C# WINFORMS: DBASE Programming

ADO.NET

It stands for ActiveX Data Object with the .NET technology

Data access technology included in the .NET framework

It consists of managed classes that allow .NET applications to connect to data sources (usually relational databases), execute commands, and manage disconnected data.

It is an improved version of ADO.

ADO.NET Data Providers

A *data provider* is a set of ADO.NET classes that allows you to access a specific database, execute SQL commands, and retrieve data.

It is a bridge between your application and a data source.

ADO.NET Data Providers

MySql provider: Provides access to MySql database

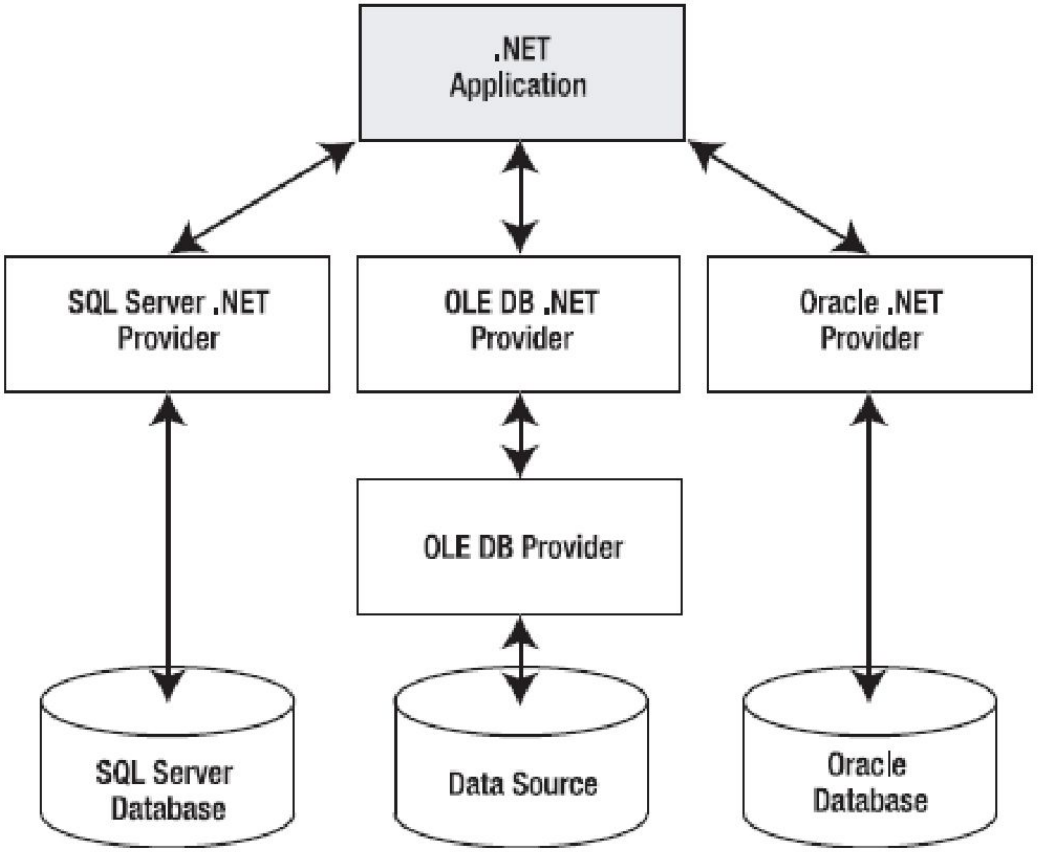
SQL Server provider: Provides optimized access to a SQL Server database (version 7.0 or later).

OLE DB provider: Provides access to any data source that has an OLE DB driver. This includes SQL Server databases prior to version 7.0.

Oracle provider: Provides optimized access to an Oracle database (version 8*i* or later).

ODBC provider: Provides access to any data source that has an ODBC driver.

Database Connections with WinForms C#



Namespaces

MySql.Data.MySqlClient

- Contains the key data container classes that model columns, relations, tables, datasets, rows, views, and constraints. In addition, contains the key interfaces that are implemented by the connection-based data objects.

Namespaces

System.Data.OleDb

- Contains the classes used to connect to an OLE DB provider, including OleDbCommand, OleDbConnection, and OleDbDataAdapter. These classes support most OLE DB providers but not those that require OLE DB version 2.5 interfaces.

System.Data.SqlClient

- Contains the classes you use to connect to a Microsoft SQL Server database, including SqlDbCommand, SqlDbConnection, and SqlDBDataAdapter. These classes are optimized to use the TDS interface to SQL Server.

Classes in the 3 Common Namespaces

MySql.Data.MySqlClient	System.Data.OleDB	System.Data.SqlClient
MySqlConnection()	OleDbConnection()	SqlConnection()
MySqlCommand()	OleDbCommand()	SqlCommand()
MySqlDataReader()	OleDbDataReader()	SqlDataReader()
MySqlDataAdapter()	OleDbDataAdapter()	SqlDataAdapter()
Dataset	Dataset	Dataset
DataTable	DataTable	DataTable

Data Provider Classes

Connection: Used in establishing a connection to a data source.

Command: This object is used to execute SQL commands and stored procedures.

DataReader. It provides efficient way to access data. Sequentially reads data from data source.

DataAdapter: Functions as a bridge between a data source and a disconnected data class (DataSet).

- **DataSet:** It is an all-purpose container for data that you've retrieved from one or more tables in a data source.
- **DataTable:** Represents relational data into tabular form.

Connection Class

It is used to enable communication between application and database.

Properties

- **ConnectionString** - contains a String that the connection class uses to establish the database connection

Methods

- **Close() -** closes an open connection
- **Open() -** reads the contents of the ConnectionString property, sends a request to a provider, and then opens a connection

Creating a Connection

Connect to MySQL database

- **con.ConnectionString = "server=localhost;uid=root;database=dict";**

Connect to a database in SQL Server using SQL Aunthetication

- con.ConnectionString = "Data Source=ADMIN-PC\SQLEXPRESS;Initial Catalog=sis;Persist Security Info=True;User ID=sa;Password=michael“

Connect to a database in SQL Server using Windows Aunthetication

- con.ConnectionString = "Data Source=ADMIN-PC\SQLEXPRESS;Initial Catalog=sis; Integrated Security=SSPI“

Creating a connection

Connect to the database named C:\Users\Admin\Documents\sis.mdb in Access 2003

- con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Users\Admin\Documents\sis.mdb;“

Connect to the database named C:\Users\Admin\Documents\sis.accdb in Access 2007

- con.ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Admin\Documents\sis.accdb;“

Connect to a database using a MDF file

- con.ConnectionString = "Data Source=room\SQLEXPRESS;AttachDbFilename=E:\sample.mdf;

  Integrated Security=True;Connect Timeout=30;User Instance=True"

Command Class

Stores SQL statements which are used by DataAdapter to select and update rows. It executes command automatically.

Method

- **ExecuteNonQuery() -** executes a SQL statement. It does not return database rows
- **ExecuteReader()** – It returns a set of rows from the database. Forward only retrieval of records and it is used to read table values from first to last.

DataAdapter Class

It works in conjunction with the DataSet class to read and write data

Methods

- **Fill() -** uses a connection to retrieve database records. Records are loaded into the DataSet.

Example

MySqlDataAdapter adapter;

- DataSet customerDataSet = new DataSet();
  adapter = New MySqlDataAdapter("select * from employee", con);
  // Fill the DataSet with data from the Customers table
  adapter.Fill(customerDataSet, "Customers");

DataSet Class

It represents a disconnected view of the data stored in one or more database tables

Method

- **Clear() -** removes all rows from all tables in the DataSet

Example

```
MySqlDataAdapter adapter;
DataSet customerDataSet = new DataSet();
adapter = New MySqlDataAdapter("select * from employee", con);
 // Fill the DataSet with data from the Customers table
 adapter.Fill(customerDataSet, "Customers");
```

DataTable Class

A database table representation and provides a collection of columns and rows to store data in a grid form.

It is a fundamental component of the ADO.NET framework. It represents an in-memory, tabular structure of data, similar to a table in a relational database, consisting of rows and columns. This class is particularly useful for managing and manipulating disconnected data within an application.

```
DataTable myTable = new DataTable("Employees");
```

DataGrid

a powerful and versatile tool for displaying and manipulating tabular data. It provides a highly customizable grid interface that can be bound to various data sources or populated manually.

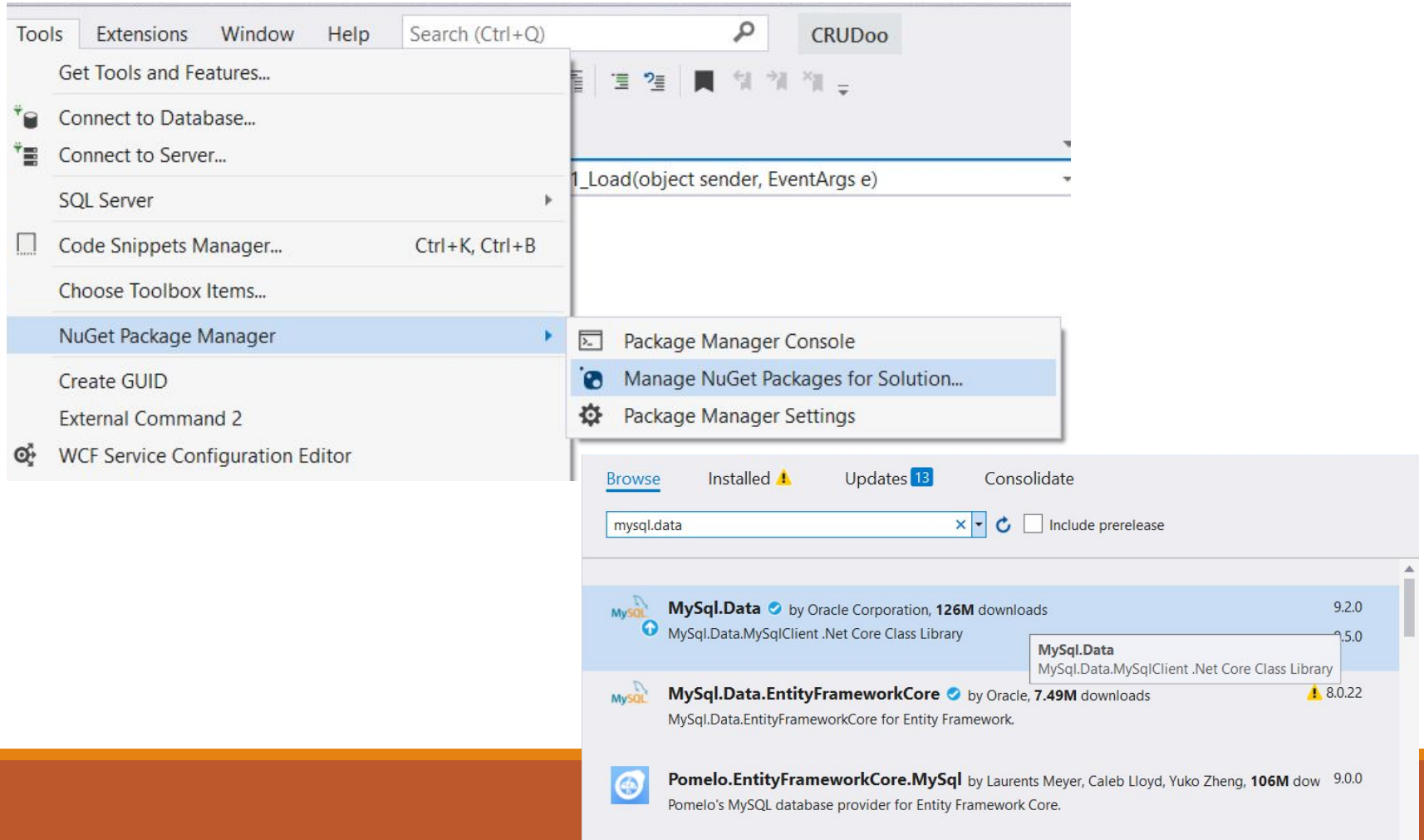
Example

```
dataGridView1.ColumnCount = 3;
dataGridView1.Columns[0].Name = "ID";
dataGridView1.Columns[1].Name = "Name";
dataGridView1.Columns[2].Name = "Department";

dataGridView1.Rows.Add(111, ”Mike Acosta”, “IT”);

//accessing rows in dataGridView1
dataGridView1.Rows[e.RowIndex].Cells[0].Value.ToString())
```

Adding MySQL from Package Manager



Adding MySQL from References

1. From the Solutions Explorer, select **References** and right-click the **Add New References**.

2. Browse **c:\Program Files (x86)>MySQL**, select the **MySQL connector folder** and select the file **MySql.Data.dll** and Click **Add.**

try-catch blocks are used for exception handling, allowing you to gracefully manage errors that occur during program execution.

Try Catch

try-catch blocks are used for exception handling, allowing you to gracefully manage errors that occur during program execution.

Purpose:

try: Encloses the code that might potentially throw an exception.

catch: Contains the code that will be executed if an exception is thrown within the corresponding try block. This block allows you to handle the error and prevent the program from crashing.

Try Catch

```
try
        {
            int[] numbers = { 1, 2, 3 };
            Console.WriteLine(numbers[5]); // This will throw an exception
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
```

Custom Error Message

```
try
        {
            int[] numbers = { 1, 2, 3 };
            Console.WriteLine(numbers[5]); // This will throw an exception
        }
        catch (Exception ex)
        {
            Console.WriteLine("Index out of range");
        }
```