



# Mục tiêu

---

- Ưu điểm của ADO.NET
- Kiến trúc của ADO.NET
- Kết nối DB với ADO.NET

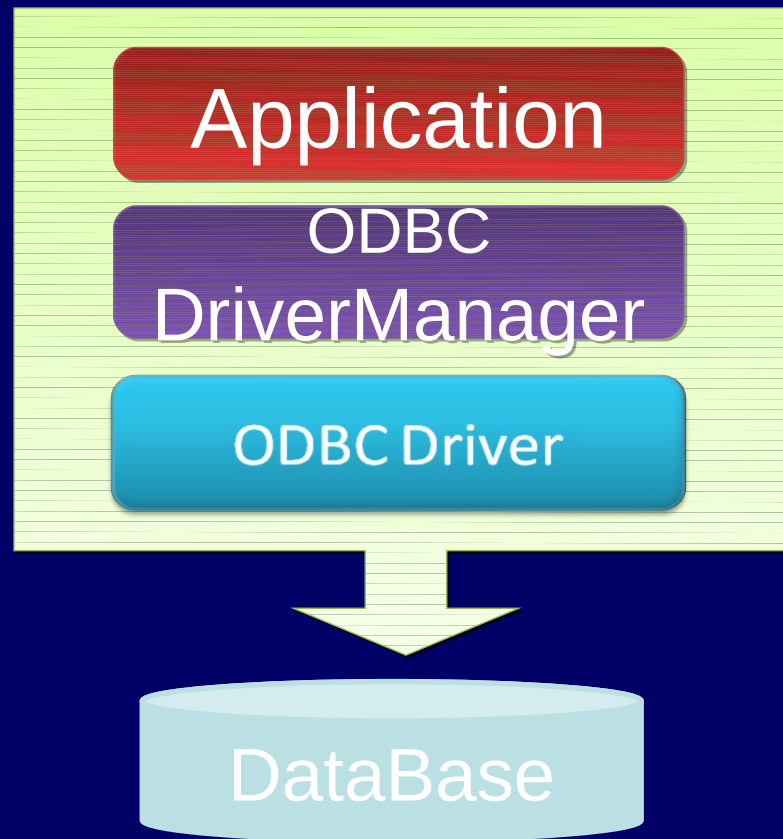


# Xây dựng App tương tác DB

---

- Khi cần xây dựng một ứng dụng cần tương tác với cơ sở dữ liệu. Có nhiều công nghệ cung cấp cho người dùng các tập hợp các đối tượng để thực hiện.

# ODBC



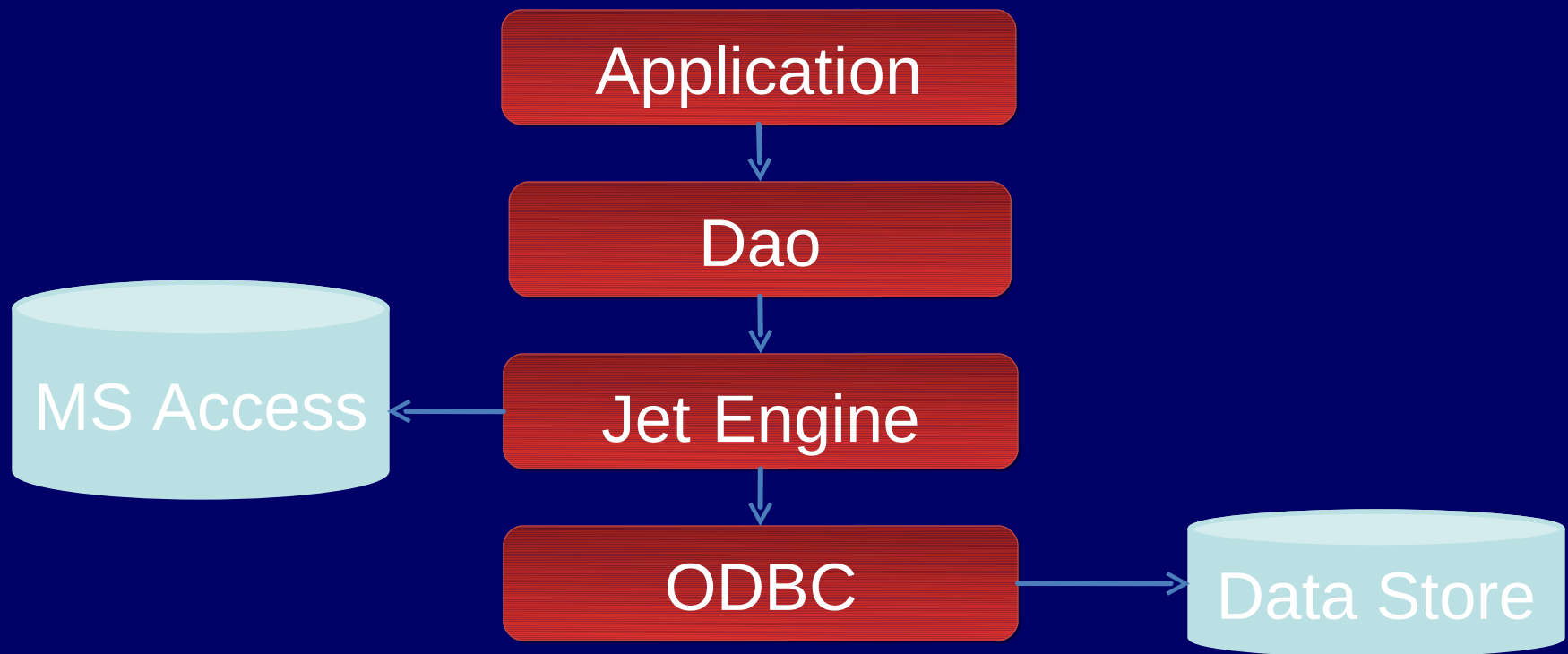


# ODBC

---

- Open Data Connectivity
- Chỉ truy suất được các thông tin quan hệ không truy suất được các dữ liệu không quan hệ như: tập tin văn bản Email.
- Phải truy nhập ODBC qua DNS

# DAO



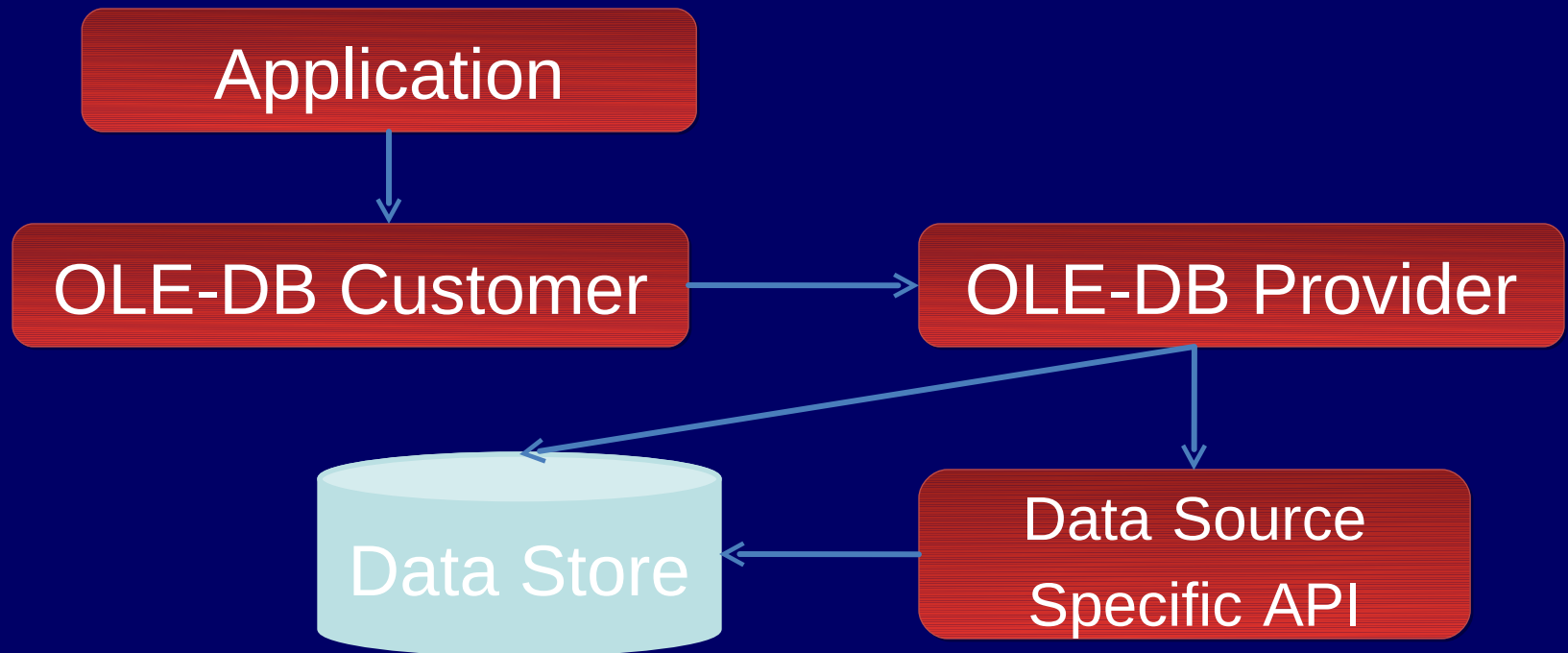


# DAO

---

- Data Access Objects
- Chỉ làm việc với Microsoft Jet Database Engine.
- Cũng có nhiều khuyết điểm như ODBC.

# OLEDB





# OLEDB

---

- Để truy cập được tất cả datastore, phải dùng OLEDB provider thông qua ODBC
- OLEDB dễ sử dụng hơn ODBC
- Là một tập hợp các giao diện COM được đóng gói thành các các System Service để tương tác với nhiều DBMS





# OleDb

---

- Datasource
- Session
- Command
- Rowset



# ADO

---

- ADO là một COM
- Được dùng với bất kỳ ngôn ngữ nào tương thích với COM.
- ADO không độc lập OS nhưng độc lập ngôn ngữ.



# ADO

---

- Connection
- Command
- Recordset



# ADO

---

- RDS của MS cho phép dùng ADO thông qua các giao thức HTTP, HTTPS, DCOM để truy cập dữ liệu Web.

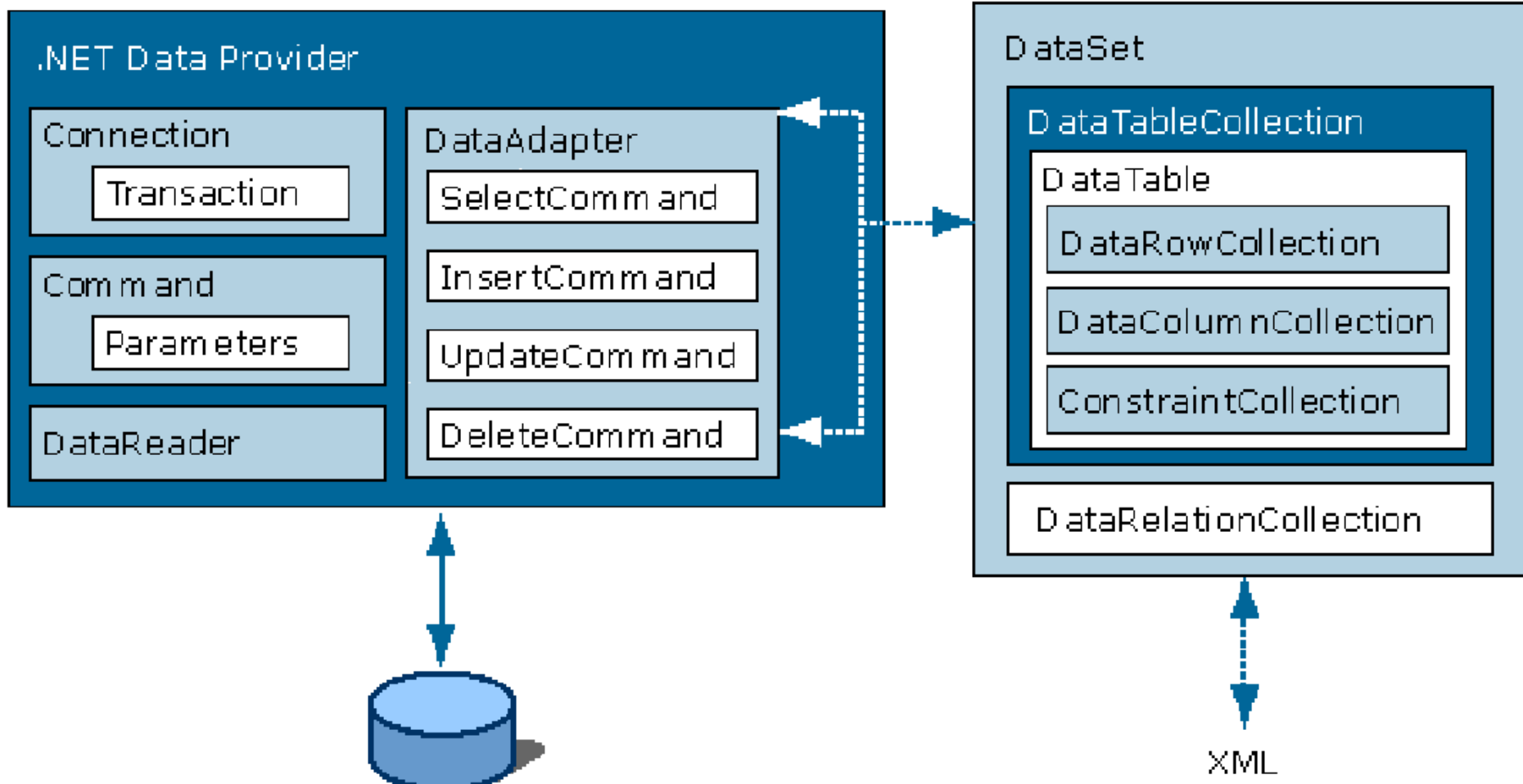


# MDAC

---

- Microsoft Data Access Components
  - ODBC
  - OLEDB
  - ADO
  - RDS

# ADO.NET



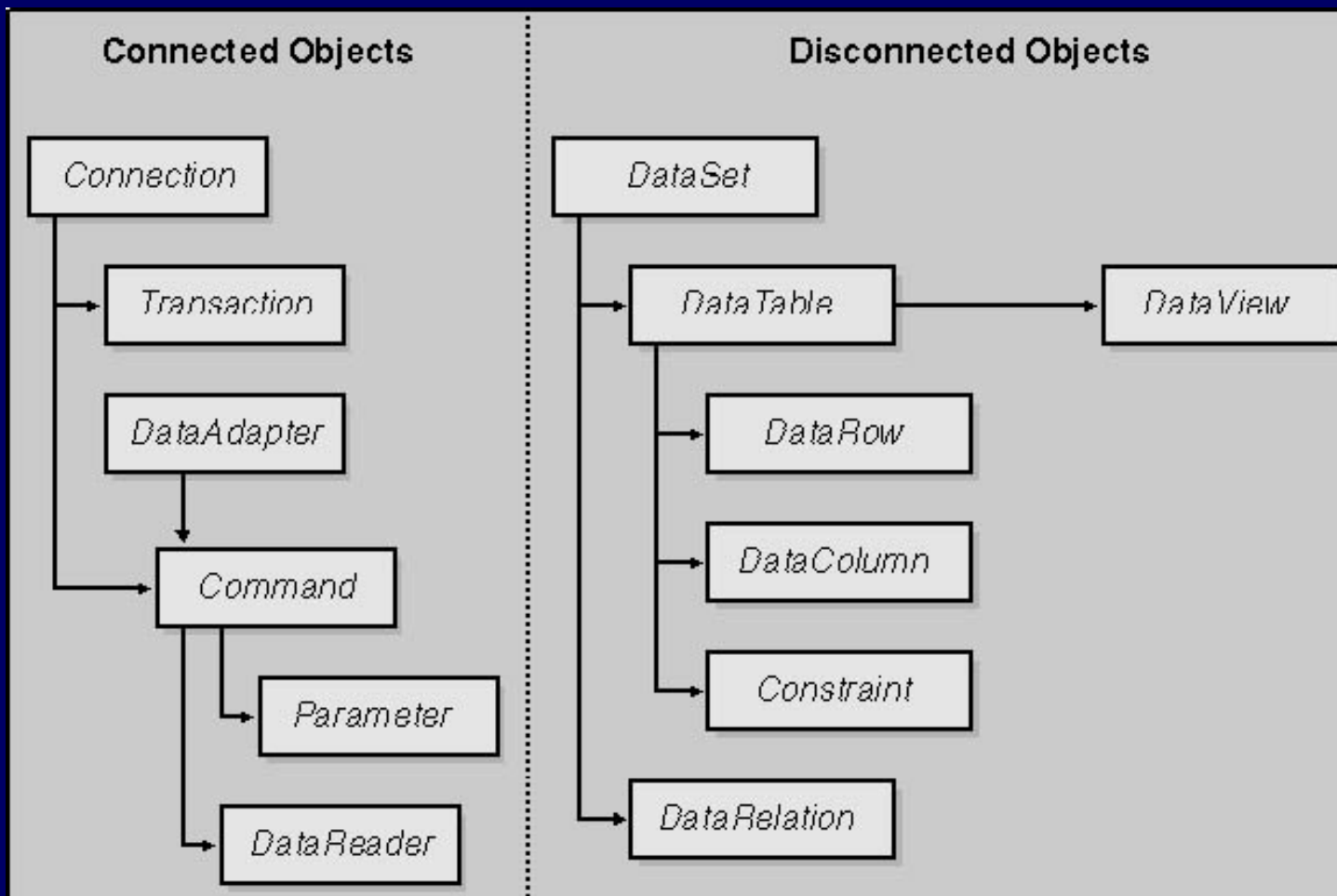


## Ưu điểm của ADO.NET

---

- Đáp ứng mô hình lập trình mới trên nền .NET.
- Hỗ trợ rất tốt SQL Server.
- Hỗ trợ thao tác các CSDL khác thông qua OLE DB.
- XML làm nền tảng.
- Hỗ trợ kiến trúc 3 lớp.
- Sử dụng namespace: System.Data, System.Xml

# Kiến trúc ADO.NET







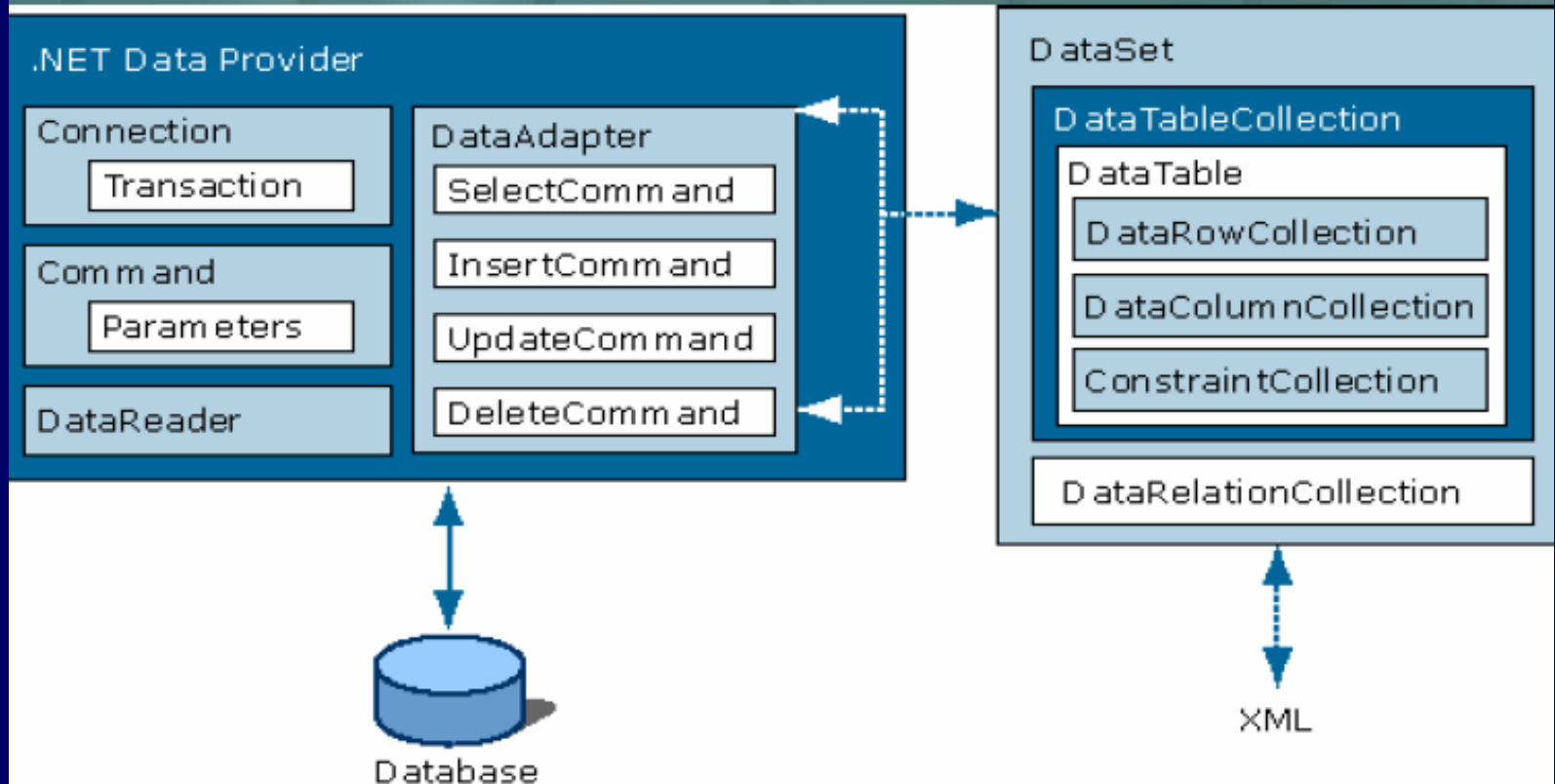
# Kiến trúc ADO.NET

---

- **Connected objects:** là những đối tượng giao tiếp trực tiếp với CSDL.
- **Disconnected objects:** cho phép các user làm việc với dữ liệu dạng offline (khi đã đóng kết nối cơ sở dữ liệu)

# Kiến trúc ADO.NET

## Kiến trúc





# The Managed Provider and Generic Data Set Classes

---

To provide both connected and disconnected DB access, ADO.NET defines two sets of classes: managed provider and generic data.



# The Managed Provider Classes

---

- **SQL Server Managed Provider Classes.**
- **OLE DB Managed Provider Classes.**
- **ODBC Managed Provider Classes.**



# SQL Server Managed Provider Classes

---

- You use the SQL Server managed provider classes to connect to a SQL Server database



# OLE DB Managed Provider Classes

---

- You use the OLE DB (Object Linking and Embedding for Databases) managed provider classes to connect to any database that supports OLE DB, such as Access or Oracle.



# ODBC Managed Provider Classes

---

- You use the ODBC (Open Database Connectivity) managed provider classes to connect to any database that supports ODBC. All the major databases support ODBC, but ODBC is typically slower than the previous two sets of classes when working with .NET.



# The Generic Data Classes

---

- The Connection Classes
- The Command Classes
- The Parameter Classes
- The ParameterCollection Classes
- The DataReader Classes
- The DataAdapter Classes
- The CommandBuilder Classes
- The Transaction Classes





## Performing a SQL *SELECT* Statement and Storing the Rows Locally

---

1. Formulate a string containing the details of the database connection.
2. Create a SqlConnection object to connect to the database, passing the connection string to the constructor.
3. Formulate a string containing a SELECT statement to retrieve the columns for the rows from the Customers table.
4. Create a SqlCommand object to hold the SELECT statement.



## Performing a SQL *SELECT* Statement and Storing the Rows Locally

---

5. Set the CommandText property of the SqlCommand object to the SELECT string.
6. Create a SqlDataAdapter object.
7. Set the SelectCommand property of the SqlDataAdapter object to the SqlCommand object.
8. Create a DataSet object to store the results of the SELECT statement.



## Performing a SQL *SELECT* Statement and Storing the Rows Locally

---

9. Open the database connection using the Open() method of the SqlConnection object.
10. Call the Fill() method of the SqlDataAdapter object to retrieve the rows from the table, storing the rows locally in a DataTable of the DataSet object.
11. Close the database connection, using the Close() method of the SqlConnection object created in step 1.
12. Get the DataTable object from the DataSet object.
13. Display the columns for each row in the DataTable, using a DataRow object to access each row in the DataTable.



## Step 1: Formulate a String Containing the Details of the Database Connection

---

- When connecting to a SQL Server database, your string must specify the following:
  - The name of the computer on which SQL Server is running.
  - The name of the database.
  - The name of the user to connect to the database as. You set this in the uid part of the string.
  - The password for the database user. You set this in the pwd part of the string.



## EX:

---

- `string connectionString = "server=localhost;database=Northwind;uid=sa;pwd=sa";`
- `String connectionstring = "provider=Microsoft.Jet.OLEDB.4.0;" + "data source=E:\\My Course\\2009\\Lap trinh Quan Ly II\\Thuc Hanh\\ado.net\\database\\HocSinh.mdb"`



## Step 2: Create a *SqlConnection* Object to Connect to the Database

---

- Create a *SqlConnection* object to connect to the database, passing the connection string created in the previous step to the constructor. You use an object of the *SqlConnection* class to connect to a SQL Server database.

- EX:

```
SqlConnection mySqlConnection  
= new SqlConnection(connectionString);
```



## Step 3: Formulate a String Containing the *SELECT* Statement

---

- EX: Select \* from HocSinh



## Step 4: Create a *SqlCommand* Object to Hold the *SELECT* Statement

- You can call the `CreateCommand()` method of `mySqlConnection` to create a new `SqlCommand` object for that connection. The `CreateCommand()` method returns a new `SqlCommand` object for the `SqlConnection` object.

- EX:

```
SqlCommand mySqlCommand =  
mySqlConnection.CreateCommand();
```





## Step 5: Set the *CommandText* Property of the *SqlCommand* Object to the *SELECT* String

---

- Set the *CommandText* property of your *SqlCommand* object to the *SELECT* string created in step 4. The *CommandText* property contains the SQL statement you want to perform.
- EX:

*mySqlCommand.CommandText = selectString;*



## Step 6: Create a *SqlDataAdapter* Object

---

- You use a *SqlDataAdapter* object to move information between your *DataSet* object and the database. You'll see how to create a *DataSet* object in step 8.
- EX:
- *SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();*



## Step 7: Set the *SelectCommand* Property of the *SqlAdapter* Object to the *SqlCommand* Object

---

- The *SelectCommand* property contains the **SELECT** statement you want to run.
- EX:

```
mySqlDataAdapter.SelectCommand =  
mySqlCommand;
```



## Step 8: Create a *DataSet* Object to Store the Results of the *SELECT* Statement

---

- You use a *DataSet* object to store a local copy of information retrieved from the database.
- EX:

*DataSet myDataSet = new DataSet();*



## Step 9: Open the Database Connection Using the *Open()* Method of the *SqlConnection* Object

- EX: `mySqlConnection.Open();`



## Step 10:

---

- Call the *Fill()* Method of the *SqlDataAdapter* Object to Retrieve the Rows From the Table.
- EX:  
*mySqlDataAdapter.Fill(myDataSet, "Customers");*



## Step 11: Close the Database Connection

---

- Close the database connection using the Close() method of the SqlConnection object created in the first step.
- For example: *mySqlConnection.Close();*

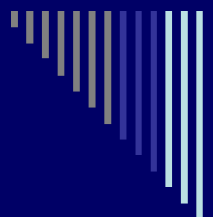


## Step 12: Get the *DataTable* Object From the *DataSet* Object

---

- You get a *DataTable* from your *DataSet* using the *Tables* property, which returns a *DataTableCollection* object. To get an individual *DataTable* from your *DataSet*.
- *EX: DataTable myDataTable = myDataSet.Tables["Customers"];*
- *Note: You can also specify the DataTable you want to get by passing a numeric value to the Tables property. For example, myDataSet.Tables[0] also returns the Customers DataTable.*





## Step 13: Display the Columns for Each Row in the *DataTable*

- Display the columns for each row in the DataTable, using a DataRow object to access each row in the DataTable. The DataTable class defines a property named Rows that returns a DataRowCollection object containing the DataRow objects stored in that DataTable.
- EX:

```
foreach (DataRow myDataRow in myDataTable.Rows)
```

```
{ // ... access the myDataRow object }
```

```
foreach (DataRow myDataRow in myDataTable.Rows)
```

```
{ Console.WriteLine("CustomerID = "+ myDataRow["CustomerID"]);
```

```
Console.WriteLine("CompanyName = "+ myDataRow["CompanyName"]);
```

```
Console.WriteLine("ContactName = "+ myDataRow["ContactName"]);
```

```
Console.WriteLine("Address = "+ myDataRow["Address"]); }
```



# EX Full

---

- Demo. Connect Step For Step