Oracle Data Provider for .NET Entity Framework Core 3

Developers Guide 19c (3.19.0-beta1) Beta 1

Pre-General Availability: May 2020

Introduction

Oracle Data Provider for .NET (ODP.NET) Entity Framework (EF) Core is a database provider that supports Entity Framework Core applications for Oracle databases.

Entity Framework Core is a cross-platform Microsoft object-relational mapper that enables .NET developers to work with relational databases using .NET objects.

ODP.NET EF Core consists of a single 100% managed code dynamic-link library, Oracle.EntityFrameworkCore.dll, available via a NuGet package. It uses the Oracle.EntityFrameworkCore namespace.

This ODP.NET EF Core beta supports the newest Entity Framework Core version 3.

This beta documentation supplements existing ODP.NET EF Core documentation. It covers the changes made to EF Core since the last production release.

System Requirements

ODP.NET EF Core has similar system requirements as ODP.NET Core. ODP.NET EF Core requires the following:

- Operating System
    - Windows x64
        - Windows 10 x64 (Pro, Enterprise, and Education Editions)
        - Windows Server 2012 R2 x64 (Standard, Datacenter, Essentials, and Foundation Editions)
        - Windows Server 2016 x64 (Standard and Datacenter Editions)
        - Windows Server 2019 x64 (Standard and Datacenter Editions)
    - Linux x64
        - Oracle Linux 7 and 8
        - Red Hat Enterprise Linux 7 and 8
- .NET Core 2.1 and 3.1
    - .NET 5 is not yet supported
- Entity Framework Core 3.1
    - EF Core 5 is not yet supported

- Required .NET Assemblies
  - ODP.NET Core 19.7 or higher
  - Microsoft.EntityFrameworkCore.Relational
- Access to Oracle Database 11g Release 2 (11.2.0.4) or higher version

ODP.NET EF Core is compatible with ASP.NET Core and ASP.NET.

ODP.NET EF Core is built with AnyCPU. It supports 64-bit and 32-bit applications.

Application Programming Interfaces

ODP.NET EF Core supports standard EF Core application programming interfaces (APIs). The APIs below are supported in addition to the APIs in the latest ODP.NET EF Core provider.

Note: These APIs are available in earlier ODP.NET EF Core versions.

DatabaseFacade.IsOracle

Returns true if ODP.NET is the currently used database provider.

Declaration

// C#

public static bool IsOracle()

Return Value

A bool

Remarks

The provider is only known after the provider is set in the DbContext. Only use this method after that.

DbContextOptionsBuilder.UseOracle

This extension method sets the provider and database connection configuration to connect to Oracle Database. Developers can set any connection string attributes that are available in ODP.NET Core. Below are the available method overloads that can be called.

- UseOracle(string connectionString)
- UseOracle(string connectionString, Action<OracleDbContextOptionsBuilder> oracleOptionsAction = null)
- UseOracle(DbConnection connection, Action<OracleDbContextOptionsBuilder> oracleOptionsAction = null)
- DbContextOptionsBuilder<TContext> UseOracle<TContext>(string connectionString, Action<OracleDbContextOptionsBuilder> oracleOptionsAction = null)

- DbContextOptionsBuilder<TContext> UseOracle<TContext>(DbConnection connection,Action<OracleDbContextOptionsBuilder> oracleOptionsAction = null)

Sample

```
// C#

optionsBuilder.UseOracle(@"User Id=blog;Password=<password>;Data Source=pdborcl;");
```

Scaffolding

Below is the ODP.NET EF Core behavior when the "–Schema" and/or "–Table" parameter is specified while scaffolding a pre-existing model using the Package Manager Console command, Scaffold-DbContext. Similar functionality is available using the EF Core tools command, dotnet ef dbcontext scaffold.

	No Schema Filter	**Schema Filter**
No Table Filter	Generates all tables within current user/schema	All tables generated in specified schema
Table Filter	Generates specified tables within current user/schema	Specified tables generated in specified schema

Note: If creating tables in another schema, the user must have at least SELECT privileges for that other schema.

Limitations and Known Issues

Code First

- The HasIndex() fluent API cannot be invoked on an entity property that will result in a primary key in the Oracle database. Oracle Database does not support the creation of indexes for primary keys, since an index is implicitly created for all primary keys. **[Bug 28624509, Bug 28610258]**
- The 11.2 Oracle databases do not support default expression to reference any PL/SQL functions nor any pseudocolumns such as '*<sequence>*.NEXTVAL'. As such, HasDefaultValue() and HasDefaultValueSql() fluent APIs cannot be used in conjunction with 'sequence.NEXTVAL' as the default value, for example, if the Oracle database is 11.2. However, the application can use the UseOracleIdentityColumn() extension method to have the column be populated with server generated values, even if the Oracle database is 11.2. Please read about UseOracleIdentityColumn() for more details.

Scaffolding

- Scaffolding of a table that uses Function Based Indexes is now supported. However, the index will NOT be scaffolded. **[Bug 29782244]**

LINQ

- Using String.IsNullOrEmpty or !String.IsNullOrEmpty within the WHERE clause of a LINQ query is NOT supported. **[Enh 29798071]**
- Oracle Database 12.1 has the following limitation: if the select list contains columns with identical names and you specify the row limiting clause, then an ORA-00918 error occurs. This error occurs whether the identically named columns are in the same table or in different tables.

  Let us suppose that database contains following two table definitions:

  ```
  SQL> desc X;
   Name    Null?    Type
   ------- -------- ----------------------------
   COL1    NOT NULL NUMBER(10)
   COL2             NVARCHAR2(2000)

  SQL> desc Y;
   Name    Null?    Type
   ------- -------- ----------------------------
   COL0    NOT NULL NUMBER(10)
   COL1             NUMBER(10)
   COL3             NVARCHAR2(2000)
  ```

  Executing the following LINQ, for example, would generate a select query which would contain "COL1" column from both the tables. Hence, it would result in error ORA-00918:

  ```
  dbContext.Y.Include(a => a.X).Skip(2).Take(3).ToList();
  ```

  This error does not occur when using Oracle Database 11.2, 12.2, and higher versions. **[Bug 29199665]**

- Certain LINQ queries cannot be executed against Oracle Database 11.2.

  Let us first imagine an Entity Model with the following entities:

  ```
  public class Gear
  {
     public string FullName { get; set; }
     public virtual ICollection<Weapon> Weapons { get; set; }
  }

  public class Weapon
  {
     public int Id { get; set; }
     public bool IsAutomatic { get; set; }
     public string OwnerFullName { get; set; }
     public Gear Owner { get; set; }
  }
  ```

The following LINQ will not work against Oracle Database 11.2:

```
dbContext.Gear.Include(i => i.Weapons).OrderBy(o => o.Weapons.OrderBy(w =>
w.Id).FirstOrDefault().IsAutomatic).ToList();
```

This is due the LINQ query creating the following SQL query:

```
SELECT "i"."FullName"
FROM "Gear" "i"
ORDER BY (
   Select
    K0 "IsAutomatic" from(
   SELECT "w"."IsAutomatic" K0
   FROM "Weapon" "w"
   WHERE ("i"."FullName" = "w"."OwnerFullName")
   ORDER BY "w"."Id" NULLS FIRST
   ) "m1"
   where rownum <= 1
) NULLS FIRST, "i"."FullName" NULLS FIRST
```

Within the SELECT statement, there are two nested SELECTs. The generated SQL will encounter a ORA-00904 : "invalid identifier" error with database 11.2 since it has a restriction where it does not recognize outer select table alias "i" in the inner nested select query.
[Bug 29336890]

- LINQ query such as this is not supported with 11.2 database.

```
from c in ss.Set<Customer>()
from o in ss.Set<Order>().Where(o => c.CustomerID == o.CustomerID).Select(o => c.City)
select new { c, o }
```

This is because these LINQ queries result in CROSS APPLY to be used in the generated SQL query which is not supported in 11.2 DB.

The generated SQL query is:

```
SELECT "c"."CustomerID", "c"."Address", "c"."City", "c"."CompanyName",
"c"."ContactName", "c"."ContactTitle", "c"."Country", "c"."Fax", "c"."Phone",
"c"."PostalCode", "c"."Region", "t"."City" "o"
FROM "Customers" "c"
CROSS APPLY (
   SELECT "c"."City", "o"."OrderID", "o"."CustomerID"
   FROM "Orders" "o"
   WHERE ("c"."CustomerID" = "o"."CustomerID")
) "t"
```

[Bug 31188818]

- LINQ query having GroupBy with Conditional Aggregate not supported.

  Using Count() with predicate as highlighted below is NOT supported when used along with GroupBy clause.

  ```
  var query = (ss => ss.Set<Order>().GroupBy(o => o.CustomerID).Select(g =>
     g.Count(o => o.OrderID < 10300)));
  var result = query.ToArray();
  ```
  [Bug 31087576]

- A nested inner LINQ query referencing a column in a LINQ query two levels up is not supported.

  Let us first imagine an Entity Model with the following entities:

  ```
  public class Gear
  {
     public Gear() {   Weapons = new List<Weapon>();  }
     // composite key
     public string Nickname { get; set; }
     public int SquadId { get; set; }
     public string FullName { get; set; }
     public string CityOrBirthName { get; set; }
     public virtual City CityOfBirth { get; set; }
     public virtual City AssignedCity { get; set; }
     public MilitaryRank Rank { get; set; }
     public virtual CogTag Tag { get; set; }
     public virtual Squad Squad { get; set; }
     public virtual ICollection<Weapon> Weapons { get; set; }
     public string LeaderNickname { get; set; }
     public int LeaderSquadId { get; set; }
     public bool HasSoulPatch { get; set; }
     [NotMapped]
     public bool IsMarcus => Nickname == "Marcus";
  }

  public class Weapon
  {
     // auto generated key (sequence)
    public int Id { get; set; }
    public string Name { get; set; }
    public AmmunitionType? AmmunitionType { get; set; }
    public bool IsAutomatic { get; set; }
    // 1 - 1 self reference
    public int? SynergyWithId { get; set; }
    public virtual Weapon SynergyWith { get; set; }
    public string OwnerFullName { get; set; }
  ```

```
  public Gear Owner { get; set; }
}

public class Officer : Gear
{
  public Officer() {  Reports = new List<Gear>();   }
  // 1 - many self reference
  public virtual ICollection<Gear> Reports { get; set; }
}
```

Given the above entity classes, the following LINQ will not work against Oracle Database:

```
gs => from o in gs.OfType<Officer>()

select new

{
   o.FullName,
   OuterCollection = from r in o.Reports
        where r.FullName != "Foo"
        select new
        {
           r.FullName,
           InnerCollection = from w in r.Weapons where w.Name != "Bar"
              select new { w.Name, o.Nickname }
        }
 }
```

This is because the LINQ query creates the following SQL query:

```
SELECT "g"."FullName", "g"."Nickname", "g"."SquadId", "t0"."FullName", "t0"."Nickname",
"t0"."SquadId", "t0"."Name", "t0"."Nickname0", "t0"."Id"
FROM "Gears" "g"
OUTER APPLY (
   SELECT "g0"."FullName", "g0"."Nickname", "g0"."SquadId", "t"."Name", "t"."Nickname"
"Nickname0", "t"."Id"
   FROM "Gears" "g0"
   LEFT JOIN (
      SELECT "w"."Name", "g"."Nickname", "w"."Id", "w"."OwnerFullName"
      FROM "Weapons" "w"
      WHERE (("w"."Name" <> N'Bar') OR ("w"."Name" IS NULL))) "t" ON "g0"."FullName" =
"t"."OwnerFullName"
   WHERE (((("g0"."Discriminator" IN (N'Gear', N'Officer')) AND ("g0"."FullName" <> N'Foo')))
AND ((((("g"."Nickname" = "g0"."LeaderNickname") AND ("g0"."LeaderNickname" IS NOT
NULL))) AND ("g"."SquadId" = "g0"."LeaderSquadId"))))) "t0"
```

```
WHERE (("g"."Discriminator" IN (N'Gear', N'Officer')) AND ("g"."Discriminator" = N'Officer'))
ORDER BY "g"."Nickname" NULLS FIRST, "g"."SquadId" NULLS FIRST, "t0"."Nickname" NULLS
FIRST, "t0"."SquadId" NULLS FIRST, "t0"."Id" NULLS FIRST
```

The issue is because the SELECT query that is generated uses 'OUTER APPLY'. There is an issue in Oracle Database which causes it to return incorrect data when using 'OUTER APPLY'. **[Bug 31053669, Bug 30947031]**

Migrations

- If more than one column is associated with any Sequence/Trigger, then ValueGeneratedOnAdd() fluent API will be generated for each of these columns when performing a scaffolding operation. If we then use this scaffolded model to perform a migration, then an issue occurs because each of column that is associated with the ValueGeneratedOnAdd() fluent API is made an identity column by default. To avoid this issue, use UseOracleSQLCompatibility("11") which will force it to generate Triggers/Sequences instead. **[Bug 29467919]**

Sequences

- A sequence cannot be restarted.
- Extension methods related to SequenceHiLo is not supported, except for columns with Char, UInt, ULong, and UByte data types.

Computed Columns

- Literal values used for computed columns must be encapsulated by two single-quotes. In the example below, the literal string is the comma. It needs to be surrounded by two single-quotes as shown below.

```
 // C# - computed columns code sample
modelBuilder.Entity<Blog>()
.Property(b => b.BlogOwner)
.HasComputedColumnSql("\"LastName\" || '','' || \"FirstName\"");
```

Database Scalar Function Mapping

- Database scalar function mapping does not provide a native way to use functions residing within PL/SQL packages. To workaround this limitation, map the package and function to an Oracle synonym, then map the synonym to the EF Core function. **[Bug 29644406]**