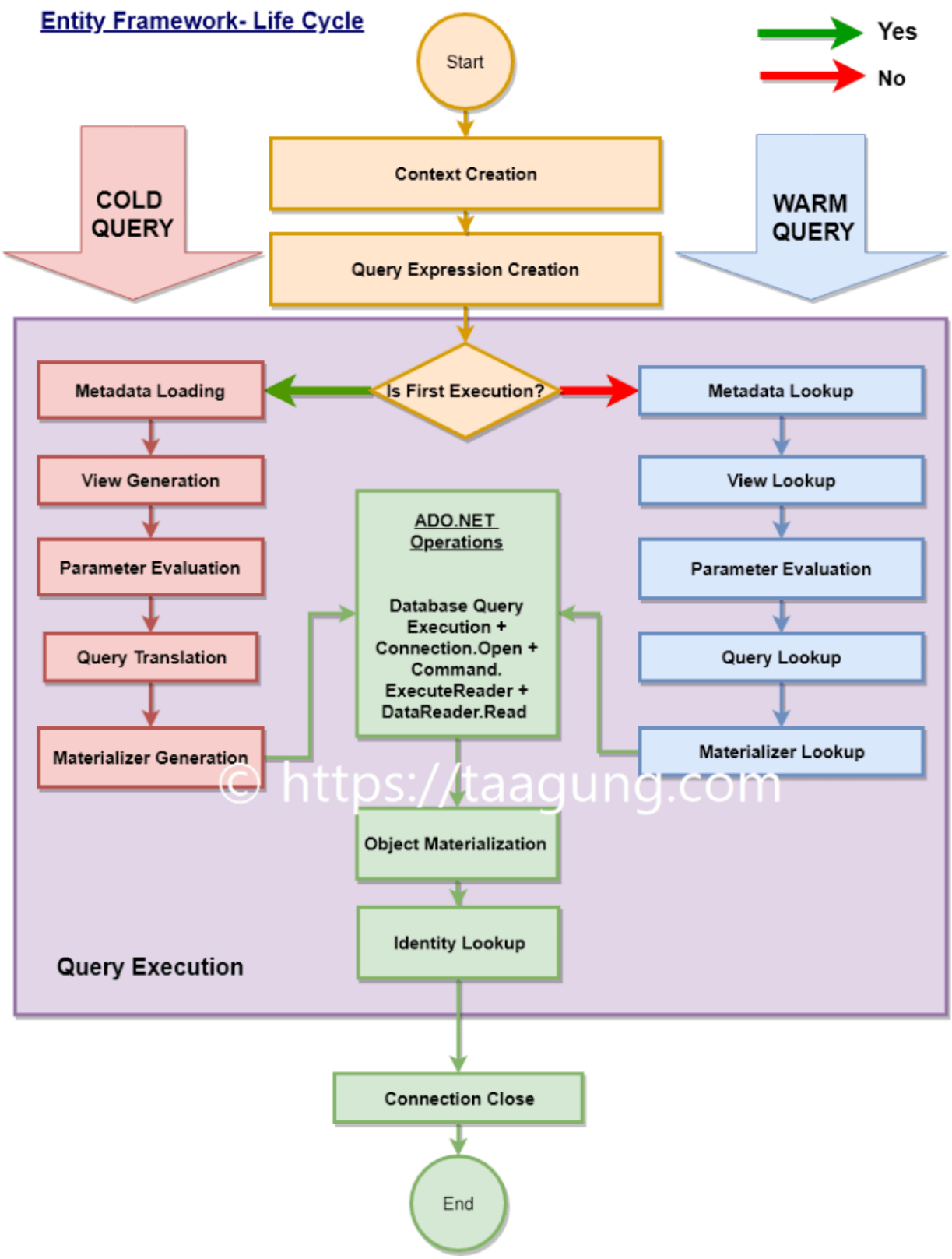
All Steps about Query Execution in Entity Framework

Entity Framework is the most popular ORM tool and it is developed by Microsoft. It facilitates the developer to work with Domain-specific objects and not to worry about underlying data in the database. It provides a high-level of abstraction for data access and results in fewer lines of code by a programmer.

Entity Framework is built on top of **ADO.NET** and due to all of its rich features and during troubleshooting, it has always made me inquisitive about what happens behind the scene. I couldn't find much information so relied totally on MSDN.

Based on this white paper, I created this diagram, which lists down all the steps involved and, in this article, I will be elaborating all of them.



Entity Framework Query Execution Life Cycle

After completing this article, you will have a good understanding of Entity Framework query processing. This requires basic knowledge of Entity Framework.

Based on this diagram, we can categorize all execution cycle in to 3 stages –

1. **Context Creation**
2. **Query Expression Creation**
3. **Query Execution**

Let us understand each of them and know them more.

Step # 1 - Context Creation

In this step, Context is created. Context is like session with database for all database operations (Create, Retrieve, Update and Delete). Context is responsible for following tasks –

1. Query Execution on the connection.
2. Tracking the changes made to the entity after data retrieval
3. Persisting object changes to the database.
4. Caching
5. The materialization of data to object/ entity

This is how we declare a context

```
public class BlogContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Article> Articles { get; set; }
}
```

And here we create a context in this way

```
public void SubmitArticle()
{
    using (var context = new BlogContext())
    {
        // Perform data access using the context
    }
}
```

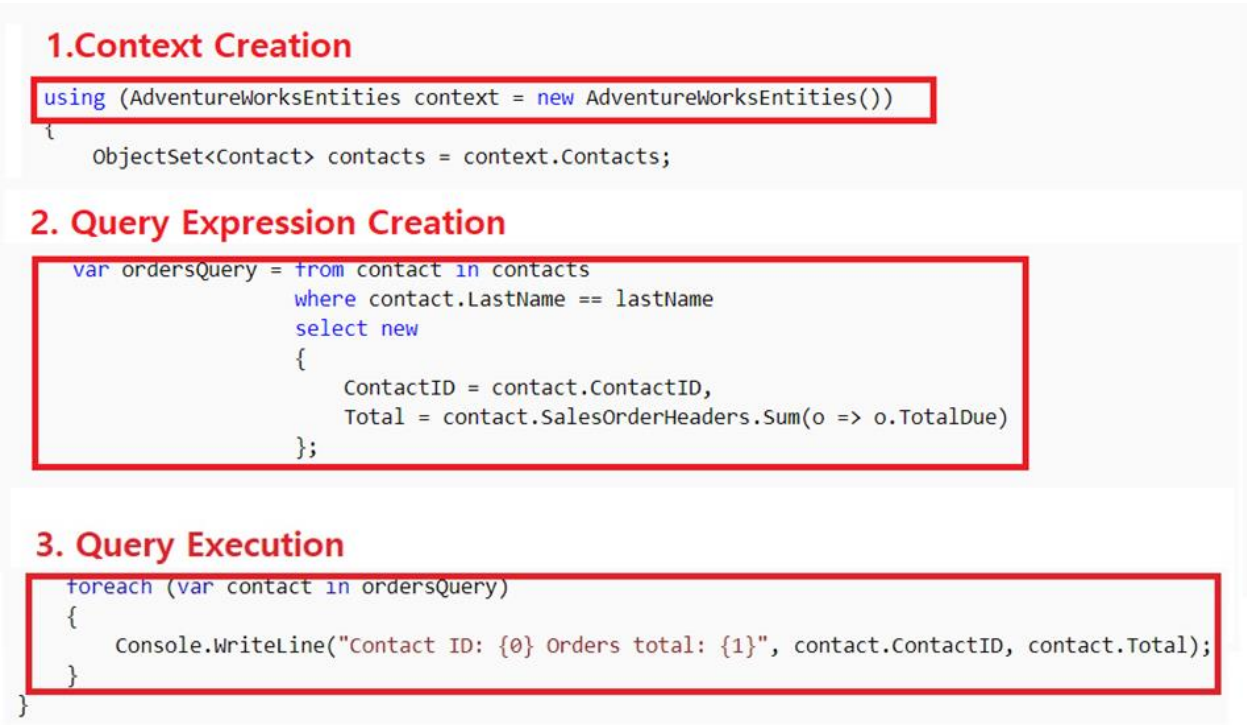
Step # 2 - Query Expression Creation

A Query is an expression which retrieves data from the data source. In the query, we specify what we need from the data source. It can be also used in sorting, grouping and massaging the data to get in needed shape.

Query expression can contain lambda expressions and anything allowed in the namespace System.Linq.Expressions. You can learn more about LINQ Queries here.

In this step, LINQ expressions are created and underlying query information is stored. Execution of LINQ expression doesn't happen in this stage and we will be covering that in more detail.

From this screen shot, we can see all the three stages which we code for.



3 major stages of LINQ Query Execution in Entity Framework

Step # 3 - Query Execution

Two more terms, in the diagram to make you curious are Cold and Warm Queries.

When A query is executed for the very first time on the data source, it does a lot of background tasks to load and validate the model. Such first queries are known and **Cold Queries** while subsequent executions are **Warm Queries**. In the diagram, *we see Cold Queries are loading and generating while Warm Queries are about looking up.*

Third stage LINQ Query Execution (or simply Query Execution) has many substages and we will be going through one by one in order as it is shown in Entity Framework Life Cycle diagram and understand more.

1. Metadata Loading/Lookup –

To understand this, first we will have to understand the models in Entity Framework (EF).

Storage Model – This is also known as a logical model. Logical/Storage Model defines entities and their relationship with other entities with foreign key constraints. Normally we write our queries and stored procedure to work with the logical model. In Entity Framework, Storage Model is defined by **Store Schema Definition Language (SSDL)** and has a file extension of .ssdl.

Conceptual Model - This can be viewed as a Domain Model for any application. It may or may not be the exact replica of our relational database. Designing of Conceptual model depends on the shape of data needed in our application. Entities and Relationship can be imagined as the object and association in the application.

Entity Framework (EF) uses **Conceptual Schema Data Language (CSDL)** to define Conceptual Model. So conceptual model facilitates user to write domain without considering storage model to ensure efficiency and maintainability.

Mapping Model – This is written **Mapping Specification Language (MSL)**, which is an XML based language to describe the relationship between Storage and Conceptual Model. At design time, Entity Framework stores the mapping information in the edmx file, which is converted to .msl file at build time which is eventually needed and used by EF at run time.

More details about these models are out of scope for this article but if needed you can refer this MSDN tutorial.

Now since we are familiar with models, let's understand the **Metadata Loading**.

In Entity Framework, Metadata code is responsible for mapping between different models. It has the responsibility of loading and parsing of models and to perform various mapping among them.

For Cold Queries, it does metadata loading as it creates for the first time for that conceptual model, While for Warm Queries, it looks up the metadata already loaded.

2. View Generation/Lookup –

Understanding of View Generation is dependent to Mapping views, so let us first get familiar with this.

There are two types of database operations, one is **query operation** and another is **update operation**. On this basis, we can have two mapping views

1. **Query Views –** Transformation from database schema to conceptual model.
2. **Update Views –** Transformation from conceptual model to the database schema.

These views are generated based on the mapping configuration in Mapping model. With this, *let us be clear that Mapping views are* **NOT** *database object* ***views***. They are auto-generated C#/VB classes. Later it is converted into data source specific queries.

Once they are created, they are validated against the model. This is a very costly process and hence they are cached at **the application domain level**. Multiple instances of the same Context in the same application domain, are reused from the cache.

So, for Cold Queries, they are generated while for Warm Queries they are reused from metadata cache.

3. Parameter Evaluation –

This step involves the parameter's evaluation, which is dynamically going to be part of our Query views or Update views. Since parameters are part of query expressions, they are supposed to follow the convention of data source, as they will be executed at the server. They need not be **CLR Compliant.** If any conflict between Server and Client, Server rules will overwrite the client configuration.

It is same for Cold and Warm Queries.

4. Query Translation/Lookup –

Both types of views are converted to CQTs (canonical query trees). Equivalent query is translated for execution against specific to data source.

For Cold Queries they are translated while Warm queries look for them to reuse.

5. Materializer Generation/ Lookup –

Now, CQTs are passed to Entity Framework Providers (EF Providers) and now data store (database) specific queries are ready for execution and if query plan is saved, they will be reused for subsequent/warm queries.

Parameter change in Where clause doesn't need a new query plan while changing the filter criteria leads to new query plan generation and caching (if enabled).

Query Plan caching is done in **MetadataWorkspace's ItemCollection**. This caching is done at the connection string level. A query is not executed on the database for any of the query (Cold and Warm) until -

- For Query View, Result is requested. e.g. loop iteration, ToList() etc.
- For update View, SaveChanges() is called

6. ADO.NET Operations

Now Some basic operations from ADO.NET are performed in the following sequence and they are common to Cold and Warm Queries.

- ADO.NET Command object created

- Connection opened
- Data is Read

7. Object Materialization

Now that query has been executed, results can be returned as

- Collection of type defined in conceptual model
- CLR type, supported by conceptual model
- Anonymous type

Process of conversion of the query result to CLR compliant type is known as **Object Materialization** and it is done by Entity Framework.

8. Identity Lookup

If the data coming from the database and already in the cache/state manager, have the same identity, their merging will be according to the **MergeOption** specified in the query.

When more than one tables are involved, the relationship among the tables is taken care of by the identity of respective tables.

9. Connection Close

When control moves out of using block, context is disposed and connection gets closed.

Disclaimer - This article is about Entity Framework NOT about Entity Framework Core as EF Core is complete rewrite so it follows different flow.

This article is originally published at taagung.

References –

- https://docs.microsoft.com/en-us/ef/ef6/fundamentals/working-with-dbcontext
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/language-reference/queries-in-linq-to-entities
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/language-reference/query-results
- https://docs.microsoft.com/en-us/ef/ef6/fundamentals/performance/perf-whitepaper
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/language-reference/query-execution

- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/language-reference/expressions-in-linq-to-entities-queries