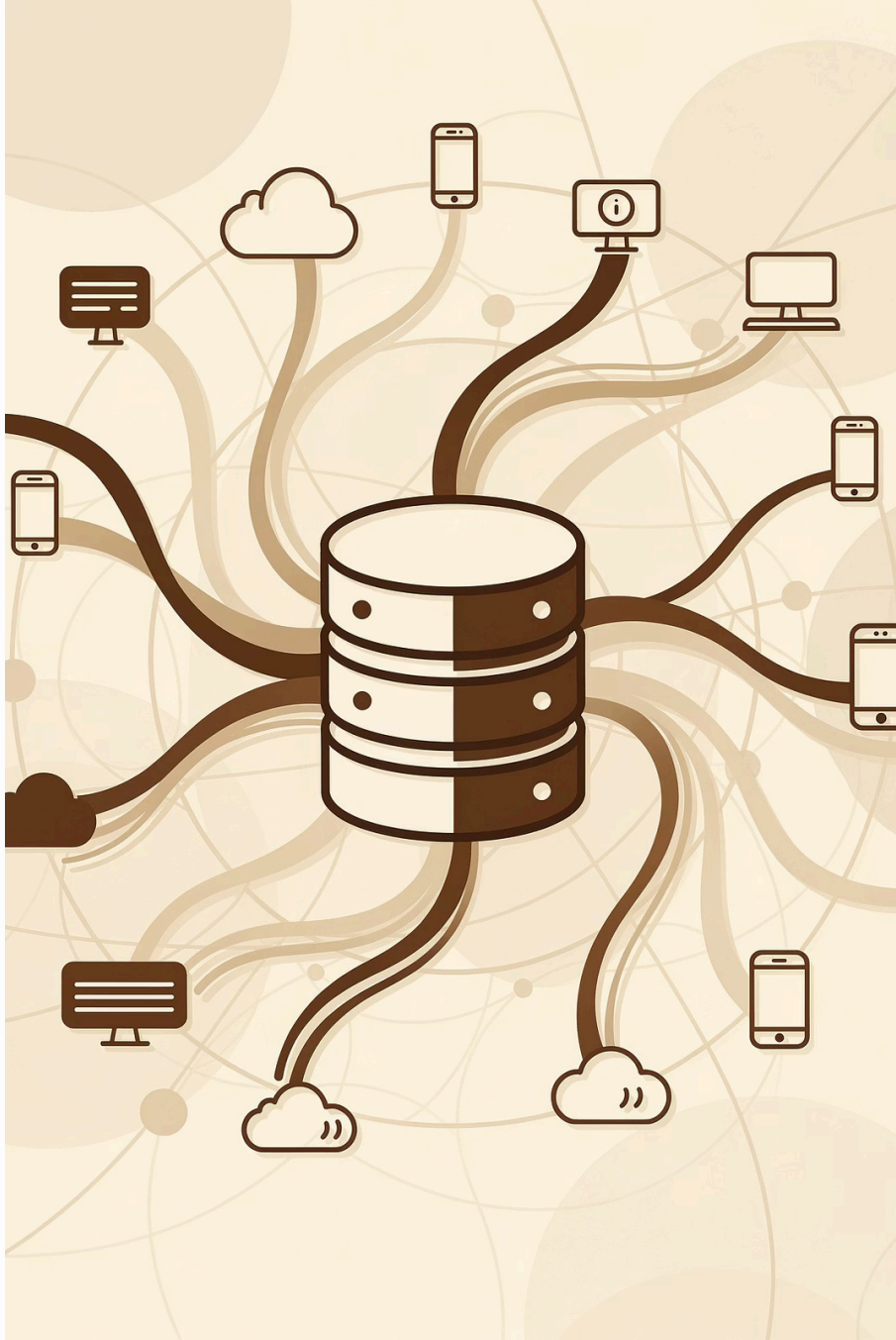
ADO.NET: Database Connectivity in .NET

Pravin Nikam



Unit Overview

What You'll Learn

ADO.NET Architecture

Understand the core components and how they work together to enable database connectivity

Data Management

Work with DataSets, DataTables, and managed providers to handle data efficiently

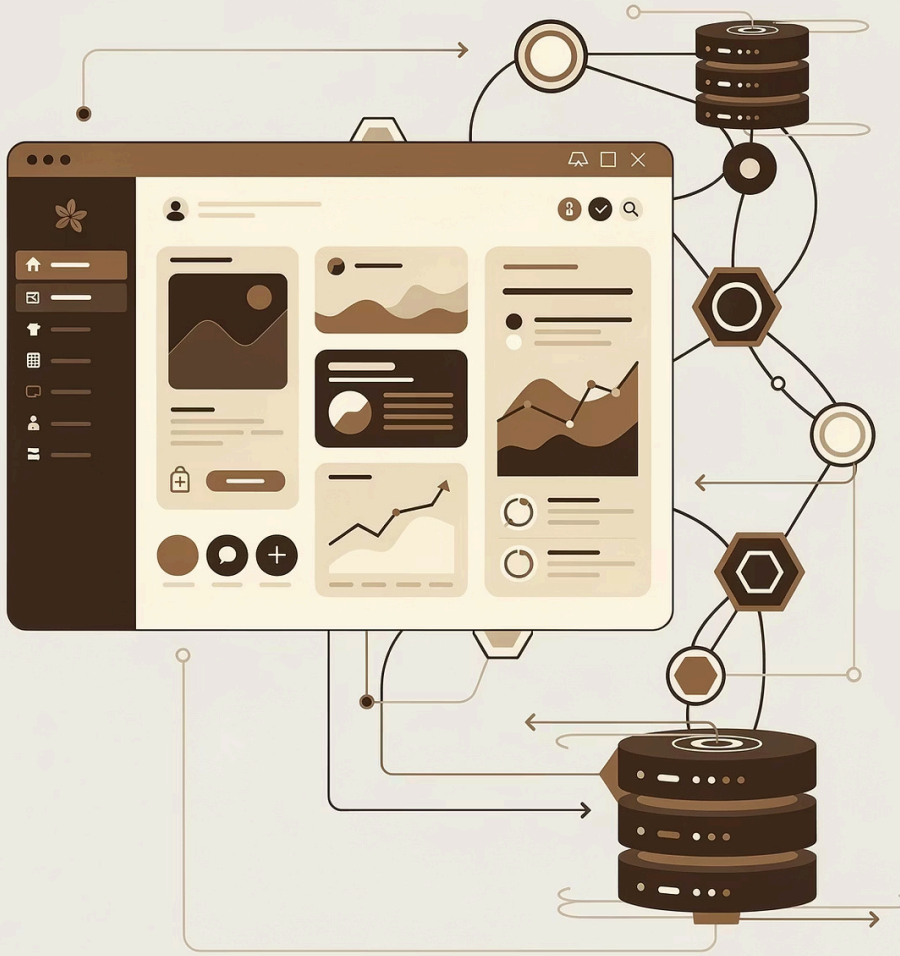
Database Operations

Perform complete CRUD operations and bind data to web controls securely

Why ADO.NET Matters

Almost every real-world application needs to store and retrieve data. Whether you're building a student management system, e-commerce platform, or social network, understanding database connectivity is fundamental to creating dynamic, data-driven applications.

ADO.NET is the bridge between your C# code and databases like SQL Server, enabling you to read, write, update, and delete information seamlessly. It's a core skill that will power nearly every application you build in your professional career.



What is ADO.NET?

ADO.NET (ActiveX Data Objects for .NET) is a set of classes that expose data access services for .NET Framework programmers. It provides a consistent way to work with different data sources, primarily Microsoft SQL Server databases.

Think of ADO.NET as a universal translator between your C# application and various databases. Just as you might use different adapters to charge devices in different countries, ADO.NET provides the right "adapter" to communicate with different database systems.

Key Capabilities

- Connect to databases
- Execute SQL commands
- Retrieve and manipulate data
- Update database records
- Work with data offline

Benefits of ADO.NET

High Performance

Optimized for speed with efficient data retrieval mechanisms. The disconnected architecture minimizes database load and enables faster application response times.

Scalability

Handles multiple concurrent users efficiently through connection pooling and disconnected data operations, making it ideal for enterprise applications.

Flexibility

Works with various data sources including SQL Server, Oracle, MySQL, and even XML files, giving you options for different project requirements.

Seamless Integration

Deeply integrated with .NET Framework, providing type safety, IntelliSense support, and consistent programming patterns across applications.

ADO.NET vs Classic ADO

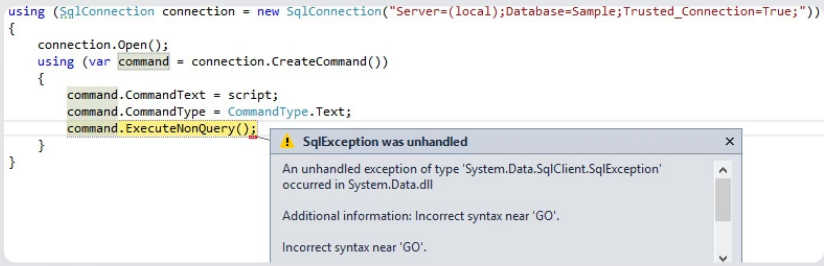
Understanding the evolution from Classic ADO to ADO.NET helps you appreciate modern database programming approaches and why certain design decisions were made.

Classic ADO (Old Approach)

- Required constant database connection
- Used COM-based technology
- Limited offline capabilities
- Recordset-based data handling

ADO.NET (Modern Approach)

- Supports disconnected architecture
- Built for .NET Framework
- Rich offline data manipulation
- DataSet and DataTable objects



The ADO.NET Architecture

ADO.NET is organized into two main architectural models: connected and disconnected. This diagram shows how components interact with data sources. The architecture includes data providers (like SqlClient) that connect to databases, and data consumers (like DataSets) that work with data in memory. Understanding this flow is essential for choosing the right approach for your application's needs.

Core Components of ADO.NET

SqlConnection

Establishes a connection to a SQL Server database. Think of it as opening a phone line to the database.

SqlCommand

Executes SQL statements or stored procedures. This is how you tell the database what you want to do.

SqlDataReader

Reads data in a fast, forward-only stream. Best for quickly reading through records once.

SqlDataAdapter

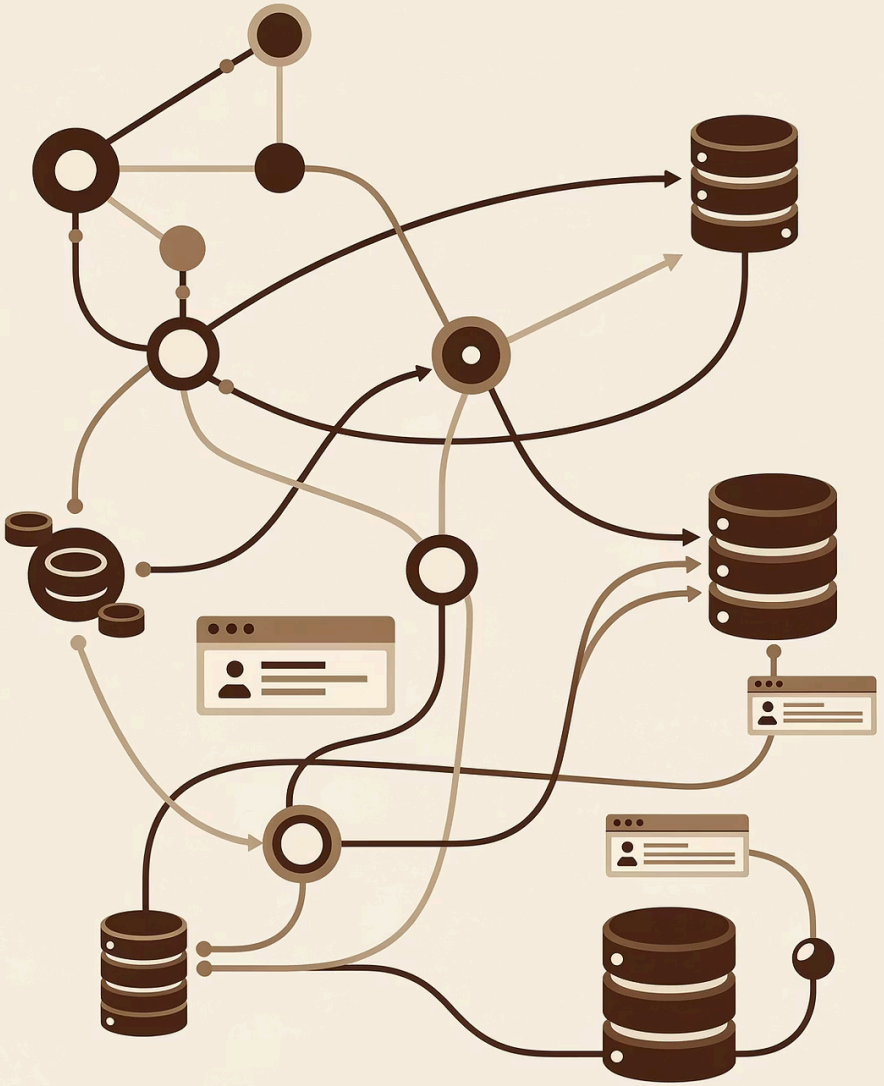
Acts as a bridge between DataSets and the database, filling and updating data automatically.

DataSet

An in-memory representation of database tables. Works like a mini-database in your application's memory.

DataTable

Represents a single table of in-memory data with rows and columns, similar to a spreadsheet.



ARCHITECTURE MODELS

Connected Architecture

In connected architecture, your application maintains an open connection to the database while working with data. It's like keeping a phone call active while discussing information—fast and direct, but occupies the line.

How It Works

Uses SqlConnection and SqlDataReader to access data directly from the database. The connection remains open during data operations and closes immediately after.

When to Use

- Reading data once
- Simple, quick queries
- Real-time data access
- Memory-efficient operations

Disconnected Architecture

Disconnected architecture loads data into memory, closes the database connection, and lets you work with the data offline. It's like downloading a document to edit later—you don't need to stay connected to the source.

Key Advantage

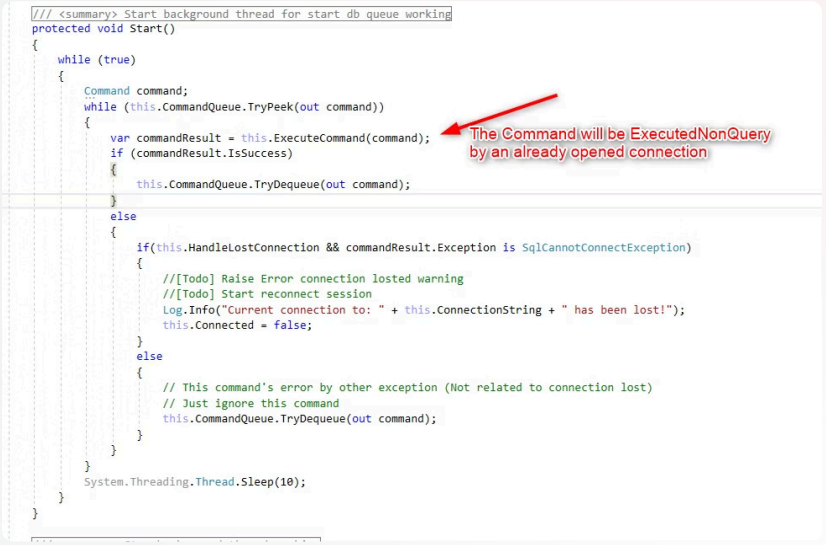
Reduces database load by minimizing connection time. Multiple users can work simultaneously without blocking each other.

Uses DataSet

Stores data in memory as DataTables. Changes are tracked and can be synchronized back to the database later.

Best For

Data manipulation, offline work, binding to UI controls, and scenarios requiring multiple updates.




```
/// <summary> Start background thread for start db queue working
protected void Start()
{
    while (true)
    {
        Command command;
        while (this.CommandQueue.TryPeek(out command))
        {
            var commandResult = this.ExecuteCommand(command);
            if (commandResult.IsSuccess)
            {
                this.CommandQueue.TryDequeue(out command);
            }
            else
            {
                if(this.HandleLostConnection && commandResult.Exception is SqlCannotConnectException)
                {
                    //[Todo] Raise Error connection losted warning
                    //[Todo] Start reconnect session
                    Log.Info("Current connection to: " + this.ConnectionString + " has been lost!");
                    this.Connected = false;
                }
                else
                {
                    // This command's error by other exception (Not related to connection lost)
                    // Just ignore this command
                    this.CommandQueue.TryDequeue(out command);
                }
            }
        }
        System.Threading.Thread.Sleep(10);
    }
}
```

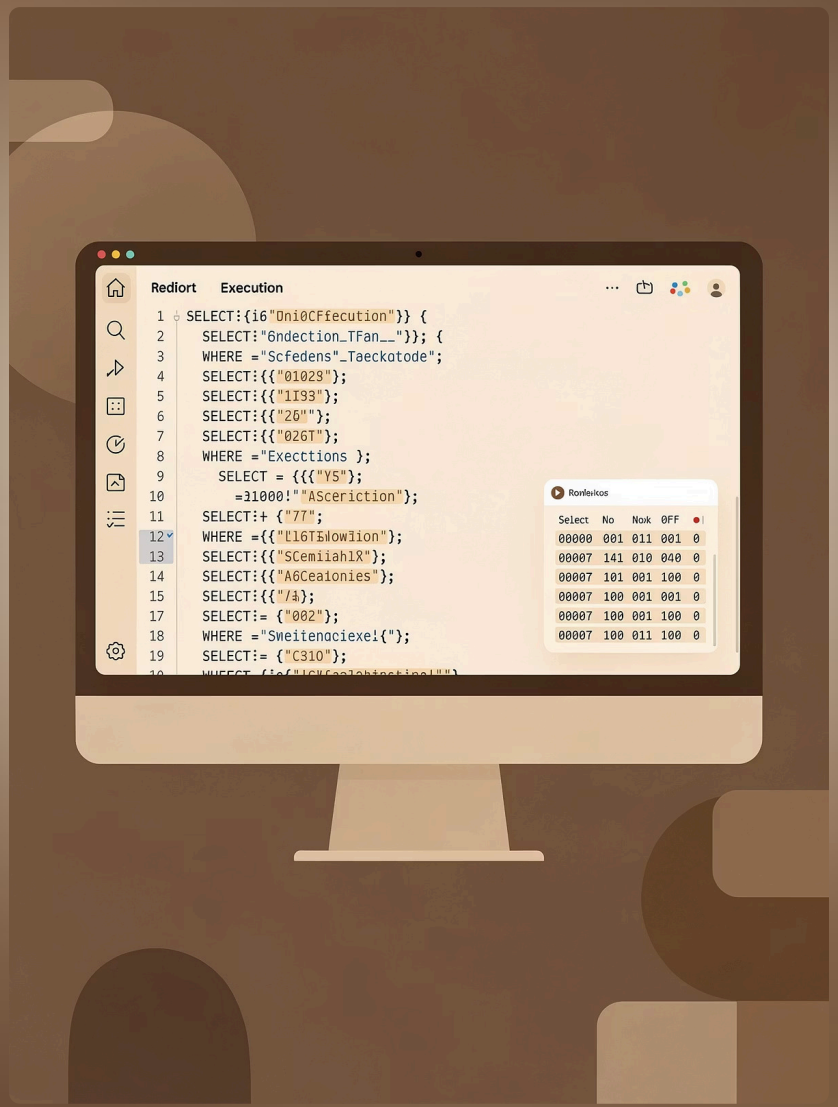
Connected vs Disconnected: At a Glance

Connected	Disconnected
It is connection oriented.	It is dis_connection oriented.
Datareader	DataSet
Connected methods gives faster performance	Disconnected get low in speed and performance.
connected can hold the data of single table	disconnected can hold multiple tables of data
connected you need to use a read only forward only data reader	disconnected you cannot
Data Reader can't persist the data	Data Set can persist the data
It is Read only, we can't update the data.	We can update data

Your First Database Connection

Let's walk through establishing a connection to SQL Server. This is the foundation of all database operations in ADO.NET.

```
using System.Data.SqlClient;

// Connection string contains database location and credentials
string connectionString = "Server=localhost;Database=StudentDB;Integrated Security=true;";

// Create connection object
SqlConnection conn = new SqlConnection(connectionString);

try
{
 conn.Open();
 Console.WriteLine("Connection successful!");
}
catch (Exception ex)
{
 Console.WriteLine("Error: " + ex.Message);
}
finally
{
 conn.Close(); // Always close connections
}
```



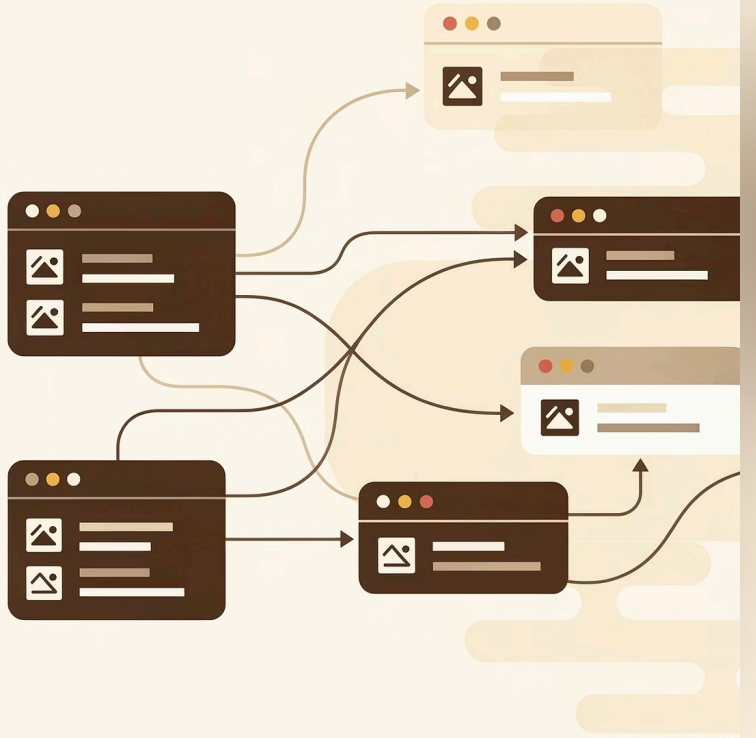
Reading Data with SqlDataReader

SqlDataReader provides the fastest way to read data from a database. It's a forward-only, read-only stream—perfect for displaying data without modification.

```
string query = "SELECT StudentID, Name, Email FROM Students";
SqlCommand cmd = new SqlCommand(query, conn);

conn.Open();
SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())
{
 int id = reader.GetInt32(0);
 string name = reader.GetString(1);
 string email = reader.GetString(2);

 Console.WriteLine($"{id}: {name} - {email}");
}

reader.Close();
conn.Close();
```

DataSets and DataTables

A DataSet is like a mini-database living in your application's memory. It can hold multiple DataTables (like having multiple sheets in an Excel workbook), relationships between them, and even constraints—all without staying connected to the actual database.



Understanding DataSet

What is a DataSet?

An in-memory cache of data retrieved from a database. It's disconnected, meaning it doesn't require an active connection to work with data.

Key Features

- Holds multiple DataTables
- Maintains relationships between tables
- Tracks changes to data
- Can be serialized to XML
- Works completely offline

Think of a DataSet as a portable, lightweight version of your database that you can carry around in your application, modify freely, and then sync back to the real database when ready.

Working with DataTable

A DataTable represents a single table of in-memory data. It has rows and columns just like a database table or Excel spreadsheet. You can add, modify, and delete rows, and even filter and sort data.

```
// Create a new DataTable
DataTable studentsTable = new DataTable("Students");

// Add columns
studentsTable.Columns.Add("StudentID", typeof(int));
studentsTable.Columns.Add("Name", typeof(string));
studentsTable.Columns.Add("Age", typeof(int));

// Add a row
DataRow row = studentsTable.NewRow();
row["StudentID"] = 1;
row["Name"] = "Alice Johnson";
row["Age"] = 20;
studentsTable.Rows.Add(row);
```

Managed Providers

Managed providers are the specialized components that know how to talk to specific database systems. ADO.NET includes different providers for different databases, each optimized for its target system.

SqlClient

Optimized for Microsoft SQL Server. Provides the best performance when working with SQL Server databases.

OleDb

Works with databases supporting OLE DB, including Access and Excel. More generic but slower than SqlClient.

Odbc

Connects to databases via ODBC drivers. Useful for legacy systems and uncommon databases.

For this course, we focus on SqlClient since we're working with Microsoft SQL Server.

SqlDataAdapter: The Bridge

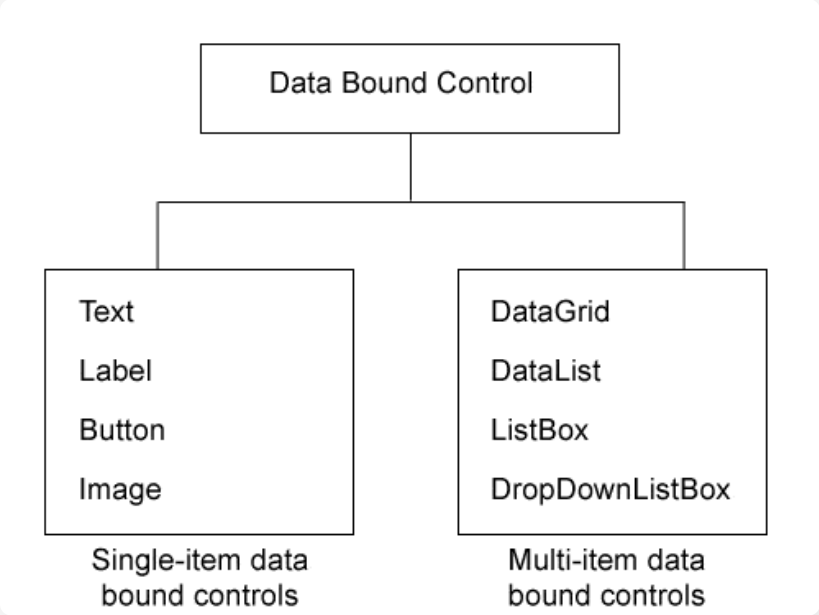
SqlDataAdapter acts as a bridge between your DataSet and the database. It automatically handles opening connections, executing queries, filling DataSets with results, and updating the database with changes—all while managing the disconnected architecture for you.

Primary Functions

- Fill DataSets from database
- Update database from DataSet changes
- Handle connection management
- Execute SELECT, INSERT, UPDATE, DELETE

Why It's Powerful

SqlDataAdapter automates much of the tedious work. You don't manually open connections, execute commands, or loop through results—it does all that for you.



Using SqlDataAdapter

Here's how to use SqlDataAdapter to fill a DataSet with data from the database. Notice how the connection is managed automatically.

```
string connectionString = "Server=localhost;Database=StudentDB;Integrated Security=true;";
string query = "SELECT * FROM Students";

// Create adapter with query and connection
SqlDataAdapter adapter = new SqlDataAdapter(query, connectionString);

// Create DataSet to hold results
DataSet dataSet = new DataSet();

// Fill the DataSet (connection opens and closes automatically)
adapter.Fill(dataSet, "Students");

// Work with data in memory
DataTable studentsTable = dataSet.Tables["Students"];
foreach (DataRow row in studentsTable.Rows)
{
 Console.WriteLine(row["Name"]);
}
```

Data Binding Basics

Data binding is the process of automatically populating UI controls with data from a database or other source. Instead of manually writing code to display each record, you simply connect the control to your data source and it handles the display for you.

Data Source

DataSet, DataTable, or other data collection

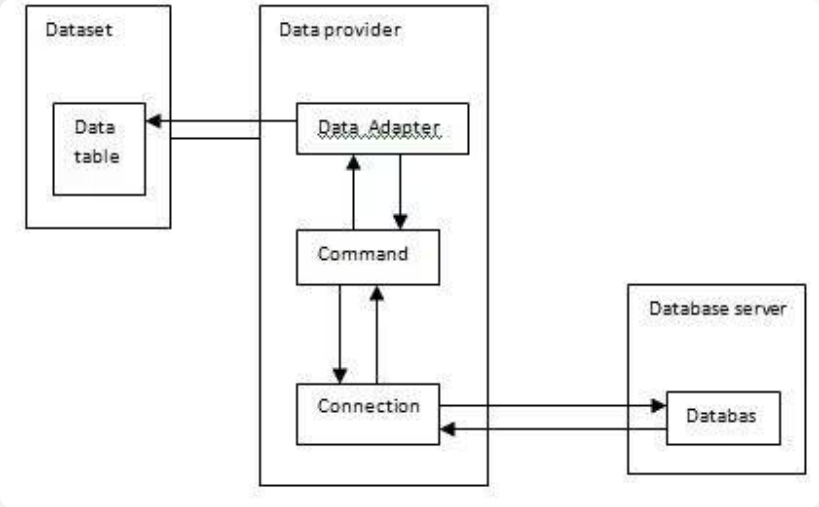
Binding

Connect source to control

UI Control

GridView, DropDownList, Repeater, etc.

Think of data binding like plugging a USB drive into your computer—the connection is made automatically and the data flows without you having to manually transfer each file.



Binding Data to GridView

GridView is one of the most common controls for displaying tabular data in ASP.NET. Here's how to bind a DataSet to a GridView control.

```
// C# Code-Behind
protected void Page_Load(object sender, EventArgs e)
{
    string connectionString = "Server=localhost;Database=StudentDB;Integrated
Security=true;";
    string query = "SELECT StudentID, Name, Email, Major FROM Students";

    SqlDataAdapter adapter = new SqlDataAdapter(query, connectionString);
    DataSet ds = new DataSet();
    adapter.Fill(ds);

    // Bind to GridView
    GridView1.DataSource = ds.Tables[0];
    GridView1.DataBind();
}
```

The GridView automatically creates columns and rows based on your data structure. No manual HTML table creation needed!

Data Source Controls

ASP.NET provides special controls called Data Source Controls that simplify data access. They declaratively handle data retrieval and updates without requiring much C# code.

SqlDataSource

Connects directly to SQL Server databases. Configurable through properties in design view.

ObjectDataSource

Connects to business objects or custom classes. Good for multi-tier applications.

LinqDataSource

Works with LINQ queries. Provides strongly-typed data access.

SqlDataSource is the most straightforward for beginners and perfectly suited for learning database operations.

SqlDataSource Control

SqlDataSource is a powerful server control that enables you to access databases declaratively in your ASPX markup, reducing the amount of C# code you need to write. It handles connection management, command execution, and parameter passing automatically.

Key Properties

- ConnectionString
- SelectCommand
- InsertCommand
- UpdateCommand
- DeleteCommand

Why Use It?

Less code to write, easier maintenance, and automatic parameter handling. Perfect for rapid development and learning.

Update and Delete Operations

Similarly, you can configure UpdateCommand and DeleteCommand to modify or remove records. Parameters ensure safe value passing and prevent SQL injection attacks.

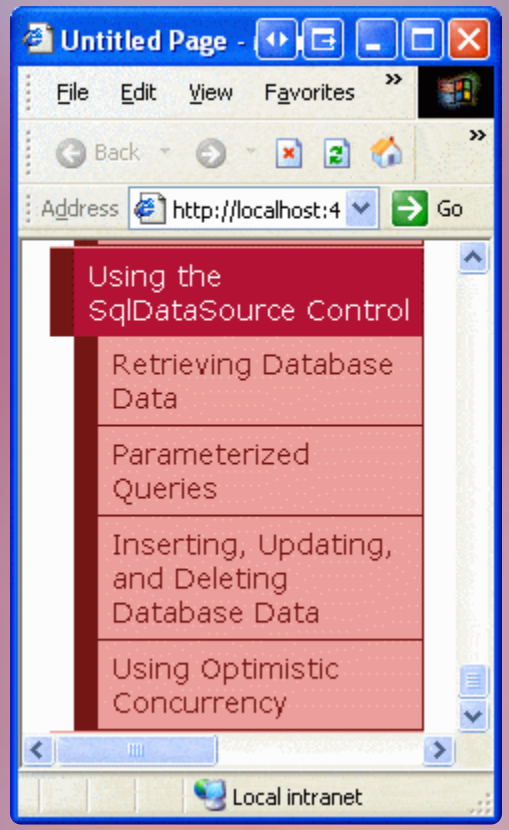
Update Example

```
UpdateCommand="UPDATE
Students
SET Name=@Name, Email=@Email
WHERE StudentID=@StudentID"
```

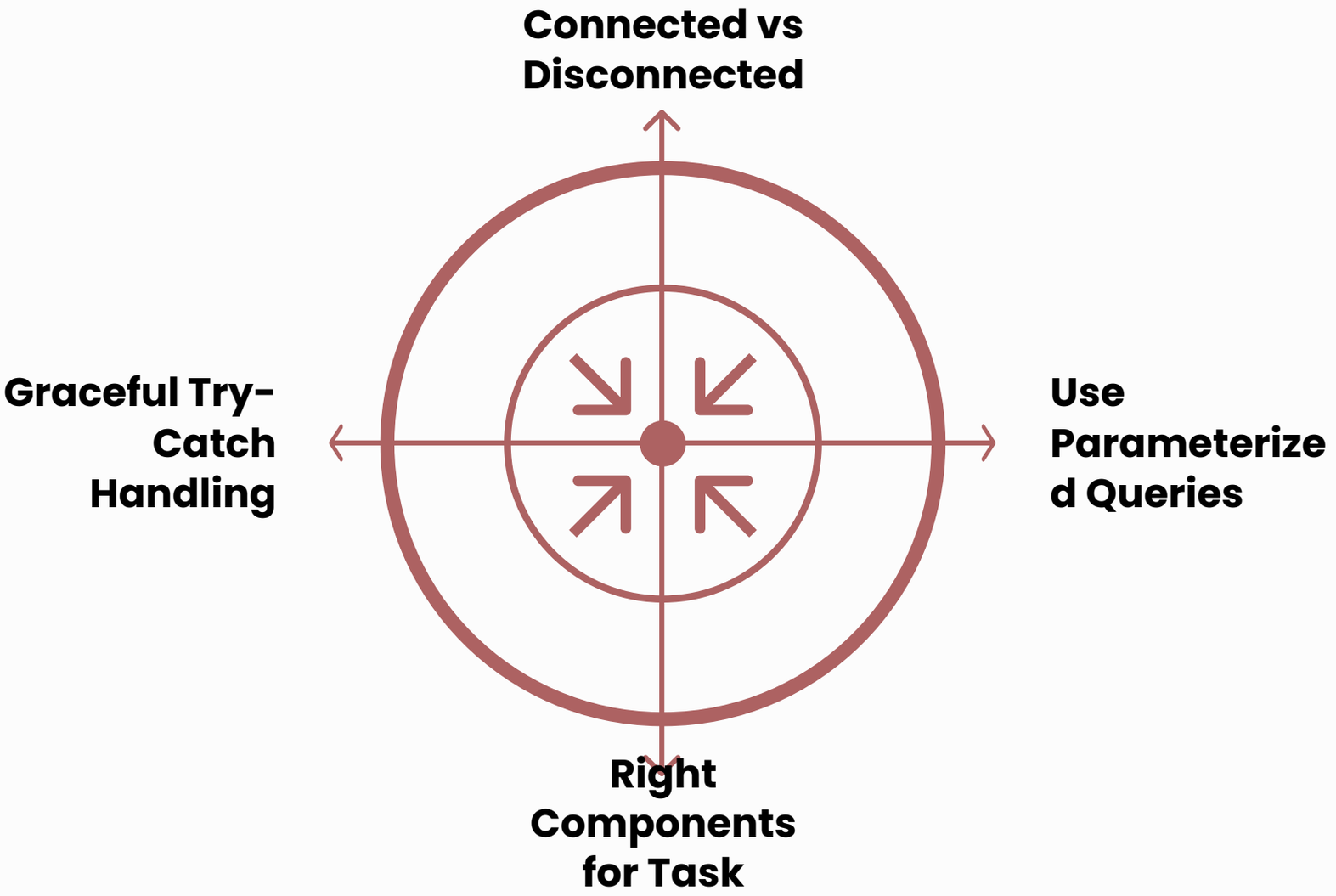
Delete Example

```
DeleteCommand="DELETE FROM
Students
WHERE StudentID=@StudentID"
```

GridView controls can automatically generate Edit and Delete buttons when you enable editing and deleting features.



Key Takeaways



You've learned the foundation of database programming in .NET—from establishing connections to performing complete CRUD operations. These skills are essential for building any real-world web application. Practice building forms that interact with databases, and always prioritize security through parameterized queries and proper error handling.

Practical Application Mapping

The concepts you've learned directly map to practical labs you'll complete. Each lab reinforces different aspects of ADO.NET.

1. **Session State in SQL Server**

   Foundation for database integration and understanding server-side state management with databases

2. **User Input Storage**

   Practice INSERT operations by capturing form data and storing it in database tables

3. **Currency Conversion**

   Store conversion history and retrieve past calculations from the database

4. **Temperature Conversion**

   Implement database logging to track all conversion requests for analysis

WHAT'S NEXT

Moving Forward

Unit 4 Preview

Windows Forms and Controls

Build rich desktop applications with graphical user interfaces. You'll explore WinForms controls, event handling, and creating interactive desktop applications.

Continue Practicing

Build a complete student management system using ADO.NET. Practice all CRUD operations and experiment with both connected and disconnected architectures. The more you code, the more natural database programming becomes.

Remember: every great web application you've ever used relies on the database concepts you just learned. You're building the foundation for your career as a developer.