библиотеку Npgsql. Проведён теоретический и практический анализ, выявлены узкие места в популярных реализациях, представлены бенчмарки и даны рекомендации по выбору оптимального метода вставки данных.

Ключевые слова: миграция данных, массовая вставка, производительность, PostgreSQL, Npgsql, COPY, Entity Framework, C#, оптимизация, масштабируемость.

Введение

С развитием информационных технологий растёт объём хранимых и обрабатываемых данных, что делает проблему эффективной миграции критически важной. Особенно остро она встаёт при переходе между платформами, обновлении баз данных или интеграции разнородных систем. Стандартные методы вставки, такие как построчная запись с использованием ORM (Object-Relational Mapping), становятся неэффективными при работе с большими объёмами информации, т.к. создают значительные накладные расходы на установку соединений, генерацию SQL-запросов и транзакционную обработку.

Наиболее перспективным направлением в этой области являются bulk-операции — механизмы пакетной вставки данных, минимизирующие число обращений к СУБД. Среди них особенно выделяется команда COPY в PostgreSQL, которая обеспечивает быструю и надёжную загрузку больших массивов информации. Интеграция данного подхода в C#-приложения возможна с помощью библиотеки Npgsql и её метода BeginBinaryImport,

позволяющего передавать данные напрямую в бинарном формате, что исключает избыточную обработку и повышает общую производительность.

Для понимания различий в эффективности необходимо рассмотреть архитектурные особенности и механизмы работы наиболее распространённых подходов к вставке данных [5].

1. Построчная вставка через ORM (например, Entity Framework). ORM (Object-Relational Mapping) представляет собой слой абстракции между объектной моделью приложения и реляционной базой данных. При вставке данных каждый объект преобразуется в SQL-команду INSERT, которая исполняется отдельно. Это приводит к следующим издержкам:

- каждое обращение требует установления соединения, подготовки запроса, его компиляции и исполнения;
- сериализация и десериализация данных накладывают дополнительные расходы на процессор и память;
- транзакции, обрабатываемые на уровне приложения, неэффективны при массовой вставке, так как отсутствует агрегирование операций. Таким образом, при объёмах от сотен тысяч строк и выше производительность стремительно падает. Такой подход подходит лишь для небольших объемов данных или сценариев с высокой степенью бизнес-логики.

2. Пакетная вставка через EF Core Bulk Extensions. EF Core Bulk Extensions — это надстройка над Entity Framework, позволяющая выполнять вставку данных блоками. Вместо генерации множества одиночных запросов, формируются пакеты с множеством строк, что снижает нагрузку на сетевой канал и базу. Тем не менее, остаются ограничения:

- каждый пакет всё равно требует отдельной команды INSERT с множеством значений;
- объём пакета ограничен параметрами СУБД;
- выполнение пакетов идёт последовательно, что ограничивает масштабируемость. Кроме того, при больших объёмах данных возникает проблема разбивки на батчи, управления транзакциями и контроля ошибок на уровне пакета.

3. Нативный механизм COPY через Npgsql. Команда COPY — это встроенный инструмент PostgreSQL, разработанный специально для высокоэффективной загрузки данных. Она позволяет загружать данные напрямую в таблицу, минуя SQL-парсер, компилятор и планировщик запросов. Использование метода BeginBinaryImport из библиотеки Npgsql предоставляет следующие преимущества:

- бинарный формат передачи данных значительно уменьшает объём передаваемой информации;
- данные обрабатываются потоком без буферизации всего массива в памяти;
- минимизируются накладные расходы на протокол, поскольку взаимодействие с сервером происходит через один вызов;
- весь процесс выполняется на уровне внутреннего механизма СУБД, что исключает лишние слои абстракции.

Так как COPY — это инструмент, изначально предназначенный для bulk-вставок, он демонстрирует кратное превосходство над ORM и пакетными вставками. Его эффективность обусловлена архитектурными особенностями PostgreSQL и низкоуровневым взаимодействием с сервером базы данных.

Проведённое тестирование подтверждает эти теоретические предпосылки.

Экспериментальные данные и бенчмарки

Для оценки эффективности были протестированы все три подхода на выборках в 100 тыс., 10 млн и 100 млн строк. Замеры, результаты которых представлены в таблице 1, проводились в идентичных условиях с учётом времени выполнения и вычисления средней скорости вставки.

Таблица 1.

Результаты вставки больших объемов данных различными методами.

Метод вставки	**Объём данных**	**Время (сек.)**	**Скорость (стр./сек.)**
Entity Framework (ORM)	100 000	4626,60	21,61
EF Core Bulk Extensions	100 000	0,86	116 762,62
COPY (BeginBinaryImport)	100 000	0,26	388 800,98
EF Core Bulk Extensions	10 000 000	~57	~174 000
COPY (BeginBinaryImport)	10 000 000	~18	~565 000
EF Core Bulk Extensions	100 000 000	~2560	~39 000

COPY (BeginBinaryImport)	100 000 000	~437	~228 000

Результаты показали, что производительность ORM-решения крайне низка. Это объясняется тем, что каждый INSERT-запрос инициирует отдельную транзакцию и процесс компиляции SQL-команды. Даже при использовании EF Core Bulk Extensions достигается прирост за счёт группировки, однако метод остаётся ограниченным логикой ORM.

COPY, напротив, использует внутренние механизмы PostgreSQL, загружая данные напрямую через сокет соединения. Бинарный формат передачи исключает преобразования типов и парсинг, что даёт кратный прирост производительности. Таким образом, эффективность этого метода объясняется не только меньшим числом запросов, но и минимизацией вычислительных затрат.

Реализация на платформе C# с использованием Npgsql Разработка модуля миграции данных включала реализацию следующих компонентов:

- потоковое чтение из источника через IDataReader;
- преобразование и маппинг с учётом структуры целевой базы;
- использование BeginBinaryImport для записи данных в PostgreSQL;
- возможность параллельной загрузки для повышения пропускной способности.

Заключение

На основании проведенного исследования можно сделать вывод о целесообразности использования нативных bulk-операций в задачах миграции данных на C#. Механизм COPY, реализованный через библиотеку Npgsql, демонстрирует существенное преимущество по сравнению с альтернативными подходами. Использование метода BeginBinaryImport позволяет достичь высокой производительности, надежности и масштабируемости.

Результаты, полученные в ходе тестирования, подтверждают, что данный метод является оптимальным выбором для обработки больших объемов информации. Это особенно актуально в условиях ограниченного времени и ресурсов при интеграции или модернизации информационных систем.

Список литературы

1. Melton, J., Simon, A. SQL:1999: Understanding Relational Language Components. — 1-е изд. — San Francisco: Morgan Kaufmann, 2001. — 550 с.
2. PostgreSQL Global Development Group. PostgreSQL documentation: COPY [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/docs/current/sql-copy.html (дата обращения: 28.03.2025).
3. Freeman, A., Sanderson, A. Pro ASP.NET Core 6. — Berkeley: Apress, 2022. — 1145 с.
4. Richter, J. CLR via C#. — 4-е изд. — Redmond: Microsoft Press, 2012. — 826 с.
5. Юнусов А.Р., Кремлева Э.Ш. Дискретная математика в информационных системах и технологиях // Тенденции развития науки и образования. — 2023. — №. 107-3. — С. 162-166.
6. Gravell, M. Dapper Documentation [Электронный ресурс]. — Режим доступа: https://github.com/DapperLib/Dapper (дата обращения: 28.03.2025).
7. Microsoft. Entity Framework Core documentation [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/en-us/ef/core/ (дата обращения: 28.03.2025).
8. Npgsql Development Team. Npgsql Documentation [Электронный ресурс]. — Режим доступа: https://www.npgsql.org/doc/index.html (дата обращения: 28.03.2025).
9. ISO/IEC 9075-1:2016. Information technology — Database languages — SQL — Part 1: Framework. — Geneva: International Organization for Standardization, 2016.