sibedge

Не EF Core единым: альтернативная ORM LINQ to DB изменение данных

Алексей Фадеев, старший разработчик .NET

О себе

Алексей Фадеев, старший разработчик .NET

Компания sibedge, г.Томск

Сфера деятельности

Backend-разработка
Работа с СУБД, оптимизация
Флагманская СУБД: PostgreSQL,

Контакты

Email: fadeevas@sibedge.com
https://vk.com/fadeev

LINQ to DB

Год назад здесь
https://dotnext.ru/archive/2023/talks/0dd8c50ae2bf4064ab9937256b60dbf8

Альтернативная ORM для .NET

Богатый функционал «из коробки»

Возможность легко писать расширения

Высокая производительность

Всё это про чтение данных (SELECT)

LINQ to DB

Изменение данных

- Сложные операции (обычно недоступно в ORM)
- Массовая вставка (производительность)

linq2db.EntityFrameworkCore

Опыт внедрения на крупном проекте

EF Core, NHibernate – расширяемость?

Unit of Work

Логическая транзакция

- Накопление информации об изменениях
- Синхронизация изменения с БД (SaveChanges)

LINQ to DB не реализует Unit of Work!

Unit of Work

Идеальный код бэкенда

- Простой
- Приближенный к б/л

Идеальное взаимодействие с БД

- Оптимальный
- Приближенный к логике БД

Unit of Work

Можно не использовать в EF Core (начиная с 7)

```
await dbContext.Items
    .Where(x => x.Id == id)
    .ExecuteUpdateAsync(x => x
        .SetProperty(u => u.Value, 100));
```

LINQ to DB

```
await dbContext.Items
    .Where(x => x.Id == id)
    .UpdateAsync(x => new Item { Value = 100 });
```

LINQ to DB

Поднять зарплату всем!

```
await dbContext.Employees
    .UpdateAsync(x => new Employee { Salary = x.Salary + 5000 });
```

Без загрузки данных на бэкенд

```
UPDATE employee SET salary = salary + 5000
```

LINQ to DB

Изменение данных

- Сложные операции (обычно недоступно в ORM)
- Массовая вставка (производительность)

linq2db.EntityFrameworkCore

Опыт внедрения на крупном проекте

EF Core, NHibernate – расширяемость?

Update + Join

company

id	name	director_id
1	Рога и копыта	20
2	NULL	35

individual

id	name	surname	second_name
1	Иванов	Иван	Иванович
2	Козлов	Семён	Степанович

Если имя не задано: задать по типу «ИП Иванов Иван Иванович»

Update + Join*

Таблица, которую обновляем, первая в JOIN!

```
var updatable =
    (from company in context.Companies
     join individual in context.Individual
         on company.Id equals individual.DirectorId
     select new { company, individual })
         .AsUpdatable();

updatable = updatable.Set(pair => pair.company.Name,
     pair => "ИП " + pair.individual.Surname + " " + pair.individual.Name
         + " " + pair.individual.SecondName);

await updatable.UpdateAsync();
```

* У EF Core документация намного лучше :/

Update + Join

```
UPDATE
        company
SET
        name = 'ИП ' || individual.surname || ' ' || individual.name
            || ' ' || individual.second_name
FROM
        individual
WHERE
        company.director_id = individual.id
```

В Oracle не работает

Insert From Select

NHibernate умеет, EF Core - нет

company

id	name	director_id
1	Рога и копыта	20
2	NULL	35

individual

id	name	surname	second_name
1	Иванов	Иван	Иванович
2	Козлов	Семён	Степанович

Каждому физлицу – по ИП! (если ещё нет)

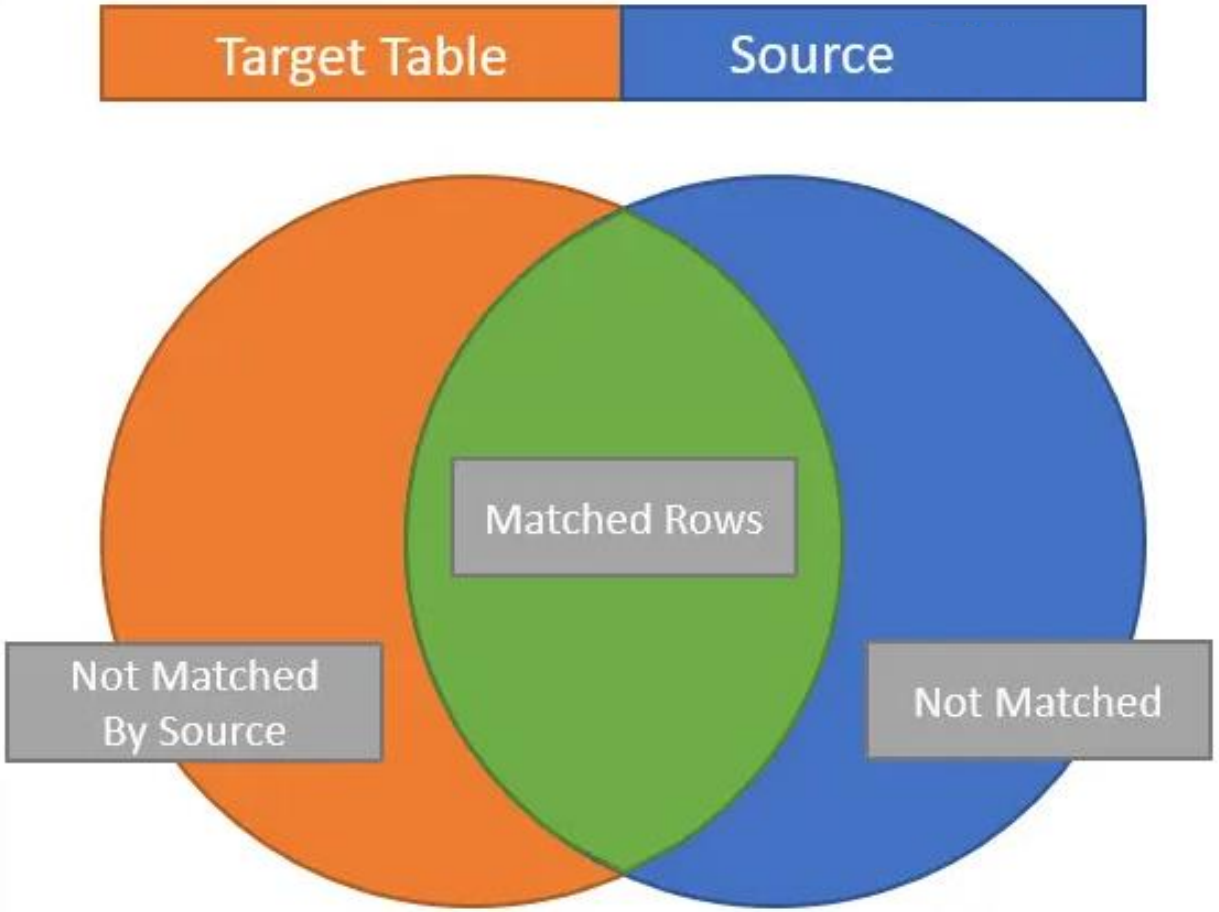
Insert From Select

```
var pairs =
    from c in context.Companies
    join i in context.Individuals
        on c.Id equals i.DirectorId
    select c;

var freeIndividuals = context.Individuals
    .Where(i => !pairs.Any(x => x.DirectorId == i.id));

await freeIndividuals
    .Into(context.Companies)
    .Value(x => x.Name, i => "ИП " + i.Surname + " " + i.Name + " " + i.SecondName)
    .Value(x => x.DirectorId, i => i.Id)
    .InsertAsync();
```

Insert From Select

```
INSERT INTO company (name, director_id)
SELECT 'ИП ' || i.surname || ' ' || i.name || ' ' || i.second_name,
       i.id
FROM   individual i
WHERE  NOT EXISTS(SELECT * FROM company x
                  INNER JOIN individual i ON x.director_id = i.id)
```

Merge


Target Table
Source
Matched Rows
Not Matched By Source
Not Matched

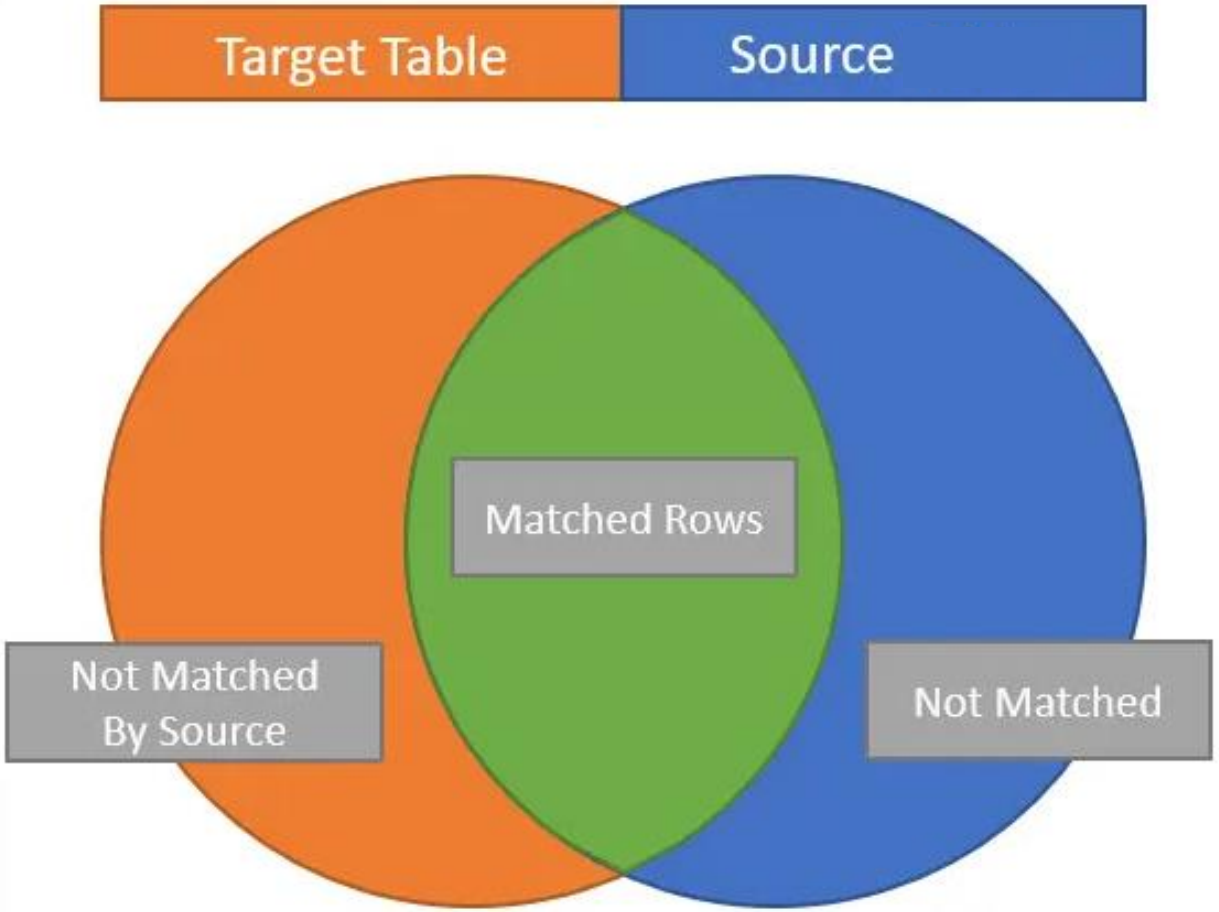
Merge

```
await context.Companies.Merge()
    .Using(context.Individuals)
    .On((c, i) => c.DirectorId == i.Id)
    .UpdateWhenMatched((c, i) => new Company
    {
        Name = "ИП " + i.Surname + " " + i.Name + " " + i.SecondName,
    })
    .InsertWhenNotMatched(i => new Company
    {
        Name = "ИП " + i.Surname + " " + i.Name + " " + i.SecondName,
        DirectorId = i.Id
    })
    .MergeAsync();
```

Merge

```
MERGE INTO company c
USING ( SELECT  i1.id as "Id", i1.name as "Name", i1.surname as "Surname",
                i1.second_name as "SecondName"
        FROM    individual i1) "Source"
( "Name", "Surname", "SecondName")
ON ("Target".director_id = "Source"."Id")

WHEN MATCHED THEN
UPDATE
SET     name = "ИП " || "Source"."Surname" || " " || "Source"."Name" || " "
    || "Source"."SecondName"

WHEN NOT MATCHED THEN
INSERT (name, director_id)
VALUES ("ИП " || "Source"."Surname" || " " || "Source"."Name" || " "
    || "Source"."SecondName", "Source"."Id")
```

Merge



Postgres: 15+

Merge на основе JSON

Данные приходят в JSON

Merge в таблицу БД

1. Передать JSON параметром
2. Нужен перечень колонок (поля JSON)
3. Развернуть элементы массива JSON в строки таблицы
4. В коде: → IQueryable
5. Merge

Merge на основе JSON

```
public static IQueryable<int> StubQueryable(this IDataContext dc)
{
    return dc.FromSqlScalar<int>(FormattableStringFactory.Create("SELECT 1"));
}


[ExpressionMethod(nameof(JsonbArrayToAnythingImpl))]
public static IQueryable<T> JsonbArrayToAnyItems<T>(this IDataContext dc, string jsonb, string columns) =>
    throw new LinqException("It is server-side method.");

private const string JsonbArrayToNamedItemsCommon1 =
 "SELECT x.* FROM (SELECT jsonb_array_elements({0}::jsonb) j) s1 INNER JOIN LATERAL jsonb_to_record(s1.j) AS x(";

private const string JsonbArrayToNamedItemsCommon2 = ") ON TRUE";

private static Expression<Func<IDataContext, string, string, IQueryable<T>>> JsonbArrayToAnythingImpl<T>() =>
    (dc, jsonb, columns) => dc.FromSql<T>(FormattableStringFactory
        .Create($"{JsonbArrayToNamedItemsCommon1} {columns} {JsonbArrayToNamedItemsCommon2}", jsonb));

public static string GetColumnsString(IList<(string, string)> columns)
{
    return string.Join(", ", columns.Select(c => $"\"{c.Item1}\" {c.Item2}"));
}
```

Merge на основе JSON

```
var qJsonb = context.StubQueryable().Select(x => jsonb);

var columns = new List<(string, string)> { (nameof(NamedItem.Id), "integer"),
    nameof(NamedItem.Name), "text") };
var columnsStr = PgExt.GetColumnsString(columns);

var itemsQuery =
    (from j in qJsonb
    from ja in context.JsonbArrayToAnyItems<NamedItem>(j, columnsStr)
    select new { ja.Id, ja.Name });

await context.TestItems.Merge()
    .Using(itemsQuery)
    .On((t, j) => t.Id == j.Id)
    .UpdateWhenMatched((t, j) => new TestItem { Name = j.Name, Number = -t.Number })
    .InsertWhenNotMatched(j => new TestItem { Name = j.Name, Number = 0, Id = j.Id })
    .MergeAsync();
```

Немного про транзакции

```
context.BeginTransaction();

var c1 = context.Items.CountAsync();

// Здесь из таблицы items удалили несколько записей в другой транзакции
await Task.Delay(100);

var c2 = context.Items.CountAsync();


      c2 == c1 // ?
```

Уровни изоляции транзакций

```
context.BeginTransaction();

var c1 = context.Items.CountAsync();

await Task.Delay(100);

var c2 = context.Items.CountAsync();
```

Read Committed («чистое» чтение)

```
c1 == c2 // false
```

Repeatable Read (повторяемое чтение)

```
c1 == c2 // true
```

Если можно в одно действие...

Задача с раздачей призов

Взять первый свободный приз (порядок важен)

Назначить пользователю

Добавить запись в таблицу уведомлений

Задача с раздачей призов

Read Committed («чистое» чтение)

Две транзакции одновременно выбирают один приз

Транзакция 1 завершается

Приз – пользователю 1, уведомление

Транзакция 2 завершается

Приз – пользователю 2 (затёрли изменение трн.1), уведомление

Результат: пользователь 1 остаётся без приза, но получает уведомление

Задача с раздачей призов

Repeatable Read (повторяемое чтение)

Две транзакции одновременно выбирают один приз

Транзакция 1 завершается

- Приз – пользователю 1, уведомление

Транзакция 2 падает с ошибкой

Нужно обработать ошибку и повторить попытку

Select For Update

Блокировка на уровне строк

Режимы:

- По умолчанию (ожидание завершения других транзакций)
- NOWAIT (ошибка, если нельзя заблокировать немедленно)
- SKIP LOCKED (пропустить заблокированные)

Select For Update Skip Locked

Пропуск заблокированных строк, как будто их нет в таблице

Без ожидающих блокировок

Без падений с ошибкой

В LINQ to DB – можно!

- Сделано через механизм hints

Select For Update Skip Locked

```
await context.Prizes
    .SubQueryHint(PostgreSQLHints.ForUpdate + " " + PostgreSQLHints.SkipLocked)
    .ToListAsync();
```

```
SELECT * FROM prize p1
  FOR UPDATE SKIP LOCKED
```

LINQ to DB: чего нет в EF

UPDATE + JOIN

INSERT FROM SELECT

Поддержка оператора MERGE

SELECT FOR UPDATE

LINQ to DB

Изменение данных

- Сложные операции (обычно недоступно в ORM)
- Массовая вставка (производительность)

linq2db.EntityFrameworkCore

Опыт внедрения на крупном проекте

EF Core, NHibernate – расширяемость?

Массовая вставка (Bulk Copy)

Поддерживаются методы:

- RowByRow
- MultipleRows
- ProviderSpecific

https://linq2db.github.io/articles/sql/Bulk-Copy.html

Bulk Copy: ProviderSpecific

MS SQL Server: BULK INSERT

- Не работает внутри явной транзакции
- Игнорируются триггеры

Oracle: BULK INSERT

- Игнорируются Check Constraints
- Игнорируются триггеры

Postgres: COPY

- Всё хорошо ☺

Массовая вставка (Postgres)

```
var opt = new BulkCopyOptions
    { BulkCopyType = BulkCopyType.ProviderSpecific };

await context.BulkCopyAsync(opt, items);
```

Массовая вставка, производительность

EF Core (AddRange)

```
37313 мс
37147 мс
```

LINQ to DB

```
2234 мс
2226 мс
```

x16.7 vs EF Core

Массовое обновление

На основе JSON – вариант!

Через временную таблицу

Массовое обновление

```
var opt = new BulkCopyOptions
    { BulkCopyType = BulkCopyType.ProviderSpecific };

var tempTable = await context.CreateTempTableAsync("temp_items", items, opt);

var updatable = (from item in context.TestItems
        join tempItem in tempTable on item.Id equals tempItem.Id
        select new { item, tempItem })
    .AsUpdatable();

updatable = updatable.Set(pair => pair.item.Name, pair => pair.tempItem.Name);

await updatable.UpdateAsync();
```

Производительность

Вставка

2226 мс

Обновление

4815 мс

Массовые операции

Вставка: BulkCopy

- Важно указать метод
- Зависит от СУБД
- Postgres: ProviderSpecific (очень быстро!)

Обновление:

- Создаём временную таблицу
- Вставка через BulkCopy
- UPDATE + JOIN / MERGE

LINQ to DB

Изменение данных

- Сложные операции (обычно недоступно в ORM)
- Массовая вставка (производительность)

linq2db.EntityFrameworkCore

Опыт внедрения на крупном проекте

EF Core, NHibernate – расширяемость?

Не готовы отказаться от EF Core?

Страшно / рискованно

Существующий проект (дорого)

Зависимости

Code First

…

linq2db.EntityFrameworkCore

Позволяет выполнять запросы LINQ to DB

Классы DbContext и Entities – для EF Core

Очень просто!

linq2db.EntityFrameworkCore

Подключить

```
LinqToDBForEFTools.Initialize();
```

Включить логи запросов:

```
var sp = services.BuildServiceProvider();
var logger = sp.GetService<ILogger<MyDbContext>>();

DataConnection.TurnTraceSwitchOn();

DataConnection.WriteTraceLine = (s, _, _) =>
{
    logger.LogDebug(s);
};
```

linq2db.EntityFrameworkCore


```
var query = dbContext.Transactions
    .Select(x => new
    {
        x.Id,
        x.TransactionDate,
        x.TransactionNumber,
        Json = LinqToDbExtensions.CreateJson(x.Id,
            x.TransactionDateTime.FromTimeZone(MOSCOW_TIMEZONE))
    })
    .ToLinqToDB();
```

linq2db.EntityFrameworkCore

```
var linq2dbContext = efContext.CreateLinqToDBConnection();

var cte = linq2dbContext.GetCte<PlaceHierarchy>(hierarchy => (
        from t in dbContext.PlaceGroupTrees
        join g in dbContext.PassengerPlaceGroups on t.PgId equals g.Id
        join pg in dbContext.PlacesInGroups on g.Id equals pg.PgId
        where pg.PlaceId == id
        select new PlaceHierarchy
            { CurrentId = g.Id, ParentId = t.ParentPgId })
    .Concat(from t in dbContext.PlaceGroupTrees
        join h in hierarchy on t.PgId equals h.ParentId
        select new PlaceHierarchy
            { CurrentId = t.PgId, ParentId = t.ParentPgId }));
```

linq2db.EntityFrameworkCore

EF Core + LINQ to DB – идеальное решение?

Смотрите далее!

LINQ to DB

Изменение данных

- Сложные операции (обычно недоступно в ORM)
- Массовая вставка (производительность)

linq2db.EntityFrameworkCore

Опыт внедрения на крупном проекте

EF Core, NHibernate – расширяемость?

Коротко о проекте

Билетная система пассажирского транспорта

Oracle (поэтапно переводим на Postgres)

Более 1000 таблиц, много секционированных

NHibernate, EF Core

Внедрили linq2db.EntityFrameworkCore в конце 2023

Плюсы внедрения

Вынос кода из хранимок (пакетов)

- Рекурсивные запросы
- Вставка из Json
- Merge

Проще переходить NHibernate → EF Core

- Insert From Select

Использование расширений

Конвертация даты/времени на стороне БД

```
[Sql.Expression("FROM_TZ(CAST({0} AS TIMESTAMP), {1})", ServerSideOnly = true)]
public static DateTime FromTimeZone(this DateTime dt, string timeZone)
    => throw new NotImplementedException();
```

```
var query = dbContext.Transactions
    .Select(x => new
    {
        x.Id,
        x.TransactionNumber,
        DateTime = x.TransactionDateTime.FromTimeZone(MOSCOW_TIMEZONE))
    })
    .ToLinqToDB();
```

Ограничения Oracle

Бедный SQL

Ограничение длины строки, CLOB

Нет табличных функций

Ждём перехода на Postgres…

Что не получилось

Вставка до 31 записей

Критично время выполнения веб-запроса

Вставка как таковая:

- LINQ to DB быстрее на 15-20 мс
- Общее время выполнения не изменилось

Что не получилось

EF Core

- Инициализация сеанса EF Core

linq2db.EntityFrameworkCore

- Инициализация сеанса EF Core
- Инициализация сеанса LINQ to DB

Внедрение linq2db.EntityFrameworkCore

Однозначно оправдало себя

Проще переходить NHibernate → EF Core

Ограничения Oracle (ждём Postgres)

Небольшой оверхед на инициализацию

LINQ to DB

Изменение данных

- Сложные операции (обычно недоступно в ORM)
- Массовая вставка (производительность)

linq2db.EntityFrameworkCore

Опыт внедрения на крупном проекте

EF Core, NHibernate – расширяемость?

Расширяем NHibernate

```
RegisterFunction("AddDays", new SQLFunctionTemplate(NHibernateUtil.DateTime,
    "?1 + ?2"));
```

Расширяем NHibernate

```
RegisterFunction("AddDays", new SQLFunctionTemplate(NHibernateUtil.DateTime,
    "?1 + ?2"));
```

```
public class AddDaysGenerator : BaseHqlGeneratorForMethod
{
    public AddDaysGenerator()
    {
        SupportedMethods = new[] {
            ReflectHelper.GetMethodDefinition<DateTime?>(d => d.Value.AddDays(0))
        };
    }

    public override HqlTreeNode BuildHql(MethodInfo method, Expression targetObject,
        ReadOnlyCollection<Expression> arguments, HqlTreeBuilder treeBuilder,
        IHqlExpressionVisitor visitor)
    {
        return treeBuilder.MethodCall("AddDays", visitor.Visit(targetObject).AsExpression(),
            visitor.Visit(arguments[0]).AsExpression());
    }
}
```

Библиотека efcore.pg

Операции с массивами и JSON

Встроенный полнотекстовый поиск

Операторы Postgres

`@>` `-|-` `?&` `@@` и др.

Библиотека efcore.pg

Операции с массивами и JSON

Встроенный полнотекстовый поиск

Операторы Postgres

`@>` `-|-` `?&` `@@` и др.

Это и есть Npgsql.EntityFrameworkCore.PostgreSQL

efcore.pg vs LINQ to DB

efcore.pg – функционал ограничен

LINQ to DB – изменение данных, рекурсия, табличные функции…

efcore.pg – следите за обновлениями

Расширение EF Core

Сделать fork от efcore.pg

Реализовать функционал

…

Profit!

Расширение EF Core

Сделать fork от efcore.pg

Реализовать функционал

…

~~Profit!~~ Pull request

Расширяемость ORM

EF Core: совсем тяжело

NHibernate: так себе

LINQ to DB: просто и удобно

Вставка 31 строки: решение

Библиотека Npgsql.Bulk

Автор: Антон Шкуратов (neisbut), г.Томск

Максимально простая

Нет затрат на «прогрев»

Ускорение получили! (пока тестовая среда)

Есть ещё интересный функционал

https://github.com/neisbut/Npgsql.Bulk

Итоги: возможности LINQ to DB

Изменение данных, сложные операции

- UPDATE + JOIN
- INSERT FROM SELECT
- Поддержка оператора MERGE
- SELECT FOR UPDATE

Массовые операции (высокоскоростные)

- Вставка: BulkCopy
- Обновление: через временную таблицу

Итоги: LINQ to DB и другие ORM

linq2db.EntityFrameworkCore

- EF Core и LINQ to DB вместе
- Простое подключение и использование
- Проверено на реальном проекте
- Есть ограничения

Расширяемость разных ORM

- Так или иначе, существует в EF Core и NHibernate
- LINQ to DB – наиболее просто и удобно
- efcore.pg (Npgsql.EntityFrameworkCore.PostgreSQL) – функционал расширяется


КОНЕЦ

Ссылки

LINQ to DB

https://github.com/linq2db/linq2db

https://github.com/linq2db/linq2db.EntityFrameworkCore

https://linq2db.github.io/api/LinqToDB.DataProvider.PostgreSQL.PostgreSQLExtensions.html

https://www.nuget.org/packages/linq2db

Документация PostgreSQL на русском

https://postgrespro.ru/docs

Npgsql.EntityFrameworkCore.PostgreSQL

https://github.com/npgsql/efcore.pg

Мои контакты

fadeevas@sibedge.com

https://vk.com/fadeev

Библиотека Npgsql.Bulk

https://github.com/neisbut/Npgsql.Bulk

P.S. Postgres или Postgre?

The official name of the project is : **PostgreSQL**

The first letter and the last 3 letters (SQL) should be capitalized, and all letters should be in the same colour, to avoid accidentally emphasising an incorrect name, such as "Postgre".

Postgres is also accepted as an alternative name.

All other names are incorrect, especially *Postgre*

https://wiki.postgresql.org/wiki/Identity_Guidelines