"Почему же мой запрос на Entity Framework 6 тормозит?" Часть 9000

Алексей Капустяненко

Стек на проекте

- ASP.NET Web API
- EntityFramework 6
- AutoMapper
- MS SQL + в дальнейшем Postgres

С чего все началось

50 строк C#

3000+ строк SQL

Live Demo 1

Суть проблемы

- Запрос большой и страшный, но выполняется он быстро
- Значит, EF делает "что-то" помимо запроса, довольно продолжительное по времени

Начало расследования

- ~~Переписать запрос~~
- ~~AsNoTracking()~~
- ~~Догружать связанные сущности отдельными запросами~~
- ~~Compiled Queries~~
- ~~Перейти на EF Core~~

След найден

- Performance Considerations (Entity Framework)

- Performance considerations for EF 4, 5, and 6

Performance Considerations (Entity Framework)

https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/ef/performance-considerations

Performance Considerations (Entity Framework)

30.03.2017 • Время чтения: 12 мин • 👤 🐾 👤

This topic describes performance characteristics of the ADO.NET Entity Framework and provides some considerations to help improve the performance of Entity Framework applications.

Stages of Query Execution

In order to better understand the performance of queries in the Entity Framework, it is helpful to understand the operations that occur when a query executes against a conceptual model and returns data as objects. The following table describes this series of operations.

Operation	Relative Cost	Frequency	Comments
Loading metadata	Moderate	Once in each application domain.	Model and mapping metadata used by the Entity Framework is loaded into a MetadataWorkspace. This metadata is cached globally and is available to other instances of ObjectContext in the same application domain.
Opening the database connection	Moderate[1]	As needed.	Because an open connection to the database consumes a valuable resource, the Entity Framework opens and closes the database connection only as needed. You can also explicitly open the connection. For more information, see Managing Connections and Transactions.
Generating views	High	Once in each application domain. (Can	Before the Entity Framework can execute a query against a conceptual model or save changes to the data source, it must generate a set of local query views to access the database. Because of the high cost of generating these views, you can pre-generate

Этапы выполнения запросов

```
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│    Загрузка     │ ──▶ │    Открытие     │ ──▶ │    Создание     │
│   метаданных    │     │ подключения базы│     │ представлений   │
│                 │     │     данных      │     │                 │
└─────────────────┘     └─────────────────┘     └─────────────────┘
                                                          │
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│   Загрузка и    │ ◀── │    Подготовка   │ ◀── │ Подготовка      │
│ проверка типов  │     │  материализатора│     │ запроса         │
│                 │     │                 │     │                 │
└─────────────────┘     └─────────────────┘     └─────────────────┘
        │
┌─────────────────┐     ┌─────────────────┐
│  Отслеживание   │ ──▶ │   Выполнение    │
│                 │     │    запроса      │
│                 │     │                 │
└─────────────────┘     └─────────────────┘
```

Performance considerations for EF 4, 5, and 6

https://docs.microsoft.com/ru-ru/ef/ef6/fundamentals/performance/perf-whitepaper
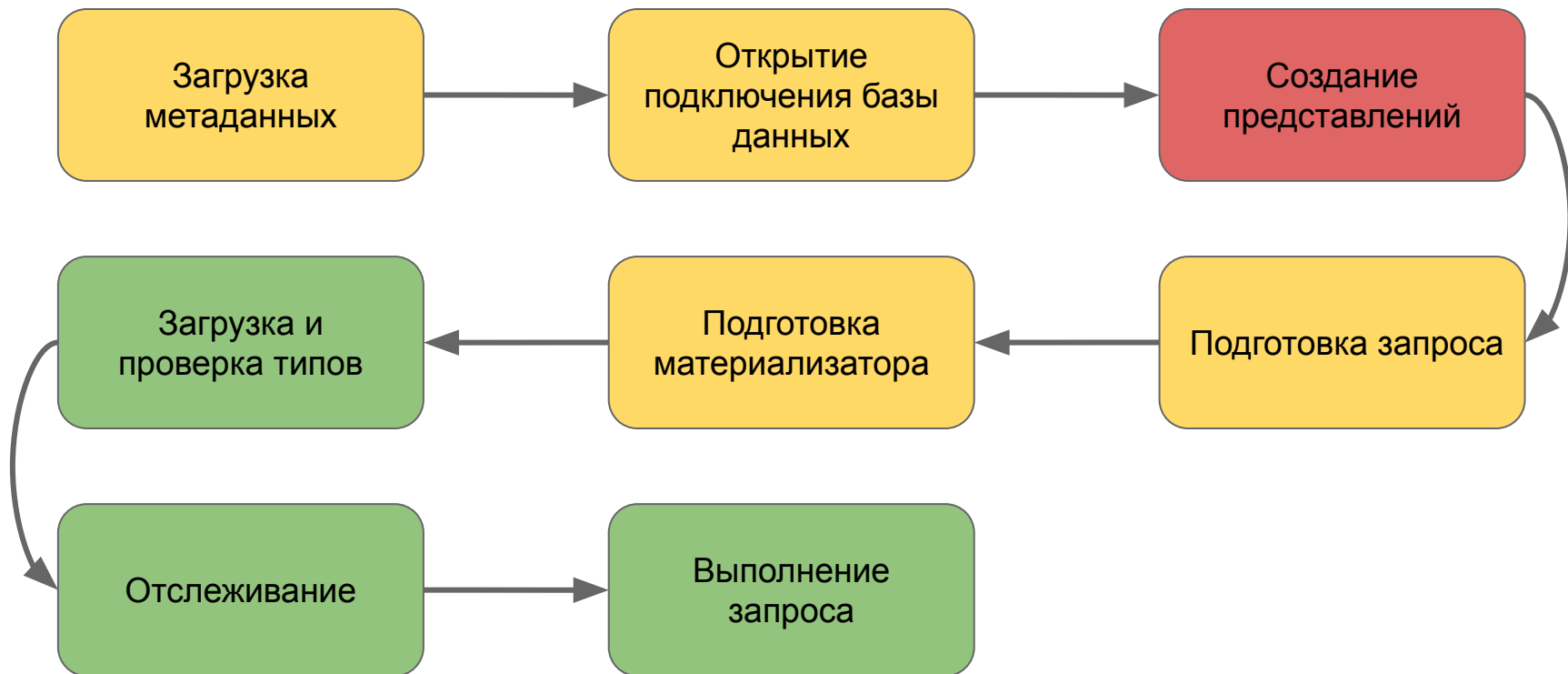
First Query Execution – cold query

Code User Writes	Action	EF4 Performance Impact	EF5 Performance Impact	EF6 Performance Impact
using(var db = new MyContext()) {	Context creation	Medium	Medium	Low
var q1 = from c in db.Customers where c.Id == id1 select c;	Query expression creation	Low	Low	Low
var c1 = q1.First();	LINQ query execution	- Metadata loading: High but cached - View generation: Potentially very high but cached - Parameter evaluation: Medium - Query translation: Medium - Materializer generation: Medium but cached - Database query execution: Potentially high + Connection.Open + Command.ExecuteReader + DataReader.Read Object materialization: Medium - Identity lookup: Medium	- Metadata loading: High but cached - View generation: Potentially very high but cached - Parameter evaluation: Low - Query translation: Medium but cached - Materializer generation: Medium but cached - Database query execution: Potentially high (Better queries in some situations) + Connection.Open + Command.ExecuteReader + DataReader.Read Object materialization: Medium - Identity lookup: Medium	- Metadata loading: High but cached - View generation: Medium but cached - Parameter evaluation: Low - Query translation: Medium but cached - Materializer generation: Medium but cached - Database query execution: Potentially high (Better queries in some situations) + Connection.Open + Command.ExecuteReader + DataReader.Read Object materialization: Medium (Faster than EF5) - Identity lookup: Medium

Performance considerations for EF 4, 5, and 6

https://docs.microsoft.com/ru-ru/ef/ef6/fundamentals/performance/perf-whitepaper

Second Query Execution – warm query

| Code User Writes	Action	EF4 Performance Impact	EF5 Performance Impact	EF6 Performance Impact
`using(var db = new MyContext()) {`	Context creation	Medium	Medium	Low
`var q1 = from c in db.Customers where c.Id == id1 select c;`	Query expression creation	Low	Low	Low
`var c1 = q1.First();`	LINQ query execution	- Metadata ~~loading~~ lookup: ~~High but cached~~ Low - View ~~generation~~ lookup: ~~Potentially very high but cached~~ Low - Parameter evaluation: Medium - Query ~~translation~~ lookup: Medium - Materializer ~~generation~~ lookup: ~~Medium but cached~~ Low - Database query execution: Potentially high + Connection.Open + Command.ExecuteReader + DataReader.Read Object materialization: Medium - Identity lookup:	- Metadata ~~loading~~ lookup: ~~High but cached~~ Low - View ~~generation~~ lookup: ~~Potentially very high but cached~~ Low - Parameter evaluation: Low - Query ~~translation~~ lookup: ~~Medium but cached~~ Low - Materializer ~~generation~~ lookup: ~~Medium but cached~~ Low - Database query execution: Potentially high (Better queries in some situations) + Connection.Open + Command.ExecuteReader + DataReader.Read Object materialization: Medium - Identity lookup: Medium	- Metadata ~~loading~~ lookup: ~~High but cached~~ Low - View ~~generation~~ lookup: ~~Medium but cached~~ Low - Parameter evaluation: Low - Query ~~translation~~ lookup: ~~Medium but cached~~ Low - Materializer ~~generation~~ lookup: ~~Medium but cached~~ Low - Database query execution: Potentially high (Better queries in some situations) + Connection.Open + Command.ExecuteReader + DataReader.Read Object materialization: Medium (Faster than EF5) - Identity lookup: Medium

Что из этого кешируется
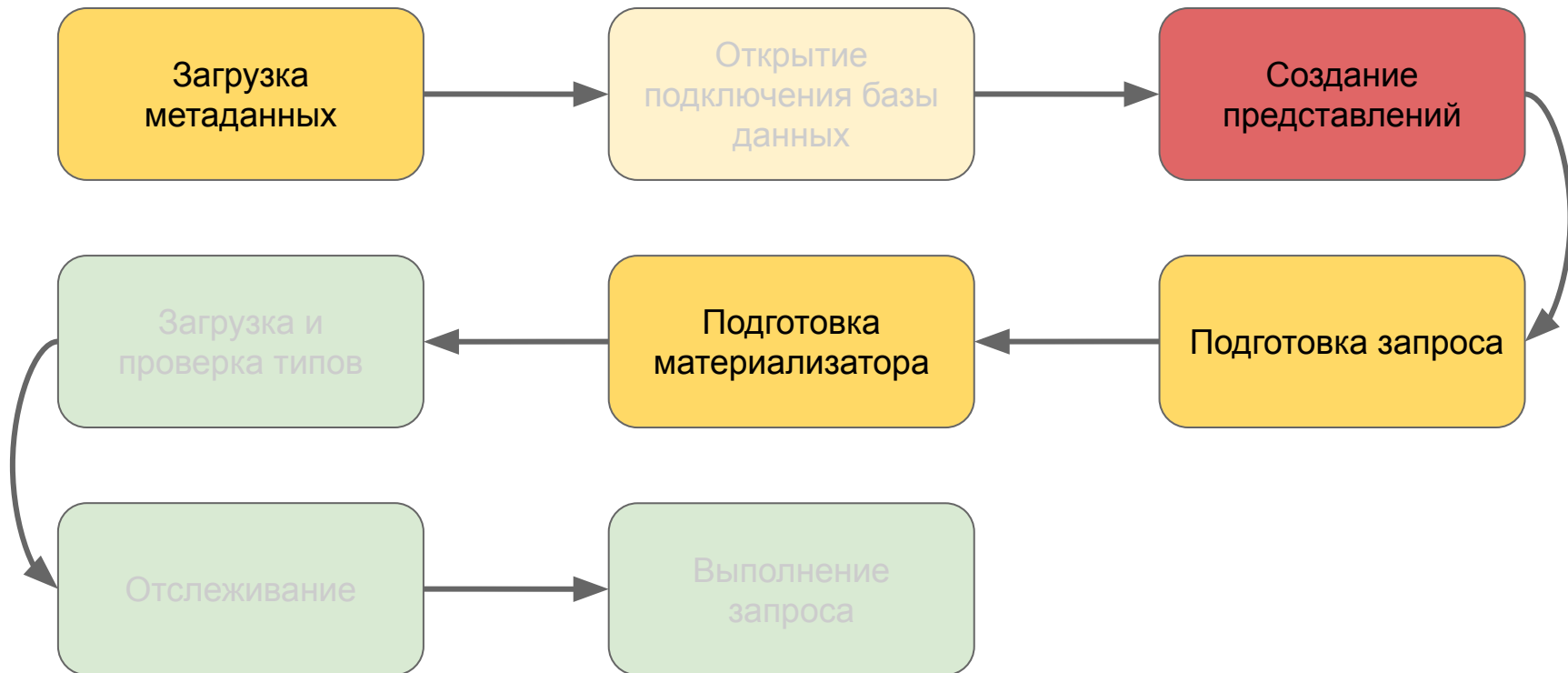
Подводные камни Entity Framework и производительность

https://habr.com/ru/post/269901/

Получается, что для оператора IN параметры не используются, а вместо этого подставляются сами значения. Такой запрос закешировать не получится, т.к. при использовании коллекции с другим содержимым текст запроса нужно будет перегенерировать. Это, кстати, бьет не только по производительности самого Entity Framework, но и по серверу базы данных, так как для любого нового списка значений в операторе IN сервер должен будет заново построить и закешировать план выполнения.

Если в коллекции, по которой делается Contains не ожидается большого числа элементов (скажем, не больше ста), проблему можно решить динамической генерацией условий, соединенных оператором OR. Это легко сделать, например, с помощью библиотеки LinqKit:

```
List<int> channels = new List<int> { 1, 5, 9 };

var channelsCondition = PredicateBuilder.False<Entity>();
channelsCondition = channels.Aggregate(channelsCondition,
    (current, value) => current.Or(e => e.Channel == value).Expand());

var query = dataContext.Entities
    .AsNoTracking()
    .Where(channelsCondition);
```

В итоге получаем уже параметризированный запрос:

```
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Name] AS [Name],
    [Extent1].[Channel] AS [Channel]
    FROM [dbo].[Entities] AS [Extent1]
    WHERE [Extent1].[Channel] IN (@p__linq__0,@p__linq__1,@p__linq__2)
```

Несмотря на то, что динамическое построение запроса выглядит дополнительной затратной работой, на практике на него уходит сравнительно немного процессорного времени. В одной из реальных задач

Contains

Live Demo 2

Any (All)

Live Demo 3

Константы

Live Demo 4

Результат

Live Demo 5

Заключение

- + Техника довольно действенная
- - Страдает код
- За абстракции приходится платить отказом от абстракций

Спасибо за внимание!!!

Гитхаб с демо:
https://github.com/alexeykap/Ef6Part9000PresentationMaterials

Мой telegram:
https://t.me/KapAleksey