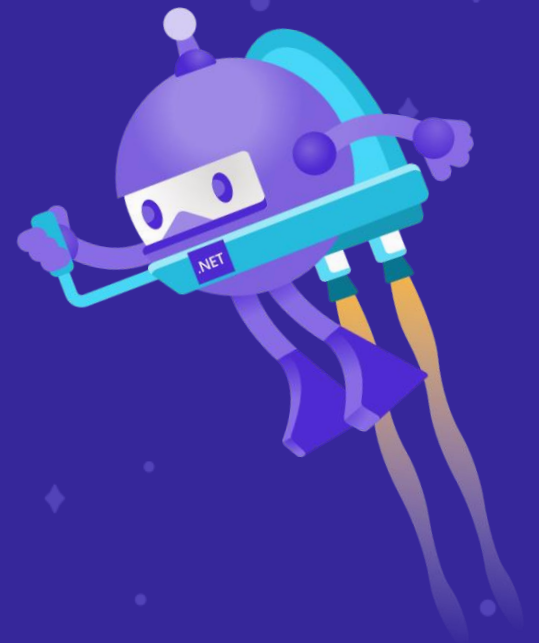


.NET Conf 2022 x Seoul

풀 스택과 **사랑**에 빠질 준비, 되셨나요?

ORM의 특성을 비교해보자.

Jungwoo Kim | rokag3@gmail.com



김정우

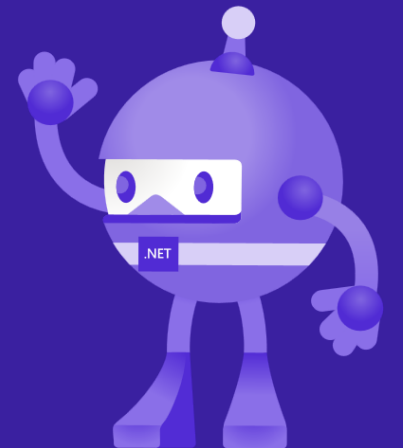
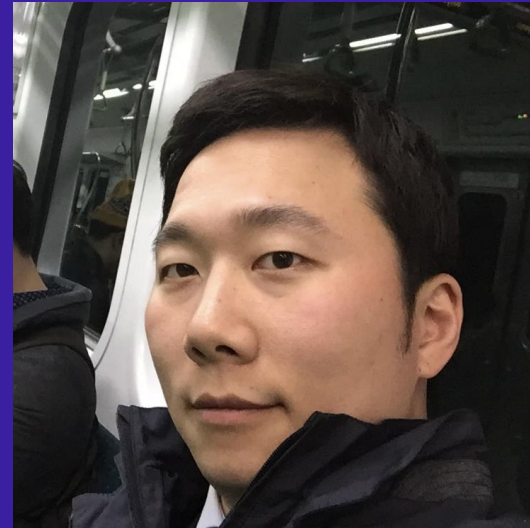
블로그: rokag3-gb.github.io



Service Development Team Leader

주로 .NET을 사용하여 웹앱, 서버, 응용앱을 개발하고 그 어플리케이션이 서비스 목표를 달성하는지에 집중하고 있습니다.

Cloud Native 기반의 어플리케이션을 개발하고자 매진하고 있습니다.



RELEASED

오늘 이야기할 것들

1. 주요 ORM 특징
2. ORM의 좋은 기능
3. ORM 샘플 소스코드
4. ORM 성능 테스트



1. 주요 ORM 특징

Entity Framework, Dapper, ADO.NET



1. 주요 ORM 특징 - Entity Framework

- Microsoft에서 개발하였고, 널리 사용되는 만큼 다양한 레퍼런스를 찾을 수 있습니다.
- .NET Framework 업데이트에 맞춰서, 또는 그보다 빠르게 새 기능이 추가되거나 다양한 기능들이 보완됩니다.
- ORM 객체를 통해 매개변수화 된 SQL을 사용하는 경우 SQL Injection 공격으로부터 안전합니다.
- SQL을 몰라도 프로그램을 작성할 수 있습니다.

1. 주요 ORM 특징 - EF

- Model class를 통합하여 제대로 관리하고 있다면, 유지보수 과정에서 발생할 수 있는 다양하고 복잡한 스키마 변경에 대해 효율적으로 대응할 수 있습니다.
- 성능을 향상시키기 위해 Memory-optimized table을 활성화하여 개발할 수 있습니다.
- SQL Server를 연동하여 사용할 경우 application이 더욱 적은 role을 부여받아도 개발이 가능해집니다.

1. 주요 ORM 특징 - EF

- 대량 업데이트에는 적절하지 않습니다.
- 아래 예시에서는, foreach 문 안에서 테이블 레코드 수 만큼 데이터베이스와 왕복이 발생하고, SaveChanges() 할 때 다시 레코드 수 만큼 데이터베이스와 왕복이 발생하게 됩니다.

```
try
{
    using (SchoolContext db = new SchoolContext())
    {
        foreach (var Student in db.Students)
        {
            Student.Age += 1;
        }

        db.SaveChanges();
    }
}
```

```
db.Database.ExecuteSqlCommand(
    "UPDATE dbo.Student SET Age = Age + 1;"
);
```


1. 주요 ORM 특징 - Dapper

- StackExchange에서 개발한 경량 ORM 입니다.
- `SqlMapper.Query<T>` 에 복잡한 조회 쿼리와 파라미터가 들어갈 수 있으며 `<T>` 로 제공받은 generic에 대한 class를 인스턴스화 하고, 이 인스턴스의 목록을 `IEnumerable<T>` 으로 반환합니다.

```
using (var con = new SqlConnection(connStr))
{
    var dataList = SqlMapper.Query<data2>(con
        , "select a.col1 from tbl_A a where a.col1 between 1 and @col1"
        , new { col1 = 999 })
        .ToList();

    label1.Text = dataList.Count.ToString();
}
```

1. 주요 ORM 특징 - Dapper

- 조회한 결과를 전달받을 때, 쿼리 결과의 데이터 타입이 model class (또는 Record) 의 데이터 타입과 호환되지 않으면 에러가 발생합니다.
- SQL에 대한 어느 정도 이해가 필요합니다.

1. 주요 ORM 특징 - 비교표

	ADO.NET	Entity Framework	Dapper
제작자	Microsoft	Microsoft	StackExchange
Initial release at	2005년 (since .NET 2.0)	2007년 (since .NET 3.5)	2011년
SQL Injection	취약	안전	보통
Select 반환 형식	DataSet	IEnumerable<T>	IEnumerable<T>
퍼포먼스	좋음	보통	좋음
장점	<ul style="list-style-type: none"> - DB 객체지향 관점의 개발팀에 적합 - 명확한 DB 성능 측정 가능 	<ul style="list-style-type: none"> - SQL 지식없이도 개발 가능 - DB 모든 기능 핸들링 가능 	<ul style="list-style-type: none"> - 가볍고 빠르다. - 복잡한 조회도 비교적 쉽게 구현
		<ul style="list-style-type: none"> - Model 기반으로 개발 편리성 - MVC, MVVM 패턴에 유리 	
단점	DataTable에 많은 cast 필요	<ul style="list-style-type: none"> - EF에 대한 러닝 커브 - Model이 복잡해지면 소스가 복잡해짐 	<ul style="list-style-type: none"> - 지원되는 기능 부족 - 2011~ 50번

1. 주요 ORM 특징 - ADO.NET의 단점

- 서버에서 DataSet 형식으로 리턴해주면 클라이언트에서는 DataSet.Table[n]을 추가로 가공하여 다양한 cast 작업을 하여 내부에서 활용하게 됩니다.
- 프로그램의 백엔드에서 바라보는 model에 데이터를 치환하는 로직을 개발자가 직접 커버해야 했습니다.

```
SqlDataAdapter adapter = new SqlDataAdapter("StoredProcedureName", con);

adapter.SelectCommand.CommandType = CommandType.StoredProcedure;
adapter.SelectCommand.Parameters.AddRange(sqlParam);

DataSet filledDataSet = new DataSet();
adapter.Fill(filledDataSet);
```

1. 주요 ORM 특징 - ADO.NET의 단점

- 추억의 Structured type은 굉장히 편리했으나 그만큼 보안 취약점도 많이 대두되었습니다.

```
DataTable dt = new DataTable();
dt.Rows.Add(
    // payload
);

sqlParam[n].SqlDbType = SqlDbType.Structured;
sqlParam[n].Value = dt;
```

CA2350: <code>DataTable.ReadXml()</code> 의 입력을 신뢰할 수 있는지 확인하세요.	신뢰할 수 없는 입력으로 <code>DataTable</code> 을 역직렬화하면 공격자는 악의적인 입력을 만들어 서비스 거부 공격을 수행할 수 있습니다. 알 수 없는 원격 코드 실행 취약성이 있을 수 있습니다.
CA2351: <code>DataSet.ReadXml()</code> 의 입력을 신뢰할 수 있는지 확인하세요.	신뢰할 수 없는 입력으로 <code>DataSet</code> 를 역직렬화하면 공격자는 악의적인 입력을 만들어 서비스 거부 공격을 수행할 수 있습니다. 알 수 없는 원격 코드 실행 취약성이 있을 수 있습니다.
CA2352: 직렬화 가능 형식의 안전하지 않은 데이터 세트 또는 <code>DataTable</code> 은 원격 코드 실행 공격에 취약할 수 있습니다.	<code>SerializableAttribute</code> 로 표시된 클래스 또는 구조체에 <code>DataSet</code> 또는 <code>DataTable</code> 필드나 속성이 포함되어 있으며 <code>GeneratedCodeAttribute</code> 가 없습니다.
CA2353: 직렬화 가능 형식의 안전하지 않은 데이터 세트 또는 <code>DataTable</code> 입니다.	XML serialization 특성 또는 데이터 계약 특성으로 표시된 클래스 또는 구조체에 <code>DataSet</code> 또는 <code>DataTable</code> 필드나 속성이 포함되어 있습니다.
CA2354: 역직렬화된 개체 그래프의 안전하지 않은 데이터 세트 또는 <code>DataTable</code> 은 원격 코드 실행 공격에 취약할 수 있습니다.	직렬화된 <code>System.Runtime.Serialization.IFormatter</code> 로 역직렬화하고 캐스트된 형식의 개체 그래프에 <code>DataSet</code> 또는 <code>DataTable</code> 이 포함될 수 있습니다.
CA2355: 역직렬화된 개체 그래프의 안전하지 않은 데이터 세트 또는 <code>DataTable</code>	캐스팅되거나 지정된 형식의 개체 그래프에 <code>DataSet</code> 또는 <code>DataTable</code> 이 포함될 수 있는 경우 역직렬화합니다.
CA2356: 웹 역직렬화된 개체 그래프의 안전하지 않은 <code>DataSet</code> 또는 <code>DataTable</code>	<code>System.Web.Services.WebMethodAttribute</code> 또는 <code>System.ServiceModel.OperationContractAttribute</code> 를 사용하는 메서드에 <code>DataSet</code> 또는 <code>DataTable</code> 을 참조할 수 있는 매개 변수가 있습니다.
CA2361: <code>DataSet.ReadXml()</code> 을 포함하는 자동 생성된 클래스가 신뢰할 수 없는 데이터와 함께 사용되지 않도록 하기	신뢰할 수 없는 입력으로 <code>DataSet</code> 를 역직렬화하면 공격자는 악의적인 입력을 만들어 서비스 거부 공격을 수행할 수 있습니다. 알 수 없는 원격 코드 실행 취약성이 있을 수 있습니다.
CA2362: 자동 생성된 직렬화 가능 형식의 안전하지 않은 <code>DataSet</code> 또는 <code>DataTable</code> 은 원격 코드 실행 공격에 취약할 수 있음	<code>BinaryFormatter</code> 를 사용하여 신뢰할 수 없는 입력을 역직렬화할 때 역직렬화된 개체 그래프에 <code>DataSet</code> 또는 <code>DataTable</code> 이 포함되어 있으면 공격자가 악의적인 페이로드를 만들어 원격 코드 실행 공격을 수행할 수 있습니다.

2. ORM의 좋은 기능



2. ORM의 좋은 기능 - EF

- EF에는 좋은 기능들이 많이 있습니다.
- Extension을 활용하여 Update from, Delete from을 할 수 있습니다.

```
using (SchoolContext db = new SchoolContext())
{
    db.Students
        .Where(x => x.EnrollmentDate > new DateTime(2020, 1, 1))
        .UpdateFromQuery(x => new Student { Age = 0 });
}

using (SchoolContext db = new SchoolContext())
{
    db.Enrollments
        .Where(x => x.CourseID == 1050)
        .DeleteFromQuery();
}
```

2. ORM의 좋은 기능 - LINQ

- System.Linq를 통하여 outer apply을 구현할 수 있습니다.

```
using SchoolContext dc = new SchoolContext();

/*
select  a.*
        , b.LastName
        , b.FirstMidName
from    Enrollments a
        outer apply (
            select top 1 b.LastName, b.FirstMidName
            from      Student b
            where     b.ID == a.StudentID
        ) b
where   a.EnrollmentID > 8;
*/
var result = from a in dc.Enrollments
              .Where(a => a.EnrollmentID > 8)
              from b in dc.Students
                  .Where(b => b.ID == a.StudentID)
                  .Take(1) // top 1
              select new { a, b.LastName, b.FirstMidName };

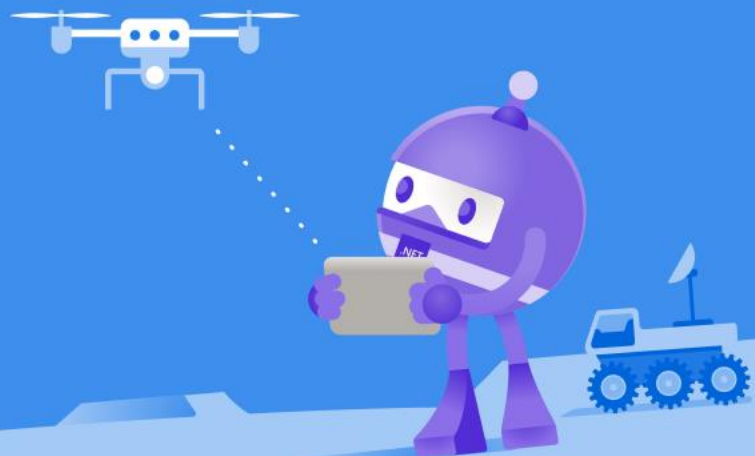
foreach (var row in result)
{
    // do something
}
```


2. ORM의 좋은 기능 - Dapper

- SQL Server에서 prepared query로 인식하기 때문에 injection을 예방할 수 있습니다.
- DB 내에서 쿼리 구문 분석 과정이 단축되고, plan cache를 재사용할 수 있는 확률이 높아지기 때문에 성능 상 이득이 있습니다.

Results Messages												
	bucketid	refcounts	usecounts	size_in_bytes	memory_object_address	cacheobjtype	objtype	plan_handle	pool_id	parent_plan_handle	text	query_plan
1	5175	2	1	73728	0x00000206126EC060	Compiled Plan	Adhoc	0x06000900D2C...	2	NULL	select a.* , q.text , p.query_plan from sys.dm...	<ShowPlanXML xmlns="http://schemas.micr
2	12730	2	2	81920	0x000002063048E060	Compiled Plan	Adhoc	0x060009006122...	2	NULL	select a.* , q.text , p.query_plan from sys.dm...	<ShowPlanXML xmlns="http://schemas.micr
3	10815	2	5	40960	0x0000020615998060	Compiled Plan	Prepared	0x0600090014C2...	2	NULL	(@col1 int)insert into tbl_A (col1) values (@col1)	<ShowPlanXML xmlns="http://schemas.micr
4	15253	2	5	40960	0x00000206170B6060	Compiled Plan	Prepared	0x060009008CC...	2	NULL	(@1 int)INSERT INTO [tbl_A] values(@1)	<ShowPlanXML xmlns="http://schemas.micr
5	24218	2	5	49152	0x000002061AE04060	Compiled Plan	Prepared	0x060009006399...	2	NULL	(@col1 int)select a.col1 from tbl_A a where a.co...	<ShowPlanXML xmlns="http://schemas.micr

3. ORM 샘플 소스코드



3. ORM 샘플 소스코드 - ADO.NET

- ADO.NET에서 간단하게 참고할만한 소스 코드가 있습니다.
- SqlInfoMessageEventHandler를 활용하여 저장프로시저 단에서 발생하는 다양한 출력 메시지를 받아볼 수 있습니다.

```
con.InfoMessage += new SqlInfoMessageEventHandler((sender, eArgs) =>
{
    ... this.InfoMessage += eArgs.Message;
});
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

BEGIN TRY
BEGIN TRAN

PRINT '1st info message';

-- sql

PRINT '2nd info message';

-- sql

PRINT '3rd info message';

COMMIT TRAN
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0 ROLLBACK TRAN;
    DECLARE @ERR_MSG VARCHAR(2000) = ERROR_MESSAGE(),
            @ERR_SEVERITY INT = ERROR_SEVERITY(),
            @ERR_STATE INT = ERROR_STATE()
    RAISERROR(@ERR_MSG, @ERR_SEVERITY, @ERR_STATE)
    RETURN
END CATCH;
```

3. ORM 샘플 소스코드 - ADO.NET

- ExceptionExtensions을 활용하여 SqlException에 내가 원하는 형태로 메시지를 가공하여 throw할 수 있습니다.

```
catch (SqlException SqlEx)
{
    SqlEx.SetMessage(string.Concat("[", SqlEx.Number, "] ", SqlEx.Message));

    public static class ExceptionExtensions
    {
        // DotNet - How to change exception message of Exception object? ...
        public static void SetMessage(this Exception exception, string message)
        {
            var type = typeof(Exception);
            var flags = System.Reflection.BindingFlags.Instance
                | System.Reflection.BindingFlags.NonPublic;
            var fieldInfo = type.GetField("_message", flags);
            fieldInfo.SetValue(exception, message);
        }
    }

    throw SqlEx;
}
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

BEGIN TRY
BEGIN TRAN

PRINT '1st info message';

-- sql

PRINT '2nd info message';

-- sql

PRINT '3rd info message';

COMMIT TRAN
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0 ROLLBACK TRAN;
    DECLARE @ERR_MSG VARCHAR(2000) = ERROR_MESSAGE(),
            @ERR_SEVERITY INT = ERROR_SEVERITY(),
            @ERR_STATE INT = ERROR_STATE();
    RAISERROR(@ERR_MSG, @ERR_SEVERITY, @ERR_STATE)
    RETURN
END CATCH;
```

3. ORM 샘플 소스코드 - EF

- 먼저 Model class를 작성합니다.
- Student, Course, Enrollment

```
참조 11개
public enum Grade
{
    A, B, C, D, F
}

참조 18개
public class Enrollment
{
    참조 1개
    public int EnrollmentID { get; set; }
    참조 14개
    public int CourseID { get; set; }
    참조 14개
    public int StudentID { get; set; }
    참조 10개
    public Grade? Grade { get; set; }

    참조 0개
    public virtual Course Course { get; set; } // Foreign key
    참조 0개
    public virtual Student Student { get; set; } // Foreign key
}
```

```
using System.ComponentModel.DataAnnotations.Schema;

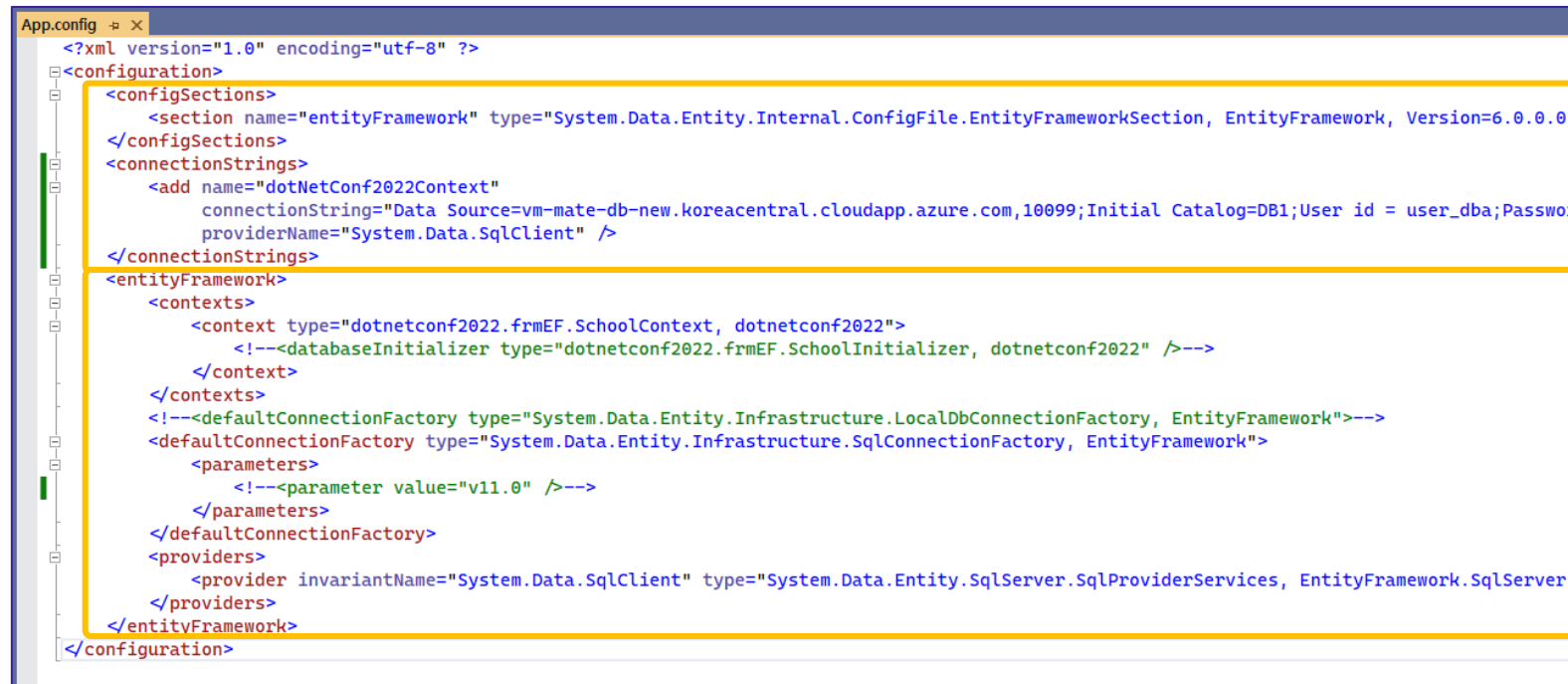
namespace dotnetconf2022.Model
{
    참조 18개
    public class Student
    {
        참조 9개
        public int ID { get; set; }
        참조 14개
        public string LastName { get; set; }
        참조 13개
        public string FirstMidName { get; set; }
        참조 7개
        public int Age { get; set; }
        참조 12개
        public DateTime EnrollmentDate { get; set; }
        참조 1개
        public virtual ICollection<Enrollment> Enrollments { get; set; }
    }

    참조 10개
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        참조 7개
        public int CourseID { get; set; }
        참조 7개
        public string Title { get; set; }
        참조 7개
        public int? Credits { get; set; } // int nullable 처리

        참조 0개
        public virtual ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

3. ORM 샘플 소스코드 - EF

- App.config, Web.config에 configSections, connectionStrings, entityFramework 에 대한 내용을 추가합니다.



The screenshot shows the App.config file in Visual Studio. The configuration is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0" />
  </configSections>
  <connectionStrings>
    <add name="dotNetConf2022Context"
          connectionString="Data Source=vm-mate-db-new.koreacentral.cloudapp.azure.com,10099;Initial Catalog=DB1;User id = user_dba;Password=12345678"
          providerName="System.Data.SqlClient" />
  </connectionStrings>
  <entityFramework>
    <contexts>
      <context type="dotnetconf2022.frmEF.SchoolContext, dotnetconf2022">
        <!--<databaseInitializer type="dotnetconf2022.frmEF.SchoolInitializer, dotnetconf2022" />-->
      </context>
    </contexts>
    <!--<defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework"-->
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework">
      <parameters>
        <!--<parameter value="v11.0" />-->
      </parameters>
    </defaultConnectionFactory>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
</configuration>
```

3. ORM 샘플 소스코드 - EF

- System.Data.Entity.DbContext를 상속받은 class를 작성하고, 해당 context 내부에서 사용할 DbSet<>을 선언합니다.

```
public class SchoolContext : DbContext
{
    참조 10개
    public DbSet<Student> Students { get; set; }
    참조 3개
    public DbSet<Enrollment> Enrollments { get; set; }
    참조 1개
    public DbSet<Course> Courses { get; set; }

    참조 0개
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}
```

3. ORM 샘플 소스코드 - EF

- EF에서 데이터를 조회하는 예시 입니다.

```
using (SchoolContext db = new SchoolContext())
{
    var resultset = db.Students
        .Where(p => p.ID ≥ 6)
        .OrderByDescending(p => p.LastName);

    if (resultset ≠ null)
    {
        foreach (var record in resultset)
        {
            Student row = (Student)record;

            // do something
        }
    }
}
```

```
using (SchoolContext dbcontext = new SchoolContext())
{
    // IQueryable
    var resultset = dbcontext.Students
        .Select(c => new { c.ID, c.Age, c.LastName, c.EnrollmentDate })
        .Where(p => p.ID ≥ 6);
    // .OrderByDescending(p => p.LastName);

    if (resultset ≠ null)
    {
        foreach (var r in resultset)
        {
            // do something
        }
    }
}
```


3. ORM 샘플 소스코드 - EF

- EF에서 데이터를 insert, update, delete 하는 예시 입니다.

```
Student new_student = new Student();
new_student.ID = 99;
new_student.LastName = "홍";
new_student.FirstMidName = "길동";
new_student.Age = 40;
new_student.EnrollmentDate =
    Convert.ToDateTime("2020-10-01");
new_student.Enrollments = new List<Enrollment> {
    new Enrollment {
        StudentID = 99, CourseID = 1045, Grade = Grade.A
    }
};

// 1건 insert
SchoolContext db = new();
db.Students.Add(new_student);
db.SaveChanges();
```

```
using SchoolContext db = new();

// 단일 1건 업데이트
var result = db.Students
    .SingleOrDefault(b => b.ID == 3);

if (result != null)
{
    result.FirstMidName = "길동";
    result.Age = 30;

    db.SaveChanges();
}
```

```
int affected_rows = 0;

using (SchoolContext dbcontext = new())
{
    // 1 record delete.
    var del_single = dbcontext.Students
        .SingleOrDefault(p => p.ID == 6);
    dbcontext.Students.Remove(del_single);

    affected_rows = dbcontext.SaveChanges();

    // multi records delete.
    var del_multi = dbcontext.Students
        .Where(p => (p.Age > 30));
    dbcontext.Students.RemoveRange(del_multi);

    affected_rows = dbcontext.SaveChanges();
}
```

3. ORM 샘플 소스코드 - Dapper

- Dapper에서 select, insert, ExecuteScalar 하는 예시 입니다.

```
private void Do_Select_Adhoc_param()
{
    using var con = new SqlConnection(connStr);

    var dataList = SqlMapper.Query<data2>(con
        , "select a.col1"
        + "from tbl_A a"
        + "where a.col1 between 1 and @col1"
        , new { col1 = 999 })
        .ToList();

    label1.Text = dataList.Count.ToString();
}
```

참조 4개

```
public record data2
{
    참조 2개
    public int col1 { get; set; }
}
```

```
using (var con = new SqlConnection(connStr))
{
    int affectedRows = con.Execute($"insert into tbl_A values ({DateTime.Now.Second})");

    label3.Text = $"affected rows: {affectedRows}";
}
```

```
using (var con = new SqlConnection(connStr))
{
    var data = con.ExecuteScalar<DateTime>("select getdate();");

    label2.Text = data.ToString();
}
```

3. ORM 샘플 소스코드 - Dapper

- DynamicParameters를 활용하여 Execute 하는 예시 입니다.
- Raw sql text를 받아서 요청하면 DB optimizer가 prepared query로 인식하기 때문에 DB가 제공하는 다양한 기능을 활용할 수 있습니다.

```
using var con = new SqlConnection(connStr);

string sql = "insert into tbl_A (col1) values (@col1)";

var dparam = new DynamicParameters();
dparam.Add("@col1", d.col1, DbType.Int32, ParameterDirection.Input);

int rows = con.Execute(sql, dparam);

label4.Text = $"affected rows: {rows}";
```

```
StringBuilder sbSQL = new StringBuilder();
sbSQL.AppendLine("select a.*");
sbSQL.AppendLine("    , b.LastName");
sbSQL.AppendLine("    , b.FirstMidName");
sbSQL.AppendLine("from Enrollments a");
sbSQL.AppendLine("    outer apply (");
sbSQL.AppendLine("        select top 1 b.LastName, b.FirstMidName");
sbSQL.AppendLine("        from Student b");
sbSQL.AppendLine("        where b.ID == a.StudentID");
sbSQL.AppendLine("    ) b");
sbSQL.AppendLine("where a.EnrollmentID >= @col1");

using var con = new SqlConnection(connStr);

var dataList = SqlMapper.Query<data2>(con
    , sbSQL.ToString()
    , new { col1 = 8 })
    .ToList();

label1.Text = dataList.Count.ToString();
```

4. ORM 성능 테스트



4. ORM 성능 테스트

Dapper_Select > deals.Count = 952

Loop count

Batch

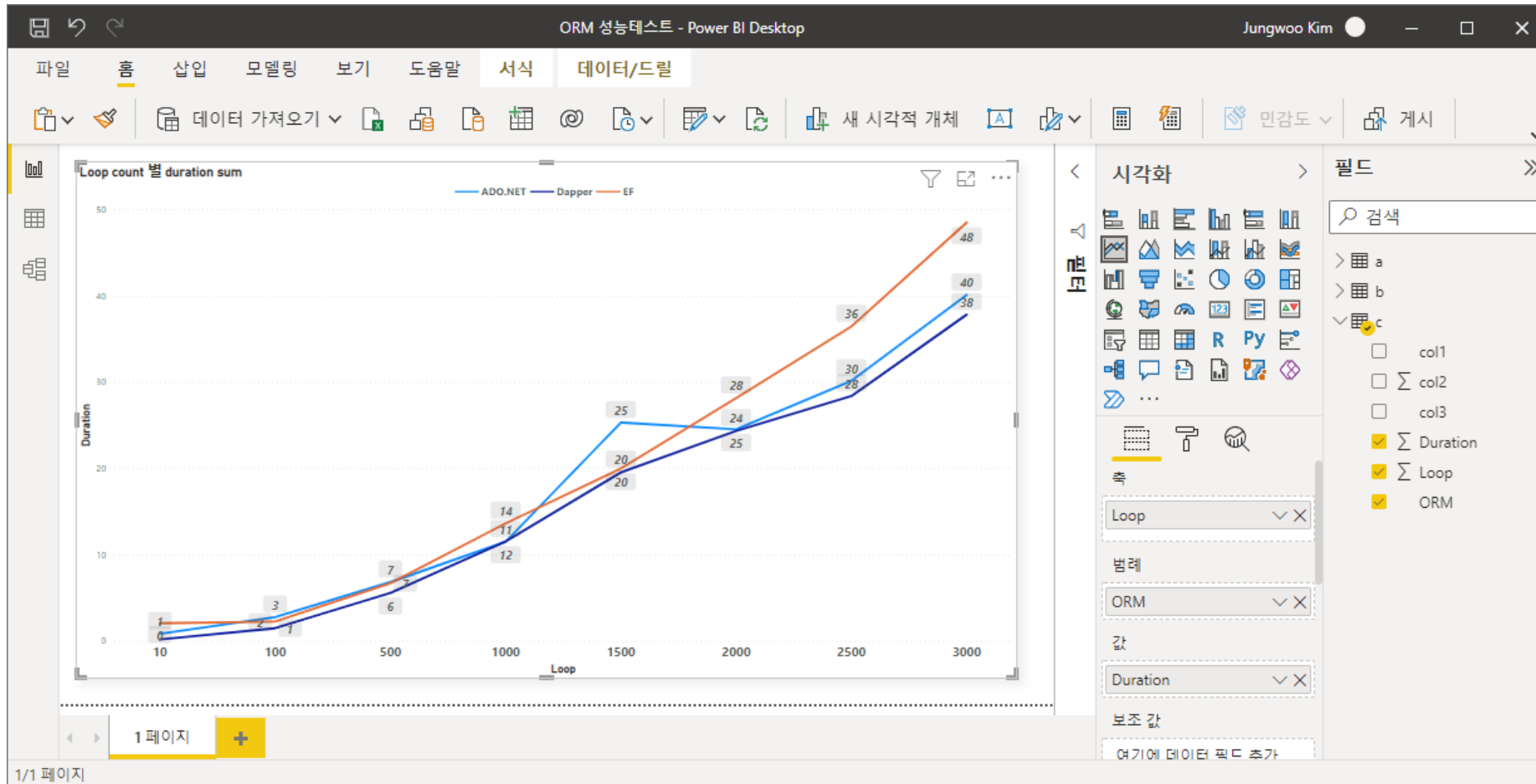
ADO.NET Select query

EF Select query

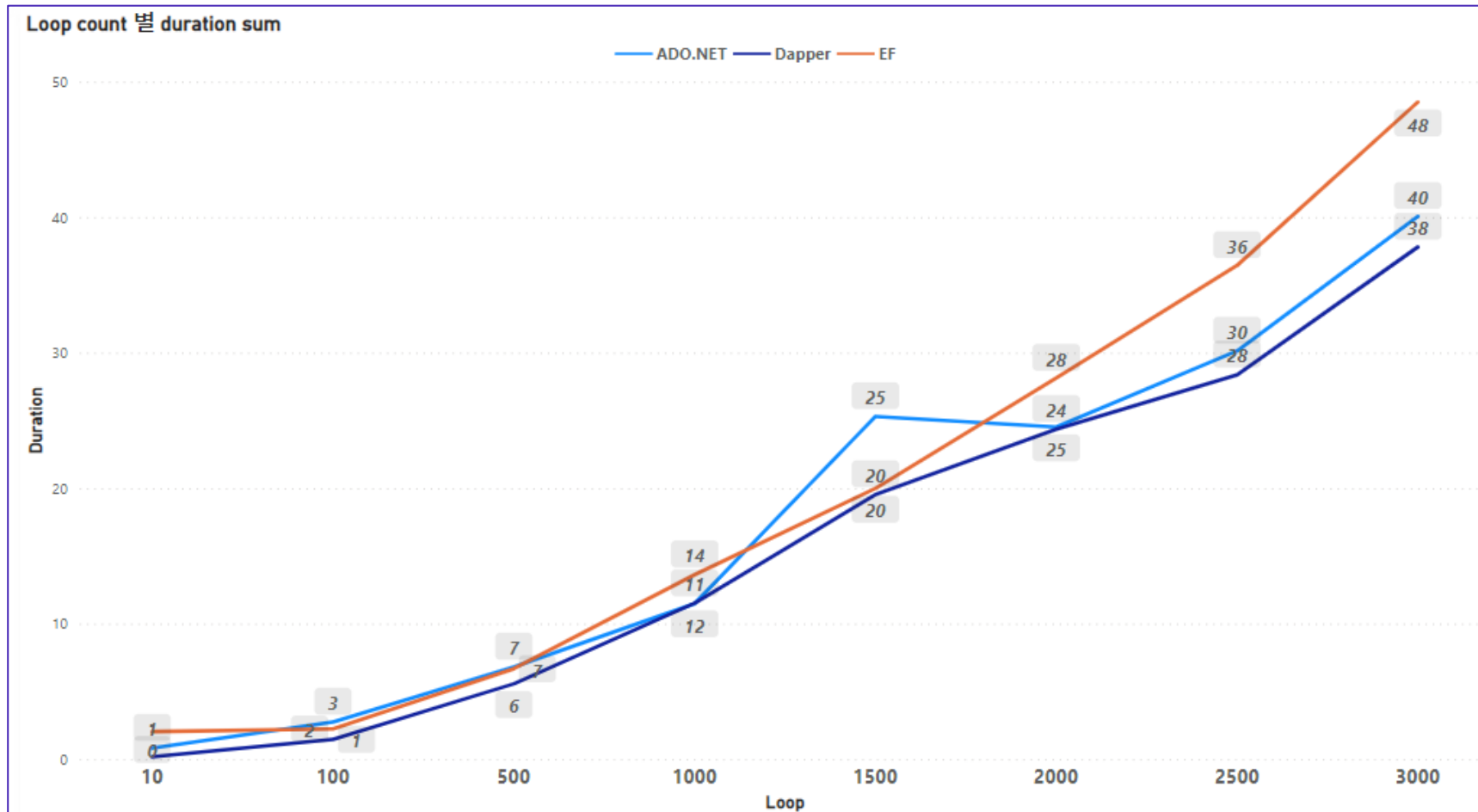
Dapper Select query

```
ADO.NET, 10, 0.6926023, 00:00:00.6, 10, 2022-01-16 오후 11:05:21
EF, 10, 1.9865301, 00:00:01.9, 10, 2022-01-16 오후 11:05:22
Dapper, 10, 0.1827715, 00:00:00.1, 10, 2022-01-16 오후 11:05:24
ADO.NET, 100, 1.7045672, 00:00:01.7, 100, 2022-01-16 오후 11:05:24
EF, 100, 2.1144004, 00:00:02.1, 100, 2022-01-16 오후 11:05:26
Dapper, 100, 1.4810985, 00:00:01.4, 100, 2022-01-16 오후 11:05:28
ADO.NET, 500, 6.2721453, 00:00:06.2, 500, 2022-01-16 오후 11:05:29
EF, 500, 7.2672999, 00:00:07.2, 500, 2022-01-16 오후 11:05:36
Dapper, 500, 6.4579796, 00:00:06.4, 500, 2022-01-16 오후 11:05:43
ADO.NET, 1000, 13.5290597, 00:00:13.5, 1000, 2022-01-16 오후 11:05:49
EF, 1000, 18.4576515, 00:00:18.4, 1000, 2022-01-16 오후 11:06:03
Dapper, 1000, 12.2564839, 00:00:12.2, 1000, 2022-01-16 오후 11:06:21
ADO.NET, 1500, 21.1391966, 00:00:21.1, 1500, 2022-01-16 오후 11:06:34
EF, 1500, 20.2704662, 00:00:20.2, 1500, 2022-01-16 오후 11:06:55
Dapper, 1500, 19.8557028, 00:00:19.8, 1500, 2022-01-16 오후 11:07:15
ADO.NET, 2000, 25.7960727, 00:00:25.7, 2000, 2022-01-16 오후 11:07:35
EF, 2000, 26.8694396, 00:00:26.8, 2000, 2022-01-16 오후 11:08:01
Dapper, 2000, 25.9672418, 00:00:25.9, 2000, 2022-01-16 오후 11:08:28
```

4. ORM 성능 테스트



4. ORM 성능 테스트



고맙습니다!

