Oracle Data Provider for .NET Entity Framework Core 5

Developers Guide 21c (5.21.1)

February 2021

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 version supports the newest Entity Framework Core 5.

This documentation supplements existing ODP.NET EF Core documentation. It covers the changes made to EF Core since the ODP.NET EF 3.19.80 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
    - Red Hat Enterprise Linux 7 and 8
- .NET 5 and .NET Core 3.1
- Entity Framework Core 5
- Required .NET Assemblies
  - ODP.NET Core 21.1 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.

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 5 application programming interfaces (APIs). Existing APIs are documented in the ODP.NET Developer's Guide. Notable additions in the newest Oracle EF Core provider are documented below.

DbContextOptionsBuilder.UseOracle Extension Method

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. In this new Oracle EF Core version, there is a new method overload available.

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

This extension configures the EF Core context to connect to an Oracle database without initially setting any DbConnection nor connection string. The DbConnection or connection string must be set before the DbContext attempts to connect to a database. To set the connection using, use RelationalDatabaseFacadeExtensions.SetDbConnection or RelationalDatabaseFacadeExtensions.SetConnectionString.

Declaration

public static DbContextOptionsBuilder UseOracle(this DbContextOptionsBuilder, Action<OracleDbContextOptionsBuilder>)

Parameters:

- DbContextOptionsBuilder - The builder being used to configure the context
- Action<OracleDbContextOptionsBuilder> - An optional action to allow additional Oracle specific configuration

Return Value

The options builder so that further configuration can be chained.

Sample Code

```
// C# - Setting up the DB context

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>
optionsBuilder.UseOracle();

// Using the DB context
```

```
using (var context = new DbContext())
{
        context.Database.SetDbConnection(new OracleConnection(<connection string>));
}
```

IQueryingEnumerable.ToQueryString Extension Method

A string representation of the Oracle SQL query used. This extension method will generate SQL that can be run in Oracle Database and Oracle Autonomous Database.

When UseOracleSQLCompatibility is set to “11”, additional comment lines will be generated, such as:

-- remove when executing from OracleCommand – Start

-- remove when executing from OracleCommand – End

If you execute the method’s generated SQL in an OracleCommand object, remove any code between the comment lines. For example, below is a code block that may be generated. Any code between these two lines should be deleted when executed in an OracleCommand. The lines can be removed programmatically. Here’s an example of what the generated ToQueryString code would look like:

```
-- remove when executing from OracleCommand - Start

-- if executing this script using OracleCommand, please remove lines of code as specified and explicitly bind :result_cursor from the user application.

variable result_cursor refcursor;

-- remove when executing from OracleCommand - End

DECLARE

…

BEGIN

…

END;

-- remove when executing from OracleCommand - Start

/

print result_cursor;

-- remove when executing from OracleCommand - End
```

To execute the generated SQL programmatically, developers can adapt the following C# pseudo-code for their specific requirements. The pseudo-code demonstrates how to generate the script using ToQueryString() on a sample LINQ query, and then how to execute the script with an OracleCommand, depending on the database version backing the application.

```
using System;
using System.Data;
using System.Linq;
using System.Text.RegularExpressions;
using Microsoft.EntityFrameworkCore;
using Oracle.ManagedDataAccess.Client;
using Oracle.ManagedDataAccess.Types;

class ToQueryStringPseudoCode
{
   static void Main(string[] args)
   {
      const string START_TAG = "-- remove when executing from OracleCommand - Start";
      const string END_TAG = "-- remove when executing from OracleCommand - End";

      using (ModelContext db = new ModelContext())
      {
         //sample LINQ to convert query string from
         string name = "Name";
         var query = db.Set<Instructor>().Where(c => c.Name == name);
         string sqltext = query.ToQueryString();

         //processing the generated script and removing the code between the comment tags if
present.
         string newsqltext = TrimBetweenTags(START_TAG, END_TAG, sqltext);

         //'newsqltext' can be used directly with OracleCommand.
         OracleConnection con = new OracleConnection("<Connection String>")
         con.Open();
```

```
        OracleCommand cmd = con.CreateCommand();
        cmd.CommandText = newsqltext;
        OracleDataReader reader;

        //in case of DB 11.2, explicitly bind the parameter 'result_cursor' to the OracleCommand.
        if (<Database Version is 11.2>)
        {
           cmd.BindByName = true;
           OracleParameter outRefPrm = cmd.Parameters.Add("result_cursor", OracleDbType.RefCursor,
DBNull.Value, ParameterDirection.Output);
           cmd.ExecuteNonQuery();
           reader = ((OracleRefCursor)outRefPrm.Value).GetDataReader();
        }
          // No binding required for 12c+ DBs.
        else
        {
           reader = cmd.ExecuteReader();
        }

        //verifying the result set.
        while (reader.Read())
        {
           Console.WriteLine($"{reader[0]}, {reader[1]}, {reader[2]}, {reader[3]}");
        }
        con.Close();
     }
  }
```

```
    public static string TrimBetweenTags(string startTag, string endTag, string str)
    {
        if (String.IsNullOrEmpty(str))
            return null;

        Regex x = new Regex("(" + startTag + ")([\\s\\S]*?)(" + endTag + ")");
        return x.Replace(str, "");
    }
}
```

MigrationBuilder.IsOracle Extension Method

Returns true if the MigrationBuilder object uses ODP.NET as its database provider.

Declaration

public static bool IsOracle(this MigrationBuilder)

Parameters:

- MigrationBuilder object

Return Value

A bool

Sample Code

var migrationBuilder = new MigrationBuilder("Oracle.EntityFrameworkCore");

bool b_oracle = migrationBuilder.IsOracle(); //returns true for ODP.NET

Desupported ModelBuilder.UseOracleIdentityColumn Extension Method

Starting with with EF Core 5, ODP.NET EF Core desupports UseOracleIdentityColumn. Developers should call the ModelBuilder.UseIdentityColumn extension method instead, which provides identical functionality.

UseOracleIdentityColumn remains supported for EF Core 3.1 and earlier releases.

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 UseIdentityColumn() extension method to have the column be populated with server generated values, even if the Oracle database is 11.2. Please read about UseIdentityColumn() 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 such as this which uses Take() is not supported.

```
from c in ss.Set<Customer>().Include(c => c.Orders)
from o in ss.Set<Order>().Where(o => o.CustomerID == c.CustomerID).OrderBy(o =>
c.CustomerID).Take(5)
where c.CustomerID.StartsWith("F")
select c
```

The generated SQL query which returns incorrect results 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"."OrderID", "o0"."OrderID",
"o0"."CustomerID", "o0"."EmployeeID", "o0"."OrderDate"
FROM "Customers" "c"
CROSS APPLY (
   SELECT "o"."OrderID"
   FROM "Orders" "o"
   WHERE "c"."CustomerID" = "o"."CustomerID"
   ORDER BY "c"."CustomerID"
   FETCH FIRST 5 ROWS ONLY
   ) "t"
   LEFT JOIN "Orders" "o0" ON "c"."CustomerID" = "o0"."CustomerID"
   WHERE "c"."CustomerID" LIKE 'F%'
   ORDER BY "c"."CustomerID", "t"."OrderID", "o0"."OrderID"
```

[Bug 32050117]

- LINQ query such as this which uses OrderBy() is not supported.

```
ss.Set<Customer>()
                .OrderBy(c => c.CustomerID)
```

```
                .SelectMany(c => c.Orders.OrderBy(o =>
o.OrderID).Skip(0).Select(o => new { c.City, o.OrderDate }))
```

The generated SQL query which returns results in incorrect order is:

```
SELECT "t"."City", "t"."OrderDate"
FROM "Customers" "c"
CROSS APPLY (
   SELECT "c"."City", "o"."OrderDate"
   FROM "Orders" "o"
   WHERE "c"."CustomerID" = "o"."CustomerID"
   ORDER BY "o"."OrderID"
   OFFSET 0 ROWS
   ) "t"
   ORDER BY "c"."CustomerID"
```

[Bug 32096273]

- LINQ query such as this which uses Take() is not supported.

```
from c in ss.Set<Customer>().OrderBy(c =>
c.CustomerID).Take(3)
            select ss.Set<Order>().OrderBy(o => o.OrderID).ThenBy(o
=> c.CustomerID).Skip(100).Take(2).ToList()
```

The generated SQL query which returns incorrect results is:

```
SELECT "t"."CustomerID", "t0"."OrderID", "t0"."CustomerID",
"t0"."EmployeeID", "t0"."OrderDate"
FROM (
   SELECT "c"."CustomerID"
   FROM "Customers" "c"
   ORDER BY "c"."CustomerID"
   FETCH FIRST :p_0 ROWS ONLY
   ) "t"
   OUTER APPLY (
      SELECT "o"."OrderID", "o"."CustomerID", "o"."EmployeeID",
"o"."OrderDate", "t"."CustomerID" "CustomerID0"
      FROM "Orders" "o"
      ORDER BY "o"."OrderID", "t"."CustomerID"
      OFFSET 100 ROWS FETCH NEXT 2 ROWS ONLY
      ) "t0"
      ORDER BY "t"."CustomerID", "t0"."OrderID", "t0"."CustomerID0"
```

[Bug 32096277]

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]**