UNIT 3
Introduction to ADO.NET

(7 Hours) — Advanced .NET Technology (CACS408)

Note: This PDF may contain errors — please cross-check important points with your syllabus, prescribed textbook and teacher's notes before the exam.

Note: The syllabus typo "ADD.NET" means ADO.NET (ActiveX Data Objects .NET). Examples use SQL Server (SqlConnection). For MySQL/Oracle/OLE DB the class names change (e.g., MySqlConnection, OleDbConnection) but the pattern is identical.

3.1 Introduction and Benefits of ADO.NET

ADO.NET is the set of classes in the .NET Framework (namespace `System.Data`) used to connect to a data source, execute commands and retrieve, manipulate and update data. It supports both a **connected** model (DataReader) and a **disconnected** model (DataSet).

bcalink.com

Benefits of ADO.NET

- **Disconnected data access:** data is cached in a DataSet, so connections are held for a short time — better scalability.
- **Interoperability:** data is represented as XML, so it moves easily between applications and platforms.
- **Scalability:** connection pooling and short-lived connections suit web applications with many users.
- **Strongly typed** data through typed DataSets, with IntelliSense and compile-time checking.
- **Performance:** DataReader is a fast, forward-only reader; no COM marshalling.
- **Common programming model** for many databases through data providers.

ADO.NET compared with classic ADO

Feature	Classic ADO	ADO.NET
Main data object	Recordset (one table-like result)	DataSet (many DataTables with relations) and DataReader
Connection model	Mostly connected; cursors kept on server	Disconnected by default; also connected DataReader
Data format	Binary / COM based	XML based; easy to serialize and send
Technology	COM (unmanaged)	Managed .NET classes
Relations / constraints	JOIN queries only	DataRelation, primary/foreign key constraints in memory
Scalability	Limited — locks and open cursors	High — no open cursors, short connections
Firewall friendliness	Poor (COM marshalling)	Good (plain text XML over HTTP)
Typing	Variant data types	Strongly typed data and typed DataSets

bcalink.com

3.2 ADO.NET Architecture

ADO.NET has two main parts: (1) the **.NET Data Provider** (managed provider) which talks to the database, and (2) the **DataSet** which holds the data in memory independent of the database.

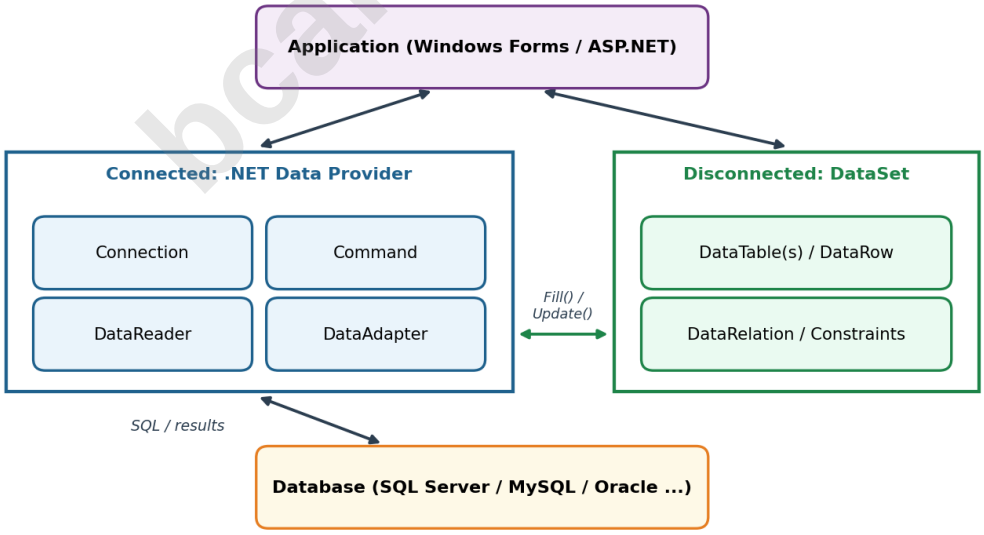


Fig 3.1: ADO.NET architecture (connected and disconnected)

Provider object	Role
Connection	Opens/closes the connection to the database (SqlConnection).
Command	Runs SQL statements or stored procedures (SqlCommand).

Provider object	Role
DataReader	Fast, read-only, forward-only stream of rows (SqlDataReader) — **connected**.
DataAdapter	Bridge between database and DataSet: Fill() loads data, Update() saves changes — **disconnected**.

Connected vs disconnected architecture

Point	Connected (DataReader)	Disconnected (DataSet)
Connection	Must stay open while reading	Opened only to fill/update, then closed
Navigation	Forward-only, read-only	Move in any direction; edit, add, delete rows
Memory	Low (one row at a time)	Higher (whole result cached in memory)
Speed	Faster for simple reads	Slower to load, but flexible afterwards
Best for	Display/report of data once	Editing, multiple tables, binding, XML, offline use

bealink.com

3.3 Shared and Database-Specific Classes

Some ADO.NET classes are **shared** (provider-independent, in `System.Data`); others are **database-specific** and exist separately for each provider.

Shared classes (System.Data)	Database-specific classes (per provider)
DataSet, DataTable, DataRow, DataColumn, DataRelation, DataView, Constraint (UniqueConstraint, ForeignKeyConstraint)	Connection, Command, DataReader, DataAdapter, Parameter, Transaction, CommandBuilder

Data source	Namespace	Class prefix
SQL Server	Microsoft.Data.SqlClient (older: System.Data.SqlClient)	Sql… (SqlConnection)
OLE DB (Access, Excel)	System.Data.OleDb	OleDb… (OleDbConnection)
ODBC	System.Data.Odbc	Odbc… (OdbcConnection)
Oracle	Oracle.ManagedDataAccess.Client	Oracle… (OracleConnection)
MySQL	MySql.Data.MySqlClient (NuGet package)	MySql… (MySqlConnection)

Note: Managed provider = .NET Data Provider. ADO.NET also has generic base classes in System.Data.Common (DbConnection, DbCommand…) for writing provider-independent code.

3.4 Using a Database Connection

A **connection string** tells the provider which server, database and credentials to use. Open the connection as late as possible and close it as soon as possible; a `using` block closes it automatically and returns it to the **connection pool**.

```
string cs = "Server=localhost;Database=College;User Id=sa;Password=pass123;" +
            "TrustServerCertificate=True;";
// Windows authentication:
// "Server=.\\SQLEXPRESS;Database=College;Integrated Security=True;"

using (SqlConnection con = new SqlConnection(cs))
{
    con.Open();
    Console.WriteLine(con.State);          // Open
    Console.WriteLine(con.ServerVersion);
}                                           // closed automatically here
```

Program 3.1: Opening and closing a connection

Executing commands

Method	Returns	Use for
ExecuteReader()	SqlDataReader	SELECT returning many rows
ExecuteNonQuery()	int (rows affected)	INSERT, UPDATE, DELETE, DDL
ExecuteScalar()	object (first column of first row)	SELECT COUNT(*), SUM(), single value

```
using (SqlConnection con = new SqlConnection(cs))
{
    con.Open();
    // INSERT using parameters (prevents SQL injection)
    SqlCommand ins = new SqlCommand(
        "INSERT INTO Student(Name, Age) VALUES(@n, @a)", con);
    ins.Parameters.AddWithValue("@n", "Gita");
    ins.Parameters.AddWithValue("@a", 20);
    int rows = ins.ExecuteNonQuery();

    // single value
    SqlCommand cnt = new SqlCommand("SELECT COUNT(*) FROM Student", con);
    int total = (int)cnt.ExecuteScalar();
}
```

Program 3.2: ExecuteNonQuery and ExecuteScalar

> *Note: Never build SQL by joining user input into a string — it allows SQL injection. Use parameters or stored procedures.*

Working with DataReader

A **DataReader** reads rows sequentially while the connection stays open. Call `Read()` to move to the next row; it returns false at the end.

bcalink.com

```
using (SqlConnection con = new SqlConnection(cs))
{
    con.Open();
    SqlCommand cmd = new SqlCommand("SELECT Id, Name, Age FROM Student", con);
    using (SqlDataReader dr = cmd.ExecuteReader())
    {
        while (dr.Read())
        {
            int id = dr.GetInt32(0);
            string name = dr["Name"].ToString();
            Console.WriteLine(id + " " + name + " " + dr["Age"]);
        }
    }   // reader closed
}
```

Program 3.3: Reading rows with SqlDataReader

3.5 Working with DataSets and Data Adapters

A **DataSet** is an in-memory, disconnected cache of data containing a collection of `DataTable` objects (each with DataRows and DataColumns), plus `DataRelation` and constraints. It does not know where the data came from. The **DataAdapter** fills it and sends changes back.

```
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Student", cs);
DataSet ds = new DataSet();
da.Fill(ds, "Student");                       // opens, fills, closes

DataTable dt = ds.Tables["Student"];
foreach (DataRow row in dt.Rows)
    Console.WriteLine(row["Name"] + " - " + row["Age"]);

// modify in memory
dt.Rows[0]["Age"] = 22;                       // update
DataRow nr = dt.NewRow(); nr["Name"] = "Hari"; nr["Age"] = 19;
dt.Rows.Add(nr);                              // insert
dt.Rows[1].Delete();                          // delete

// send all changes to the database
SqlCommandBuilder cb = new SqlCommandBuilder(da);   // auto-generates commands
da.Update(ds, "Student");

// relation between two tables, and a filtered view
ds.Relations.Add("StudDept", ds.Tables["Dept"].Columns["Id"],
                             ds.Tables["Student"].Columns["DeptId"]);
DataView dv = new DataView(dt) { RowFilter = "Age > 20", Sort = "Name" };
```

Program 3.4: DataAdapter, DataSet, DataTable, DataRelation, DataView

3.6 Data Binding

Data binding connects UI controls to a data source so that displayed values update automatically.

- **Simple binding:** binds one property of a control to one data field, e.g. a TextBox's Text to a column (`txtName.DataBindings.Add("Text", dt, "Name")`).

bcalink.com

- **Complex binding:** a control bound to a whole list/table, e.g. DataGridView, ListBox, ComboBox (`DataSource`, `DisplayMember`, `ValueMember`).
- **BindingSource** sits between the data and controls, handling currency (current row), sorting and filtering; **BindingNavigator** gives first/previous/next/last/add/delete buttons.

```
DataTable dt = new DataTable();
new SqlDataAdapter("SELECT * FROM Student", cs).Fill(dt);

BindingSource bs = new BindingSource();
bs.DataSource = dt;

dataGridView1.DataSource = bs;                        // complex binding
txtName.DataBindings.Add("Text", bs, "Name");         // simple binding
txtAge.DataBindings.Add("Text", bs, "Age");

comboDept.DataSource = deptTable;                     // combo box
comboDept.DisplayMember = "DeptName";                 // shown text
comboDept.ValueMember = "Id";                         // hidden value

bindingNavigator1.BindingSource = bs;
bs.Filter = "Age >= 18";
```

Program 3.5: Simple and complex data binding (Windows Forms)

3.7 Typed DataSets

A **typed DataSet** is a class generated from an XML schema (.xsd) that inherits from DataSet and exposes tables and columns as strongly typed properties. It is created in Visual Studio with the DataSet Designer (Add → New Item → DataSet).

Untyped DataSet	Typed DataSet
`ds.Tables["Student"].Rows[0]["Name"]`	`ds.Student[0].Name`
Names are strings — typos found only at run time	Names are properties — errors found at compile time
Needs casting / conversion of values	Values already have correct .NET type
No IntelliSense for columns	Full IntelliSense support
More flexible at run time	Tied to a fixed schema (regenerate if the schema changes)

3.8 Transactions

A **transaction** groups several operations into one unit of work that either fully succeeds (**Commit**) or is completely undone (**Rollback**). Transactions follow the **ACID** properties: Atomicity, Consistency, Isolation, Durability. A classic example is transferring money: debit one account and credit another — both must happen or neither.

bcalink.com

```
using (SqlConnection con = new SqlConnection(cs))
{
    con.Open();
    SqlTransaction tx = con.BeginTransaction();
    try
    {
        SqlCommand c1 = new SqlCommand(
          "UPDATE Account SET Bal = Bal - 500 WHERE Id = 1", con, tx);
        SqlCommand c2 = new SqlCommand(
          "UPDATE Account SET Bal = Bal + 500 WHERE Id = 2", con, tx);
        c1.ExecuteNonQuery();
        c2.ExecuteNonQuery();
        tx.Commit();                  // make both permanent
    }
    catch (Exception)
    {
        tx.Rollback();                // undo everything
    }
}
```

Program 3.6: Local transaction with Commit and Rollback

Isolation levels (`IsolationLevel`) control how transactions see each other's changes: ReadUncommitted, ReadCommitted (default in SQL Server), RepeatableRead, Serializable, Snapshot. For work across several connections use `TransactionScope` (System.Transactions): create the scope, do the work and call `scope.Complete()`.

bcalink.com

Key Takeaways — Unit 3

✓ ADO.NET = System.Data classes for data access; supports connected (DataReader) and disconnected (DataSet) models.

✓ Compared with ADO: managed, XML-based, disconnected, scalable, supports relations and strong typing.

✓ Provider objects: Connection, Command, DataReader, DataAdapter. Shared objects: DataSet, DataTable, DataRelation, DataView.

✓ ExecuteReader (rows), ExecuteNonQuery (insert/update/delete), ExecuteScalar (single value).

✓ DataAdapter.Fill() loads a DataSet and Update() saves changes; use parameters to prevent SQL injection.

✓ Data binding: simple (one value), complex (lists/grids), via BindingSource.

✓ Typed DataSets give compile-time checking and IntelliSense.

✓ Transactions: BeginTransaction → Commit or Rollback, following ACID.

— End of Unit 3 —