ADO.NET COMPLETE NOTES WITH THEORY & EXAMPLES

---

1. What is ADO.NET?

ADO.NET (ActiveX Data Objects for .NET) is a data access technology provided by Microsoft that allows .NET applications to interact with relational databases like SQL Server. It supports both **connected** (live DB connection) and **disconnected** (offline cache using `DataSet`) architectures.

📌**Real-life analogy**: Think of `SqlConnection` as a telephone line to a database. You use it to talk (send queries), listen (read results), and then hang up (close connection).

---

2. Key Components of ADO.NET

- **SqlConnection**: Establishes the DB connection.
- **SqlCommand**: Executes queries/commands (like `SELECT`, `INSERT`, etc.).
- **SqlDataReader**: Reads rows forward-only and read-only. Efficient for fast, sequential access.
- **SqlDataAdapter**: Acts as a bridge between `DataSet` and database.
- **DataSet / DataTable**: Represents an in-memory cache of data.
- **SqlTransaction**: Helps group queries into atomic units to commit or rollback.

---

3. Command Execution Methods

Method	Use	Returns
`ExecuteNonQuery()`	INSERT, UPDATE, DELETE	Affected rows (int)
`ExecuteScalar()`	Single value queries (e.g. `SELECT COUNT(*)`)	Single value (object)
`ExecuteReader()`	SELECT returning multiple rows or columns	`SqlDataReader`

---

4. Connection String Format

```
string connStr = "Data Source=(localdb)\\MSSQLLocalDB;Initial
Catalog=JKJune25;Integrated Security=True";
```

---

5. Basic Select Query Example

```
SqlConnection cn = new SqlConnection(connStr);
cn.Open();
SqlCommand cmd = new SqlCommand("SELECT * FROM Employees", cn);
SqlDataReader dr = cmd.ExecuteReader();
while (dr.Read())
{
    Console.WriteLine(dr["Name"]);
}
dr.Close();
cn.Close();
```

This code reads all employees and prints their names.

---

6. Parameterized Queries

Why? To prevent SQL Injection and type mismatches.

```
cmd.CommandText = "INSERT INTO Employees VALUES(@EmpNo, @Name, @Basic,
@DeptNo)";
cmd.Parameters.AddWithValue("@EmpNo", emp.EmpNo);
cmd.Parameters.AddWithValue("@Name", emp.Name);
```

🔗Use this instead of string concatenation in SQL.

---

7. Executing Stored Procedures

Stored procedures are pre-written SQL blocks stored in the database.

```
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = "InsertEmployee"; // Stored procedure name
```

📌Just like a function call in SQL Server.

---

8. Transactions in ADO.NET

Used to execute multiple operations as a single unit.

```
SqlTransaction t = cn.BeginTransaction();
cmd.Transaction = t;
```

```
try {
    cmd.ExecuteNonQuery();
    t.Commit(); // all good
} catch {
    t.Rollback(); // undo all
}
```

📌Use when multiple statements should succeed or fail together.

🔍 **ACID Properties of Transactions:**

Property	Meaning
Atomicity	All steps in a transaction complete or none do
Consistency	Database goes from one valid state to another
Isolation	Transactions are isolated from each other
Durability	Once committed, the changes are permanent

---

9. MARS (Multiple Active Result Sets)

Allows more than one `SqlDataReader` to be open on a connection simultaneously.

Normally, ADO.NET allows only one `SqlDataReader` per connection. But if you want to read data from multiple result sets or nested queries, enable MARS.

```
string connStr = "Data Source=...;Initial Catalog=...;Integrated
Security=True;MultipleActiveResultSets=True";
```

📌Useful for reading departments and employees nested inside one another.

---

10. NextResult() Usage

Used when your SQL returns multiple result sets:

```
SqlCommand cmd = new SqlCommand("SELECT * FROM Employees; SELECT * FROM
Departments", cn);
SqlDataReader dr = cmd.ExecuteReader();
while (dr.Read())
{
    Console.WriteLine(dr["Name"]);
}
if (dr.NextResult())
{
    while (dr.Read())
    {
        Console.WriteLine(dr["DeptName"]);
    }
}
dr.Close();
```

📌Helpful when stored procedures return multiple result sets or when using compound SQL statements.

---

11. Reusability with GetDataReader()

Used to return a `SqlDataReader` from a function.

```
static SqlDataReader GetDataReader()
{
    SqlConnection cn = new SqlConnection(connStr);
    cn.Open();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Employees", cn);
    return cmd.ExecuteReader(CommandBehavior.CloseConnection);
}
```

📌Useful to separate data-access logic from UI/business logic.

---

12. CallFuncReturningSqlDataReader()

```
SqlDataReader dr = GetDataReader();
while (dr.Read())
{
    Console.WriteLine(dr["Name"]);
}
dr.Close();
```

📌This method calls `GetDataReader()` and prints out employee names.

---

13. GetSingleEmployee() Example

Fetch one employee based on primary key:

```
static Employee GetSingleEmployee(int EmpNo)
{
    Employee emp = null;
    SqlConnection cn = new SqlConnection(connStr);
    cn.Open();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Employees WHERE EmpNo =
@EmpNo", cn);
    cmd.Parameters.AddWithValue("@EmpNo", EmpNo);
    SqlDataReader dr = cmd.ExecuteReader();
    if (dr.Read())
    {
        emp = new Employee {
            EmpNo = Convert.ToInt32(dr["EmpNo"]),
            Name = dr["Name"].ToString(),
            Basic = Convert.ToDecimal(dr["Basic"]),
            DeptNo = Convert.ToInt32(dr["DeptNo"])
        };
    }
    dr.Close();
    cn.Close();
    return emp;
}
```

📌Although one row is returned, `ExecuteReader` is used since multiple columns are involved.

---

14. GetAllEmployees()

```
static List<Employee> GetAllEmployees()
{
    List<Employee> list = new List<Employee>();
    SqlConnection cn = new SqlConnection(connStr);
    cn.Open();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Employees", cn);
    SqlDataReader dr = cmd.ExecuteReader();
    while (dr.Read())
    {
        list.Add(new Employee
        {
            EmpNo = Convert.ToInt32(dr["EmpNo"]),
            Name = dr["Name"].ToString(),
            Basic = Convert.ToDecimal(dr["Basic"]),
            DeptNo = Convert.ToInt32(dr["DeptNo"])
        });
    }
    dr.Close();
    cn.Close();
    return list;
}
```

📌This method returns all employees in a list.

This document now contains all ADO.NET topics with **theory + code + explanation** to build strong understanding for real-world .NET database interaction, including advanced features like **MARS**, **NextResult**, **Transactions**, **reusable DataReader methods**, and **stored procedures**.