ADO.NET

# Namespaces

using Microsoft.Data.SqlClient; // For SQL Server database operations

using System.Data; // For SQL Server database operations

using System.Data.OleDb; // For MS Access database operations

using System.Runtime.Versioning; // For Windows-specific features

using System.Text.Json; // For JSON serialization

# Connection String

private static readonly string connectionString = "Server=.;Database=AdoNetTrainingDB;Trusted\_Connection=True;TrustServerCertificate=True;";

private static readonly string accessConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=SampleAccessDB.accdb ";

# Connected Architecture

//Streaming data directly from the database, connection remains open.

using SqlConnection conn = new SqlConnection(connectionString);

conn.Open();

using SqlCommand cmd = new SqlCommand("SELECT TOP 5 Id, Name FROM YourTable", conn);

using SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

Console.WriteLine($"{reader["Id"]} - {reader["Name"]}");

}

# Execute Commands

// ExecuteScalar (single value)

object result = cmd.ExecuteScalar();

// ExecuteNonQuery (INSERT/UPDATE/DELETE)

int rows = cmd.ExecuteNonQuery();

// ExecuteReader (read multiple rows)

SqlDataReader r = cmd.ExecuteReader();

# CRUD (Parameterized)

using SqlCommand cmd = new SqlCommand("INSERT INTO YourTable (Name) VALUES (@name)", conn);

cmd.Parameters.Add("@name", SqlDbType.NVarChar, 100).Value = "Value";

# Disconnected Architecture

using SqlConnection conn = new SqlConnection(connectionString);

using SqlDataAdapter adapter = new SqlDataAdapter("SELECT TOP 5 Id, Name FROM YourTable", conn);

DataSet dataSet = new DataSet();

int rowsFetched = adapter.Fill(dataSet, "YourTable");

if (rowsFetched > 0)

{

DataTable? table = dataSet.Tables["YourTable"];

if (table != null && table.Rows.Count > 0)

{

Console.WriteLine($"Data loaded into DataSet ({table.Rows.Count} rows).");

foreach (DataRow row in table.Rows)

{

Console.WriteLine($"ID: {row["Id"]}, Name: {row["Name"]}");

}

}

}

# Stored Procedure

using SqlConnection conn = new SqlConnection(connectionString);

conn.Open();

using SqlCommand cmd = new SqlCommand("MyStoredProc", conn)

{

CommandType = CommandType.StoredProcedure

};

// Input parameter

cmd.Parameters.Add("@Param1", SqlDbType.NVarChar, 100).Value = "This is a test parameter.";

// Output parameter

SqlParameter outputParam = new SqlParameter("@ResultMessage", SqlDbType.NVarChar, 200)

{

Direction = ParameterDirection.Output

};

cmd.Parameters.Add(outputParam);

# SQL Bulk Copy

using SqlBulkCopy bulkCopy = new SqlBulkCopy(conn)

{

DestinationTableName = "YourTable"

};

bulkCopy.WriteToServer(dt);

# Transactions

using SqlTransaction transaction = conn.BeginTransaction();

try

{

using (SqlCommand cmd = new SqlCommand("INSERT INTO YourTable (Name) VALUES (@name)", conn, transaction))

{

cmd.Parameters.Add("@name", SqlDbType.NVarChar, 100).Value = "TransactionTest";

cmd.ExecuteNonQuery();

}

transaction.Commit();

}

catch (SqlException ex)

{

transaction.Rollback();

}

catch (Exception ex)

{

transaction.Rollback();

}

# SqlCommandBuilder

// Update database when adapter.update() is called

SqlCommandBuilder builder = new SqlCommandBuilder(adapter);

adapter.Update(dataSet, "YourTable");

# MS Access via OleDb

using OleDbConnection conn = new OleDbConnection(accessConnectionString);

conn.Open();

Console.WriteLine("\n1. Inserting a new record using parameterized command.");

using OleDbCommand insertCmd = new OleDbCommand("INSERT INTO YourAccessTable (Name, Age) VALUES (?, ?)", conn);

insertCmd.Parameters.Add("?", OleDbType.VarWChar).Value = "New Access Record";

// Parameters use '?' placeholders and are position-based.

# XML Read/Write

ds.WriteXml(xmlFilePath, XmlWriteMode.WriteSchema);

ds.ReadXml(xmlFilePath);

# JSON Serialization

private static readonly JsonSerializerOptions \_jsonOptions = new() { WriteIndented = true };

string json = JsonSerializer.Serialize(obj, new JsonSerializerOptions { WriteIndented = true });

# Tips

- Use 'using' for automatic disposal of connections/readers.

- Always parameterize queries.

- Keep connection strings in appsettings.json or environment variables for production.

- Prefer DataReader for large, forward-only reads.

- Prefer DataAdapter + DataSet for disconnected editing.