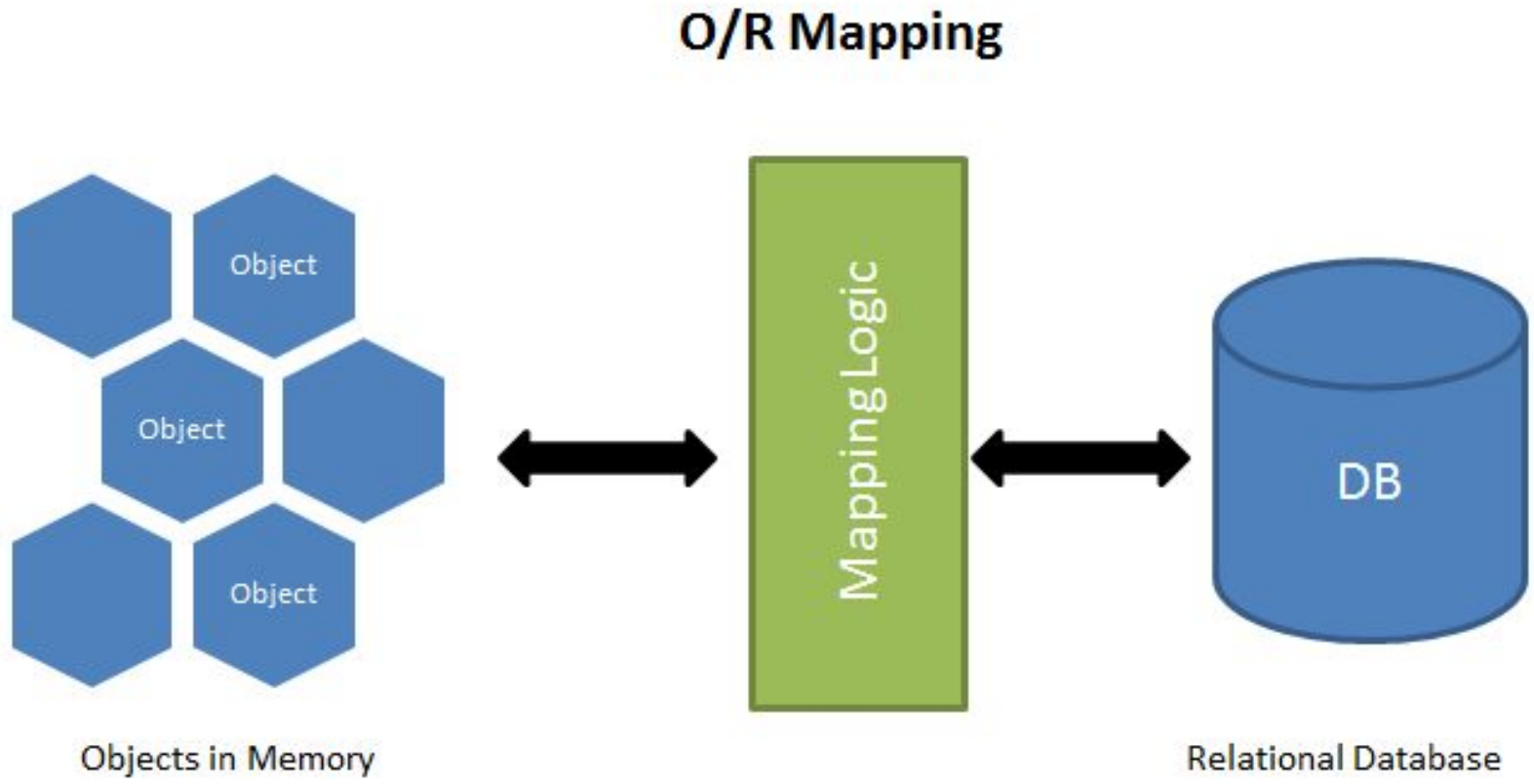
ORM technologies. Entity Framework. Dapper.

Speaker:

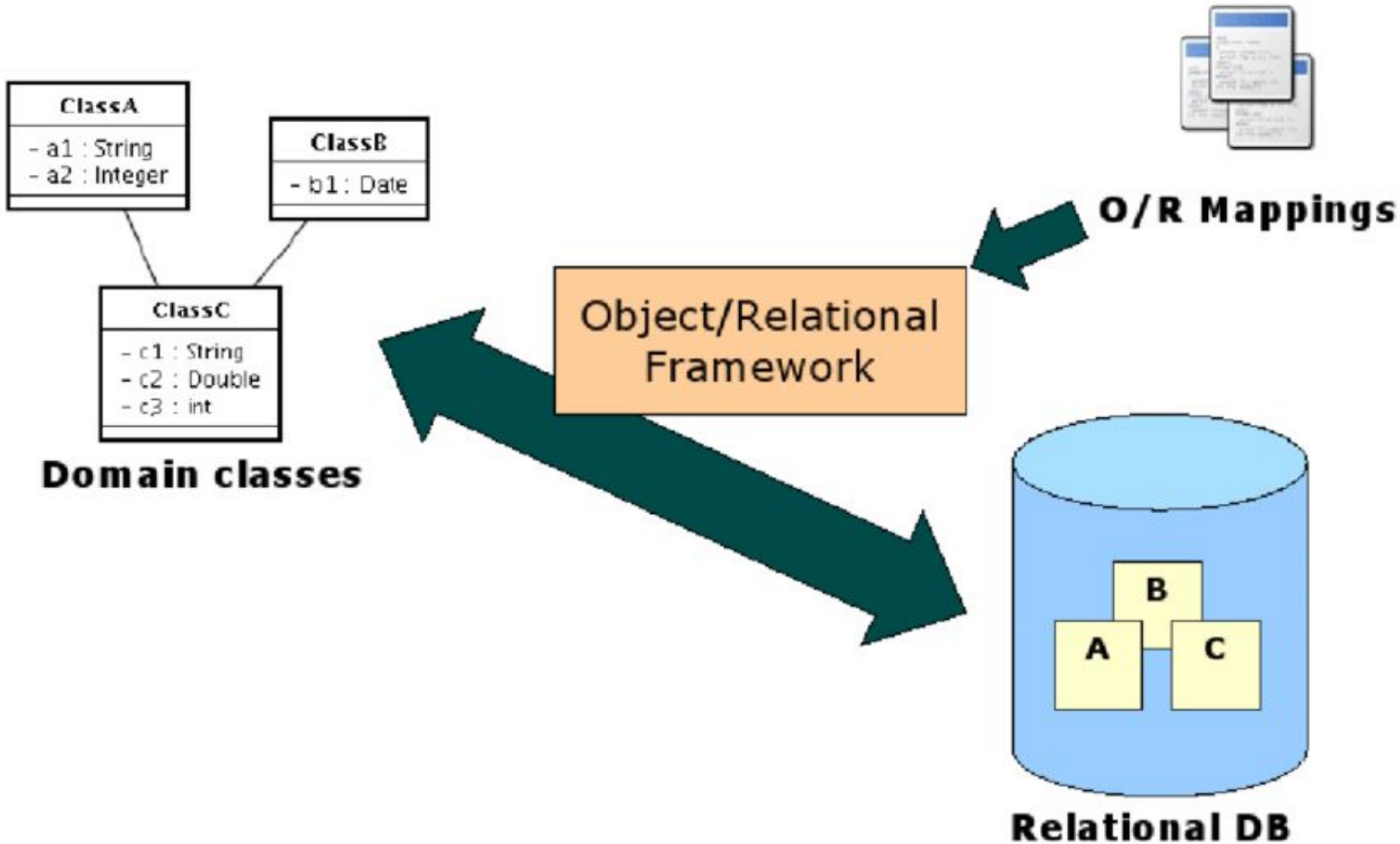
Roman Tikhiy

ENTITY FRAMEWORK

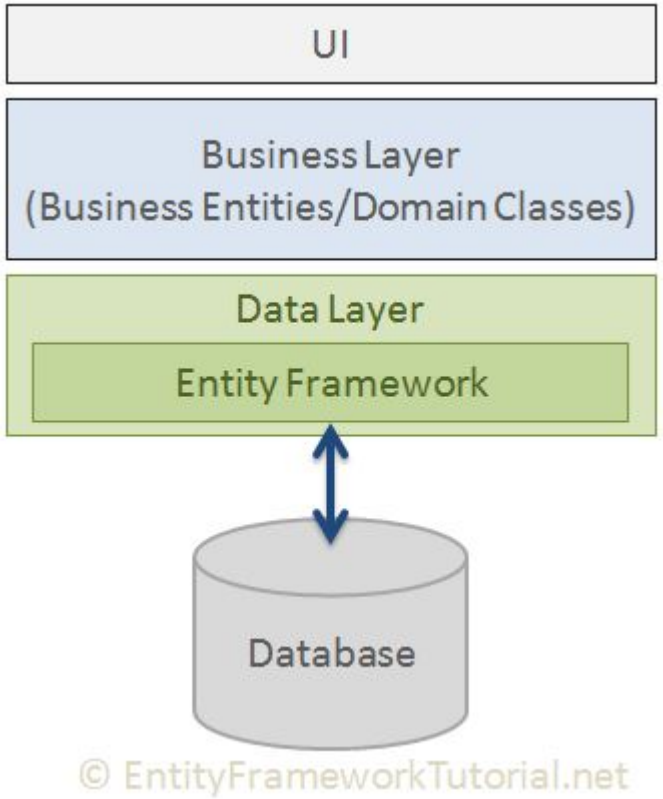
ORM(Object-Relational Mapping) is a technique that lets you query and manipulate data from a database using an object-oriented paradigm.



Essence of ORM



Entity framework core



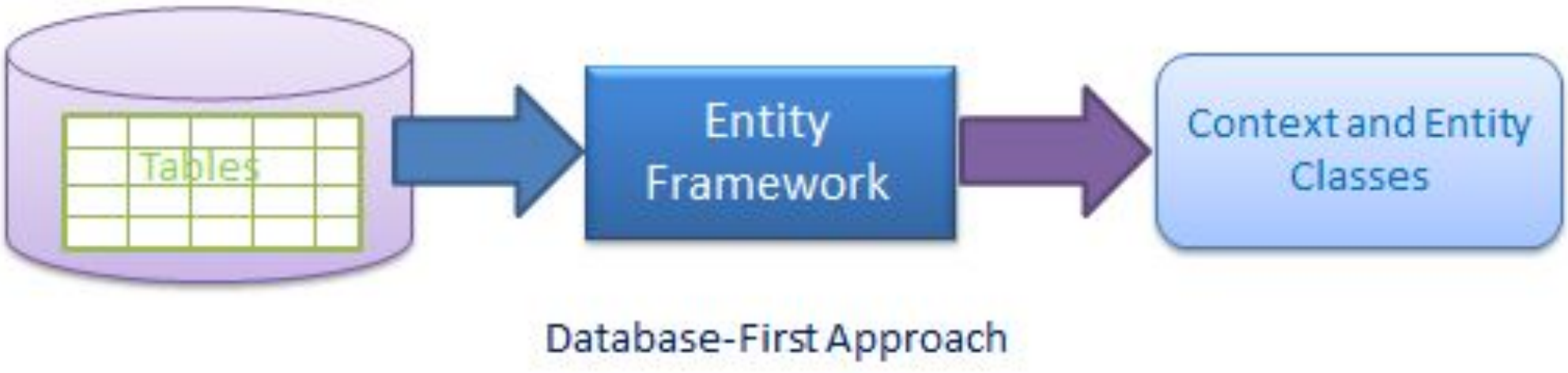
Entity Framework

Core

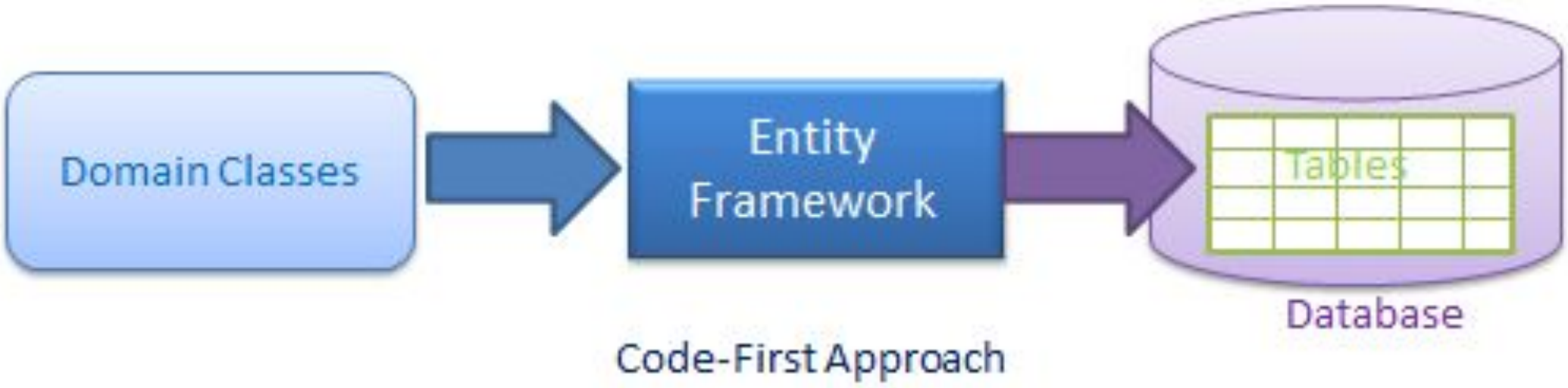
Current versions

EF 6	EF Core
✓ First released in 2008 with .NET Framework 3.5 SP1	✓ First released in June 2016 with .NET Core 1.0
✓ Stable and feature rich	✓ New and evolving
✓ Windows only	✓ Windows, Linux, OSX
✓ Works on .NET Framework 3.5+	✓ Works on .NET Framework 4.5+ and .NET Core
✓ Open-source	✓ Open-source

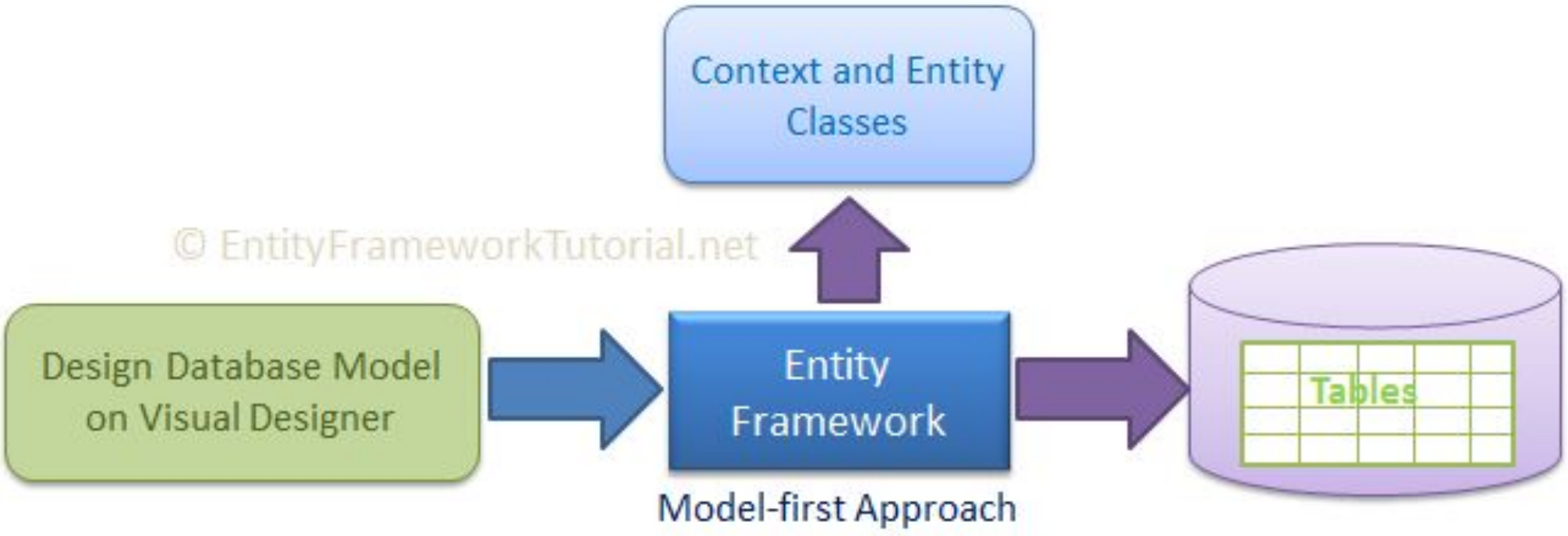
Database first



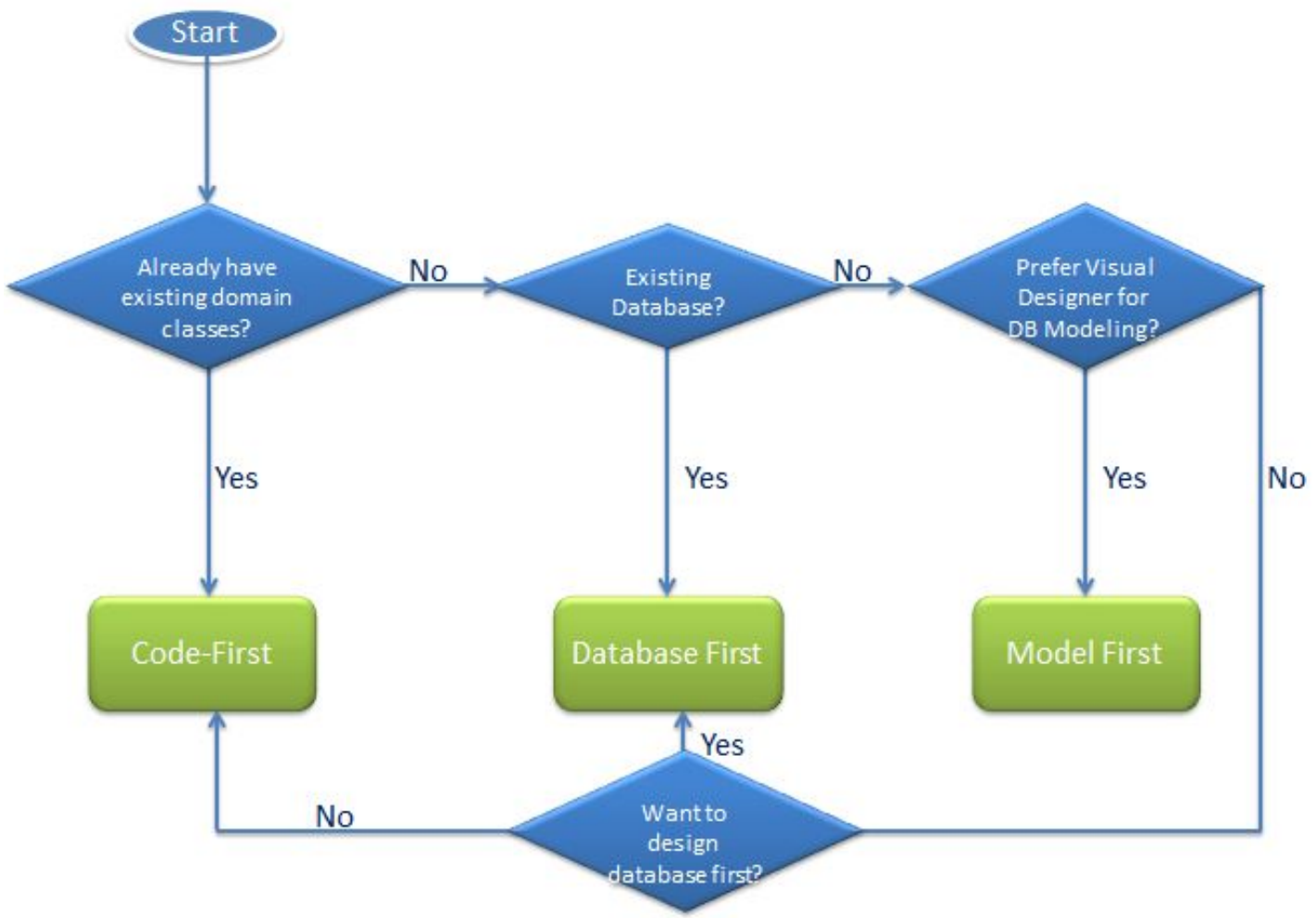
Code first


Domain Classes
Entity Framework
Tables
Database
Code-First Approach

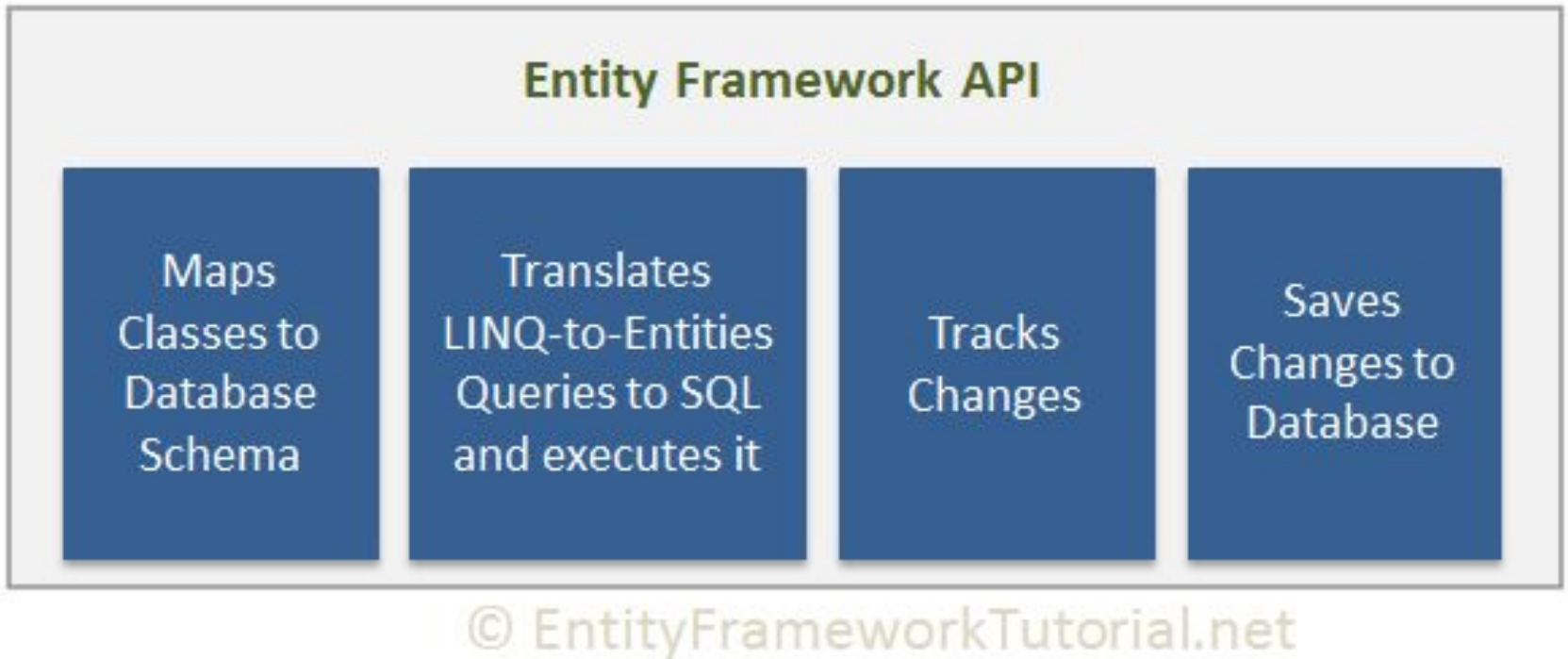
Model first


Context and Entity Classes
© EntityFrameworkTutorial.net
Design Database Model on Visual Designer
Entity Framework
Tables
Model-first Approach

What to choose?

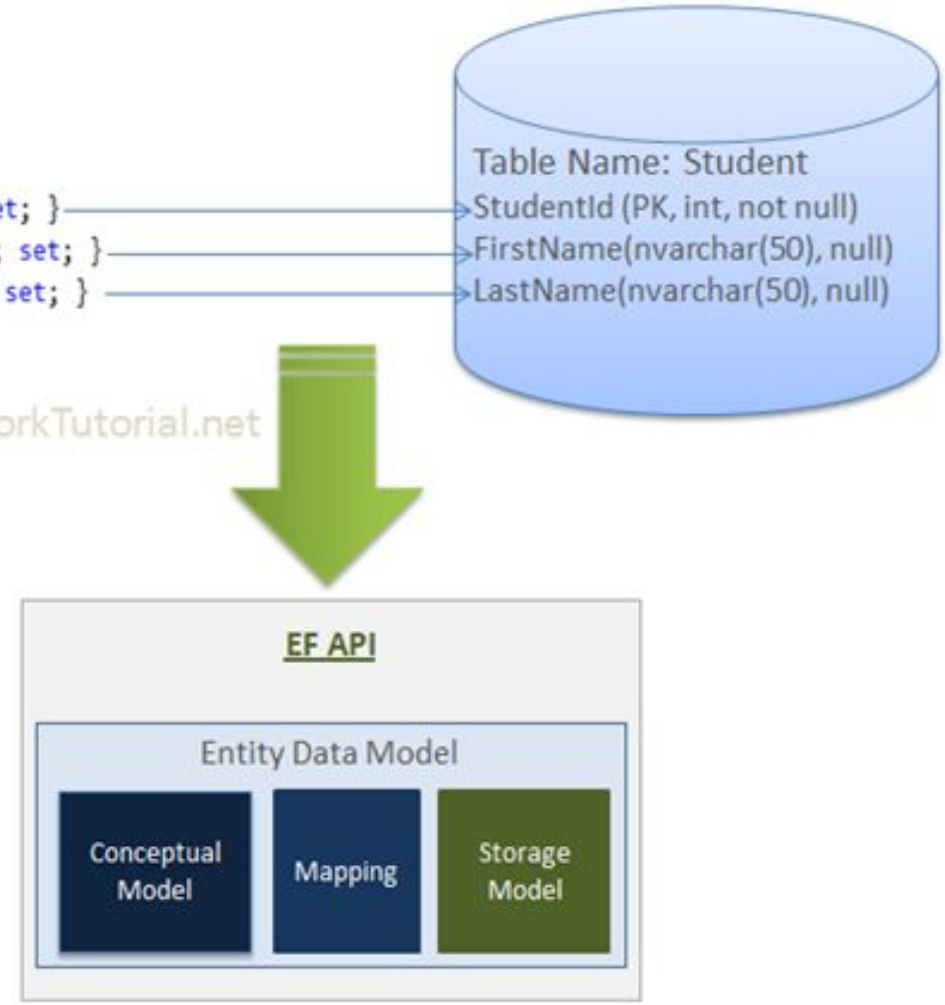


How it works


Entity Framework API
Maps Classes to Database Schema
Translates LINQ-to-Entities Queries to SQL and executes it
Tracks Changes
Saves Changes to Database
© EntityFrameworkTutorial.net

Mapping

```
public class Student
{
    public int StudentId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```



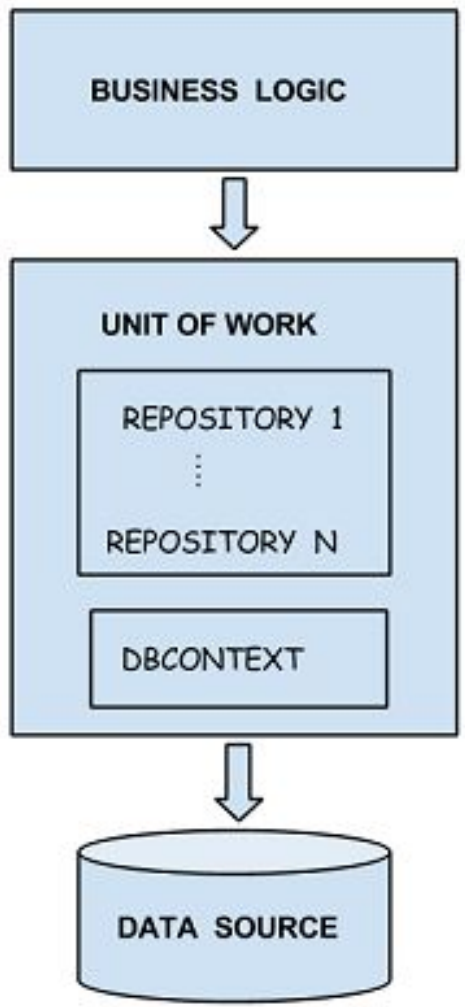
Entity

```
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public byte[]  Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }

    public Grade Grade { get; set; }
}

public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Students { get; set; }
}
```

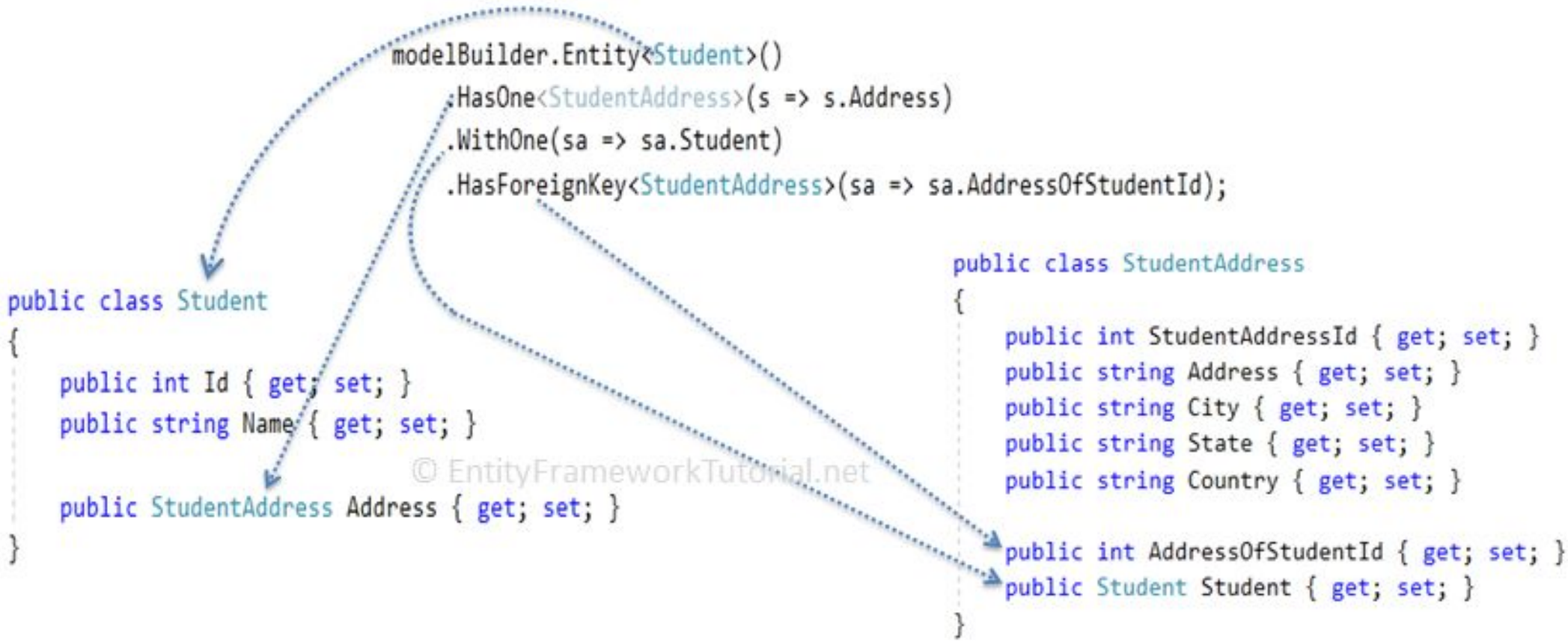
DbContext

```
public class SchoolContext : DbContext
{
    public SchoolContext()
    {

    }

    public DbSet<Student> Students { get; set; }
    public DbSet<Grade> Grades { get; set; }
}
```

Unit of work + repository



Fluent api


modelBuilder.Entity<Student>()
.HasOne<StudentAddress>(s => s.Address)
.WithOne(sa => sa.Student)
.HasForeignKey<StudentAddress>(sa => sa.AddressOfStudentId);
public class Student
{
public int Id { get; set; }
public string Name { get; set; }
public StudentAddress Address { get; set; }
}
public class StudentAddress
{
public int StudentAddressId { get; set; }
public string Address { get; set; }
public string City { get; set; }
public string State { get; set; }
public string Country { get; set; }
public int AddressOfStudentId { get; set; }
public Student Student { get; set; }
}
© EntityFrameworkTutorial.net

More fluent api

```
addressBuilder.HasKey(a => a.Id);

addressBuilder
    .HasOne(a => a.Employee)
    .WithMany(e => e.Addresses)
    .HasForeignKey(a => a.EmployeeId);

addressBuilder.HasIndex(a =>
new
{
    a.City,
    a.Country,
    a.Street
}).ForMySqlIsFullText();
```

Insert

```
using (var context = new SchoolContext())
{
    var std = new Student()
    {
        FirstName = "Bill",
        LastName = "Gates"
    };
    context.Students.Add(std);

    // or
    // context.Add<Student>(std);

    context.SaveChanges();
}
```

```
exec sp_executesql N'SET NOCOUNT ON;
INSERT INTO [Students] ( [FirstName], [LastName])
VALUES (@p0, @p1);
SELECT [StudentId]
FROM [Students]
WHERE @@ROWCOUNT = 1 AND [StudentId] = scope_identity();',N
'@p0 nvarchar(4000), @p1 nvarchar(4000) ',@p0=N'Bill',@p1=N'Gates'
go
```

Select with related entities

```
var context = new SchoolContext();

var studentWithGrade = context.Students
                            .Where(s => s.FirstName == "Bill")
                            .Include(s => s.Grade)
                            .FirstOrDefault();
```

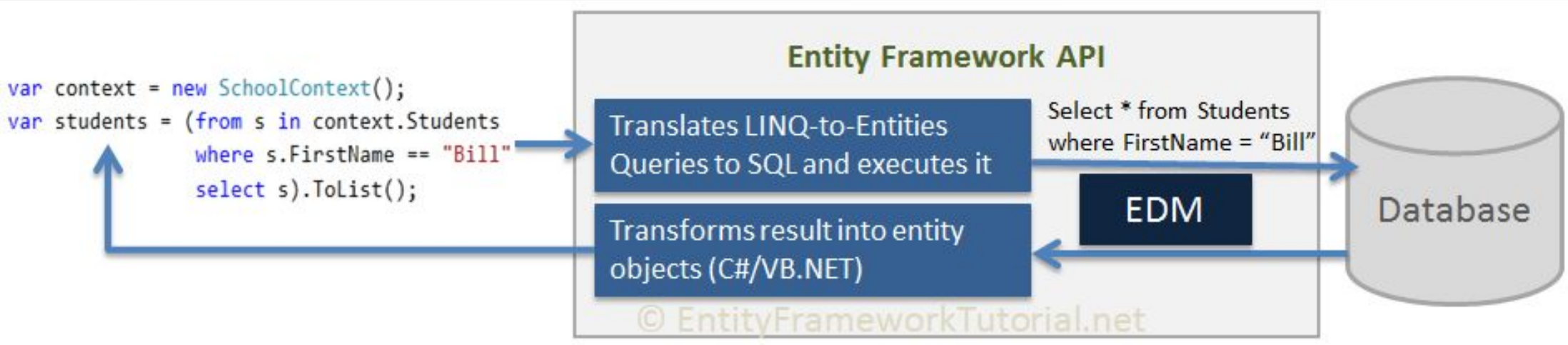
```
SELECT TOP(1) [s].[StudentId], [s].[DoB], [s].[FirstName], [s].[GradeId],[s].[LastName],
       [s].[MiddleName], [s.Grade].[GradeId], [s.Grade].[GradeName], [s.Grade].[Section]
FROM [Students] AS [s]
LEFT JOIN [Grades] AS [s.Grade] ON [s].[GradeId] = [s.Grade].[GradeId]
WHERE [s].[FirstName] = N'Bill'
```

Lazy loading

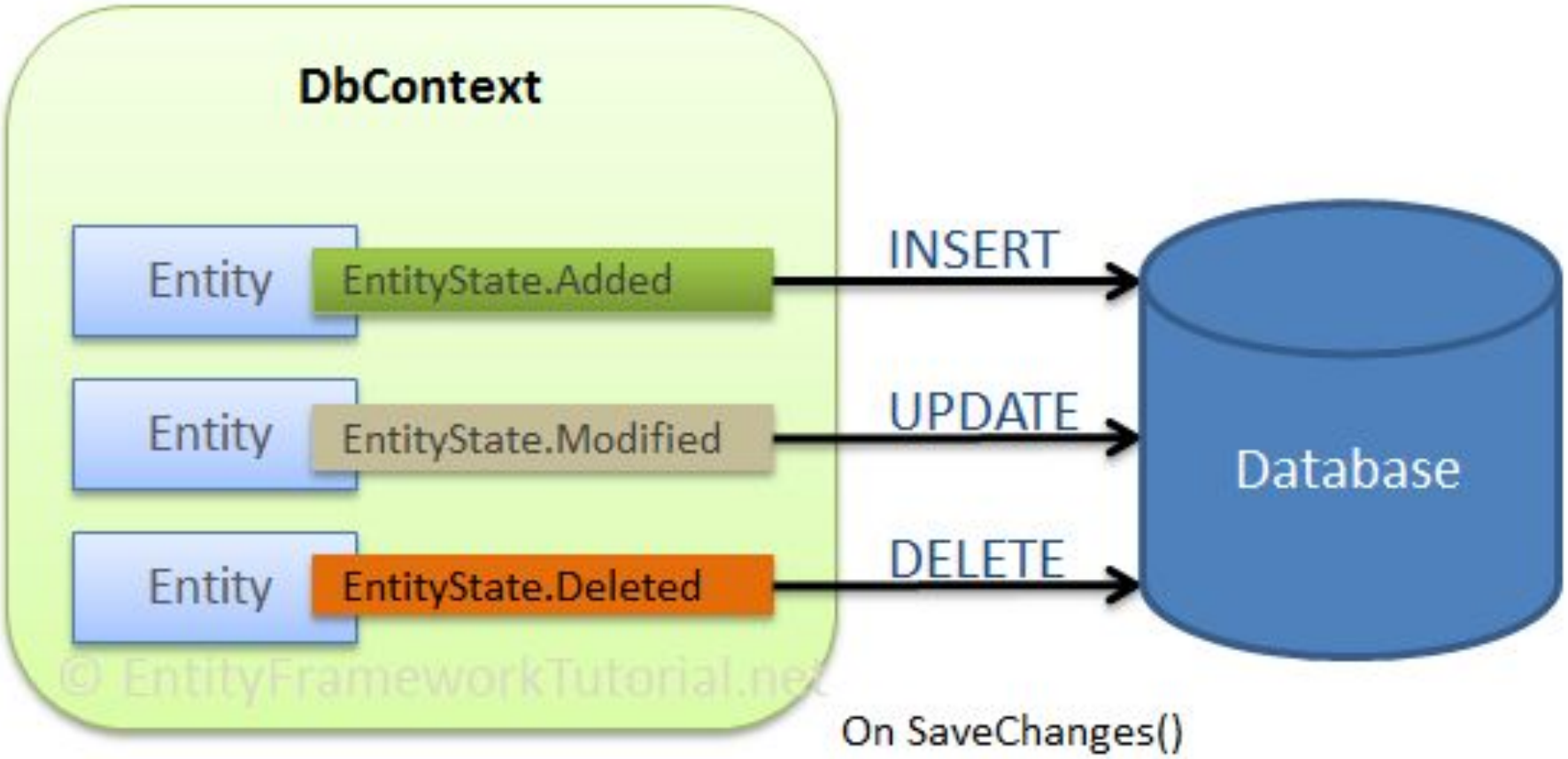
```
.AddDbContext<BloggingContext>(
    b => b.UseLazyLoadingProxies()
          .UseSqlServer(myConnectionString));
```

```
public class Blog
{
    public int Id { get; set; }
    public string Name { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
}
```

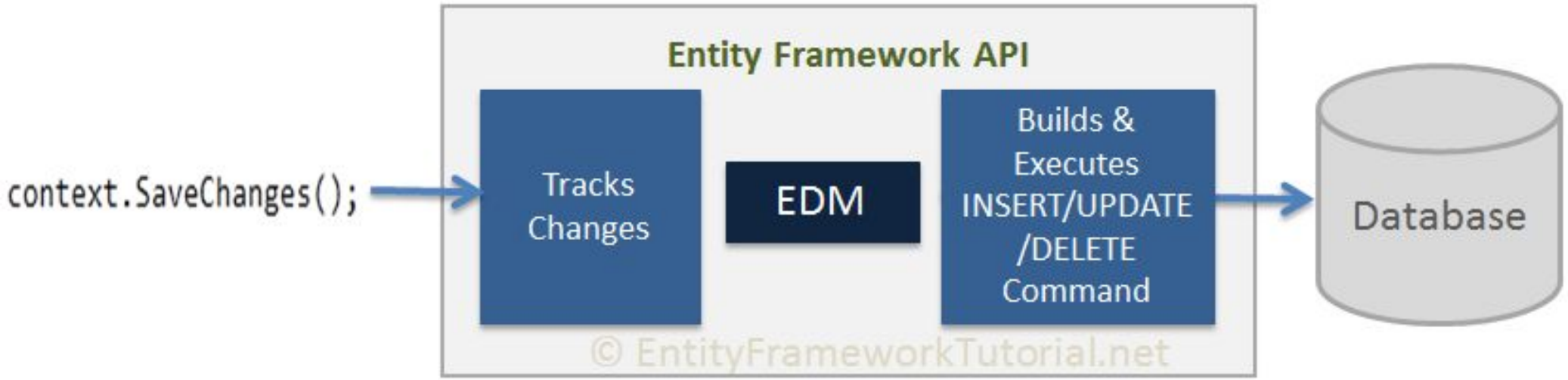
Querying


var context = new SchoolContext();
var students = (from s in context.Students
where s.FirstName == "Bill"
select s).ToList();
Entity Framework API
Translates LINQ-to-Entities Queries to SQL and executes it
Transforms result into entity objects (C#/VB.NET)
Select * from Students where FirstName = "Bill"
EDM
Database
© EntityFrameworkTutorial.net

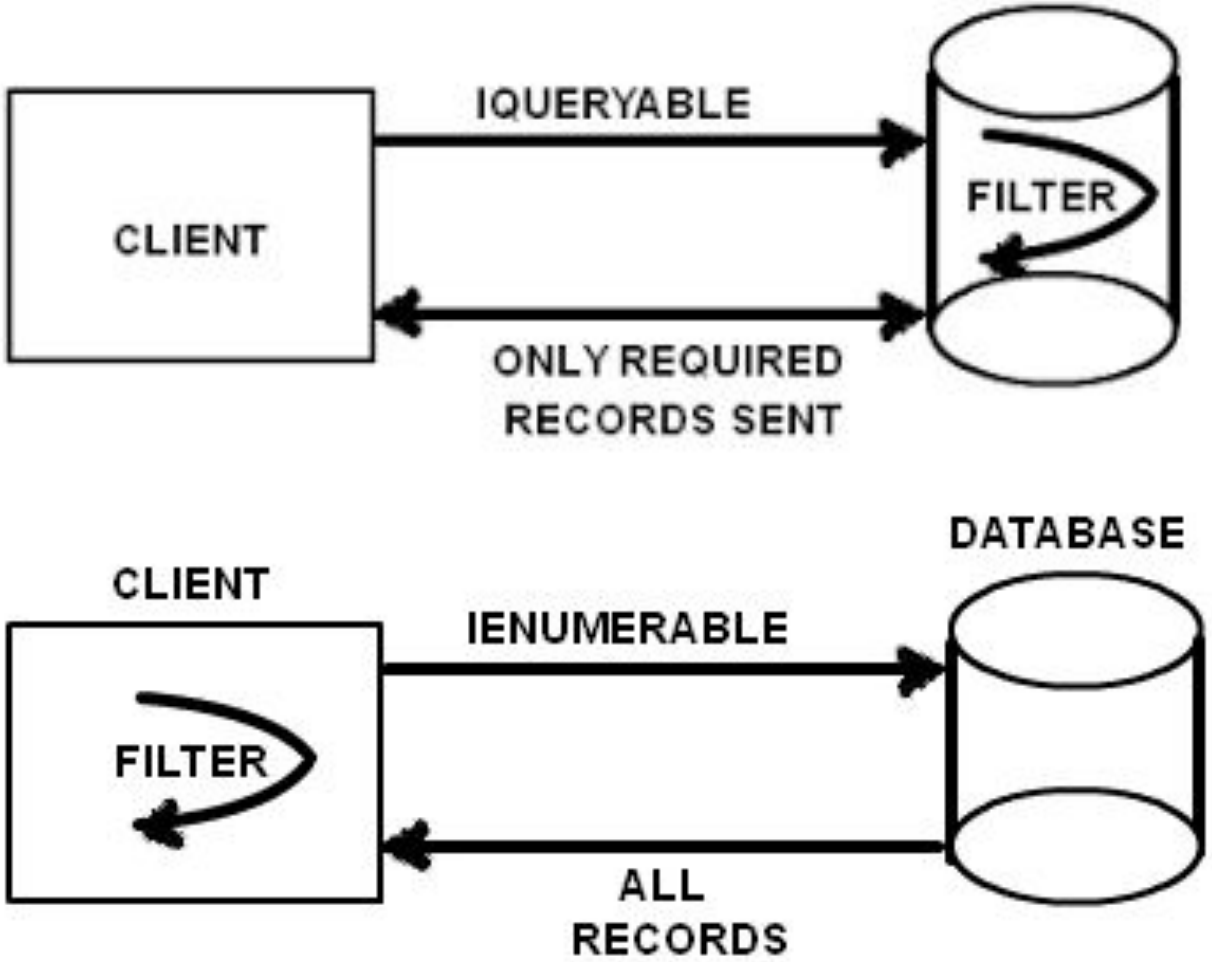
Entity state



Saving changes



IQueryable vs IEnumerable



Migrations

PMC Command	dotnet CLI command	Usage
add-migration <migration name>	Add <migration name>	Creates a migration by adding a migration snapshot.
Remove-migration	Remove	Removes the last migration snapshot.
Update-database	Update	Updates the database schema based on the last migration snapshot.
Script-migration	Script	Generates a SQL script using all the migration snapshots.

Dapper

Simple object mapper for .NET and own the title of King of Micro ORM in terms of speed and is virtually as fast as using a raw ADO.NET data reader.

Advantages and Disadvantages

Advantages

- Performance
- Easy integration
- Supports stored procedures
- Supports transactions

Disadvantages

- Attention to data types
- Support
- A lot of SQL in the code

How it works ?

- Create instance of IDbConnection
- Write an sql query
- Pass query as a parameter to `Execute` or `Query` method

Get all

```
public List<User> GetUsers()
{
    using (IDbConnection db = new SqlConnection(connectionString))
    {
        return db.Query<User>("SELECT * FROM Users").ToList();
    }
}
```

Get by id

```
public User Get(int id)
{
    using (IDbConnection db = new SqlConnection(connectionString))
    {
        return db.Query<User>("SELECT * FROM Users WHERE Id = @id", new { id }).FirstOrDefault();
    }
}
```

Insert

```
using (IDbConnection db = new SqlConnection(connectionString))
{
    var sqlQuery = "INSERT INTO Users (Name, Age) VALUES(@Name, @Age)";
    db.Execute(sqlQuery, user);
```

Update

```
using (IDbConnection db = new SqlConnection(connectionString))
{
    var sqlQuery = "UPDATE Users SET Name = @Name, Age = @Age WHERE Id = @Id";
    db.Execute(sqlQuery, user);
}
```

Delete

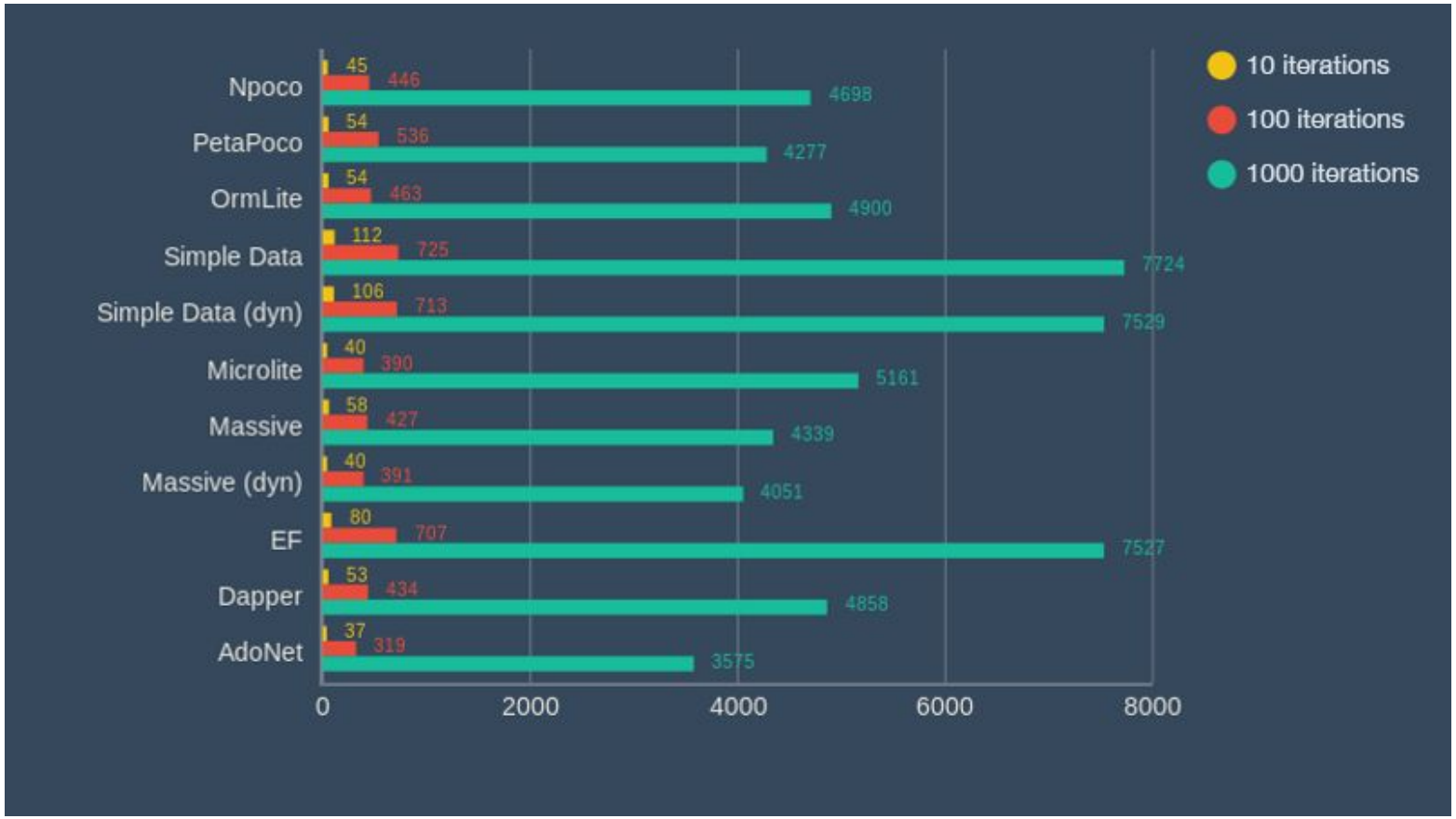
```
using (IDbConnection db = new SqlConnection(connectionString))
{
    var sqlQuery = "DELETE FROM Users WHERE Id = @id";
    db.Execute(sqlQuery, new { id });
}
```

Get with related data

```
using (var connection = new SqlConnection(FiddleHelper.GetConnectionStringSqlServ
{
    var orderDictionary = new Dictionary<int, Order>();


    var list = connection.Query<Order, OrderDetail, Order>(
        sql,
        (order, orderDetail) =>
        {
            Order orderEntry;

            if (!orderDictionary.TryGetValue(order.OrderID, out orderEntry))
            {
                orderEntry = order;
                orderEntry.OrderDetails = new List<OrderDetail>();
                orderDictionary.Add(orderEntry.OrderID, orderEntry);
            }

            orderEntry.OrderDetails.Add(orderDetail);
            return orderEntry;
        },
        splitOn: "OrderDetailID")
    .Distinct()
    .ToList();
```

Benchmarks

```
SELECT [WorkOrderID] AS Id, P.Name AS ProductName, [OrderQty] AS Quantity, [DueDate] AS Date
FROM [AdventureWorks2014].[Production].[WorkOrder] AS WO
INNER JOIN[Production].[Product] AS P ON P.ProductID = WO.ProductID
WHERE WorkOrderID = @Id
```



Thank you!