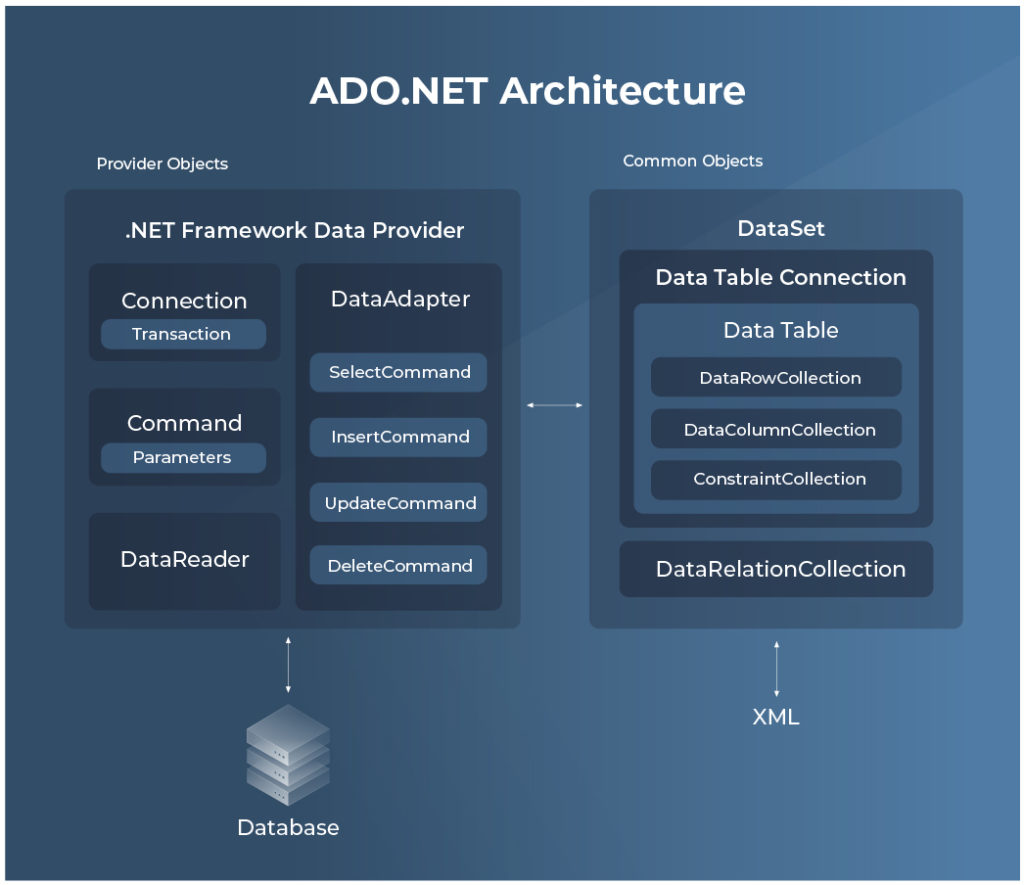
[https://blog.devart.com/ado-net-vs-entity-framework.html#:~:text=Because%20ADO.NET%20always%20establishes,query%20to%20perform%20database%20operations](https://blog.devart.com/ado-net-vs-entity-framework.html" \l ":~:text=Because%20ADO.NET%20always%20establishes,query%20to%20perform%20database%20operations).

## ****ADO.NET Architecture Concept****

Microsoft designed ADO.NET in such a way that we can perform different kinds of data source operations in the same fashion. For simplicity, we can categorize ADO.NET components into three categories: disconnect, common or shared, and the .NET data providers. The disconnected components build based on ADO.NET architecture. We can use these classes with or without data providers. For example, we can use a DataTable object with or without providers, and shared or common components are the base classes for all types of data providers. The below ADO.NET architecture diagram demonstrates related to the ADO.NET Component model and how they work together.



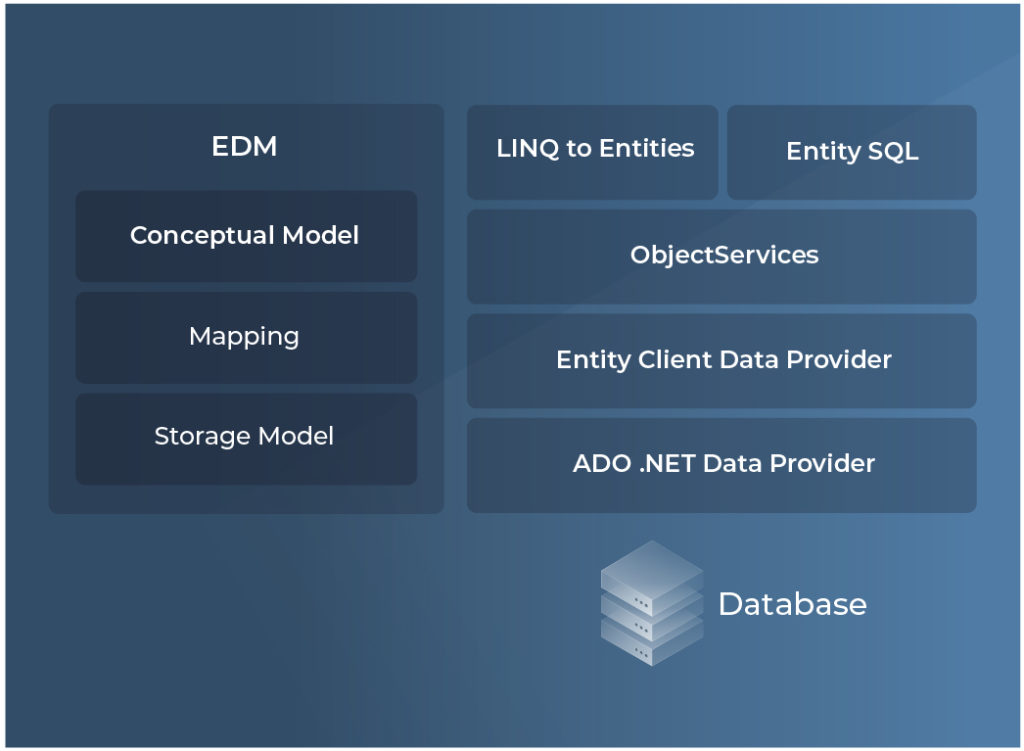
A data provider is a set of factors, similar as Connection, Command, DataAdapter, and DataReader. The **Connection**is the first element that talks to a data source. With the help of **Connection Object**, we can establish a connection between the application and the data source. These connection objects work as reference objects in the Command and DataAdapter objects. A **Command object**executes a SQL query and stored procedures to read, add, update, and cancel data of a data source via a DataAdapter. A **DataAdapter** is ground between a dataset and the connection. We can use the Command Object to execute any type of SQL Queries and Stored Procedures to fetch data from the database.

All data providers share the ADO.NET common components. These components like DataSet, DataView, and DataViewManager always represent the data on behalf of ADO.NET. The DataSet component objects normally use XML Schema to capture and return data between the Applications and the Data Providers. A DataSet is a sub-set of DataTable objects. A DataTable represents a database table. We can represent single or multiple views of a dataset with the help of DataView and DataViewManager objects. In our applications, if required we can directly use a DataView or DataViewManager component with data-bound controls like DataGrid or DataList.

## ****Entity Framework Architecture****

Let’s give a short overview of the different components available under the Entity Framework Architecture:

* **EDM (Entity Data Model):**Entity Data Model or EDM are divided into three main parts – Conceptual model, Mapping and Storage model.
  + **Conceptual Model:**In Conceptual Model, we normally define the model classes and establish the relations between them. This class-based relationship is independent of the database point of view.
  + **Storage Model:**The storage model mainly contains all the database-related design details like tables, views, stored procedures, and the relationships between different tables objects and their relationship keys. Please, read about [table per type vs table per hierarchy](https://blog.devart.com/table-per-type-vs-table-per-hierarchy-inheritance.html), if you want to know more about it.
  + **Mapping:**Mapping components mainly contain the information related to the mapping between the conceptual model and the storage model.
* **LINQ to Entities:** LINQ-to-Entities (L2E) is a query language. This query language is used to write queries against the object model in the Entity Framework layer. These queries return entities as an output where entities are used mainly from the conceptual model.
* **Entity SQL:**Entity SQL is another query language, which only available for Entity Framework v1 – v6 . This language tool is quite similar to LINQ to Entities. However, as per the operation point of view, it is a little more complex compared to the L2E.
* **Object Service:**In Entity Framework Architecture Layer, Object service is the main entry point for fetching data from the database layer. The object service layer is also responsible for the materialization of the data.
* **Entity Client Data Provider:**The main Objective of the Entity Client Data Provider layer is to convert LINQ-to-Entities or Entity SQL queries into a SQL query which is normally understood by any type of database. It always communicates with the ADO.Net data provider to return data from the database.
* **ADO.Net Data Provider:**This layer communicates with the database using the standard ADO.NET mechanism.



**What is the Difference between ADO.NET and Entity Framework**

In this chapter, we are going to explore the difference between traditional ADO.NET and Entity Framework. Although both the ADO.Net and Entity Framework are defined on the basic standard of ActiveX Data Objects to connect relational and non-relational database systems. But despite that, these two have many differences compared to each other. Some of the key differences between ADO.NET and the Entity Framework are as below:

* **Performance**: ADO.NET is much faster compared to the Entity Framework. Because ADO.NET always establishes the connection directly to the database. That’s why it provides much better performance compared to the Entity Framework. It is because, in Entity Framework, the entity first translates the LINQ queries to SQL and then it processes the query to perform database operations. Also, the Entity Framework consists of a wrapper class for the ADO.Net. Due to this wrapper, the developer can perform the coding much faster in Entity Framework.
* **Flexibility**: In the case of executing SQL Queries and stored procedures, ADO.NET always provides us much more flexibility and control compared to the Entity Framework. ADO.NET always provides full control over the database we use in applications. The LINQ queries are always resulting in an efficient SQL query because raw SQL queries are always useful. We also need to write down the raw SQL queries whenever we can’t express the query using LINQ.
* **Speed of Development**: With the help of ADO.NET, we can maintain complete control over the data layer of our applications. We can create the classes and methods from scratch which will help us to establish the communication between the database and the application. It always take some more time and effort to develop the data access layer compared to the Entity Framework. In the case of Entity Framework, data access layer creation is much easier. Because Entity Framework generates the model class and database context classes automatically. In this way, it automatically handles the operation of the database.
* **Code Maintainability**: Code can be maintained in a better way in Entity Framework. Because, when trying to debug the data access layer in ADO.NET, we can’t find the proper relationship between the model classes. But in Entity Framework, it was maintained, and also we found clear dependency on the storage model with the help of the mapping process.

Both ADO.NET and Entity Frameworks have similar and quite different features. To make the process of comparison easier and to answer numerous questions about them (e.g. “does Entity Framework use ADO.NET?” etc.), we offer a clear comparison table.

