Programming in C#

Unit-5
Database Programming with C#

11-1-2021,18-1-2021,20-1-2021,
21-1-2021,22-1-2021,23-1-2021,25-1-2021

Dr.M.Paul Arokiadass Jerald

SYLLABUS

UNIT - V: DATABASE

- Creating Connection String – Creating a Connection to a Database
- Creating a Command Object
- Working with Data Adapters
- Using Data Reader to work with Databases
- Using Dataset.

5.1 ADO.NET

- ADO.NET is a set of classes (a framework) to interact with data sources such as databases and XML files.
- ADO is the acronym for ActiveX Data Objects.
- The data access classes for the .NET framework
- Designed for highly efficient data access
- Support for XML and disconnected record sets

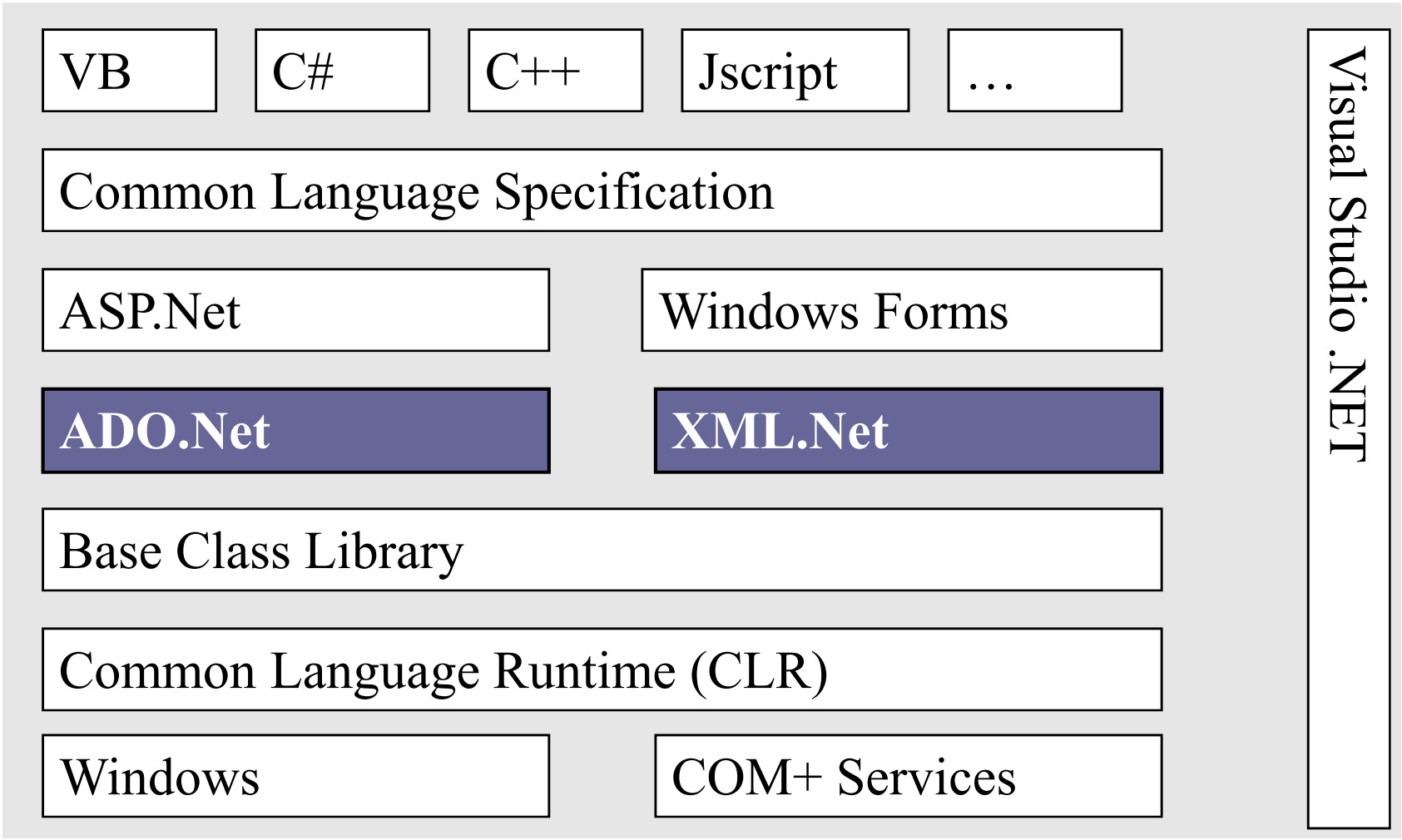
ADO.Net in.NET framework

- A standard cross language interface
- Encapsulation of services, classes and data types
- Uses XML for data representation

BENFITS OF ADO.NET

- Interoperability
- Scalability
- Productivity
- Performance

ADO.NET in .NET Framework



ADO existance?

- ADO still exists.
- ADO is tightly coupled to client server architectures
- Needs COM marshalling to pass data between tiers
- Connections and locks are typically persisted

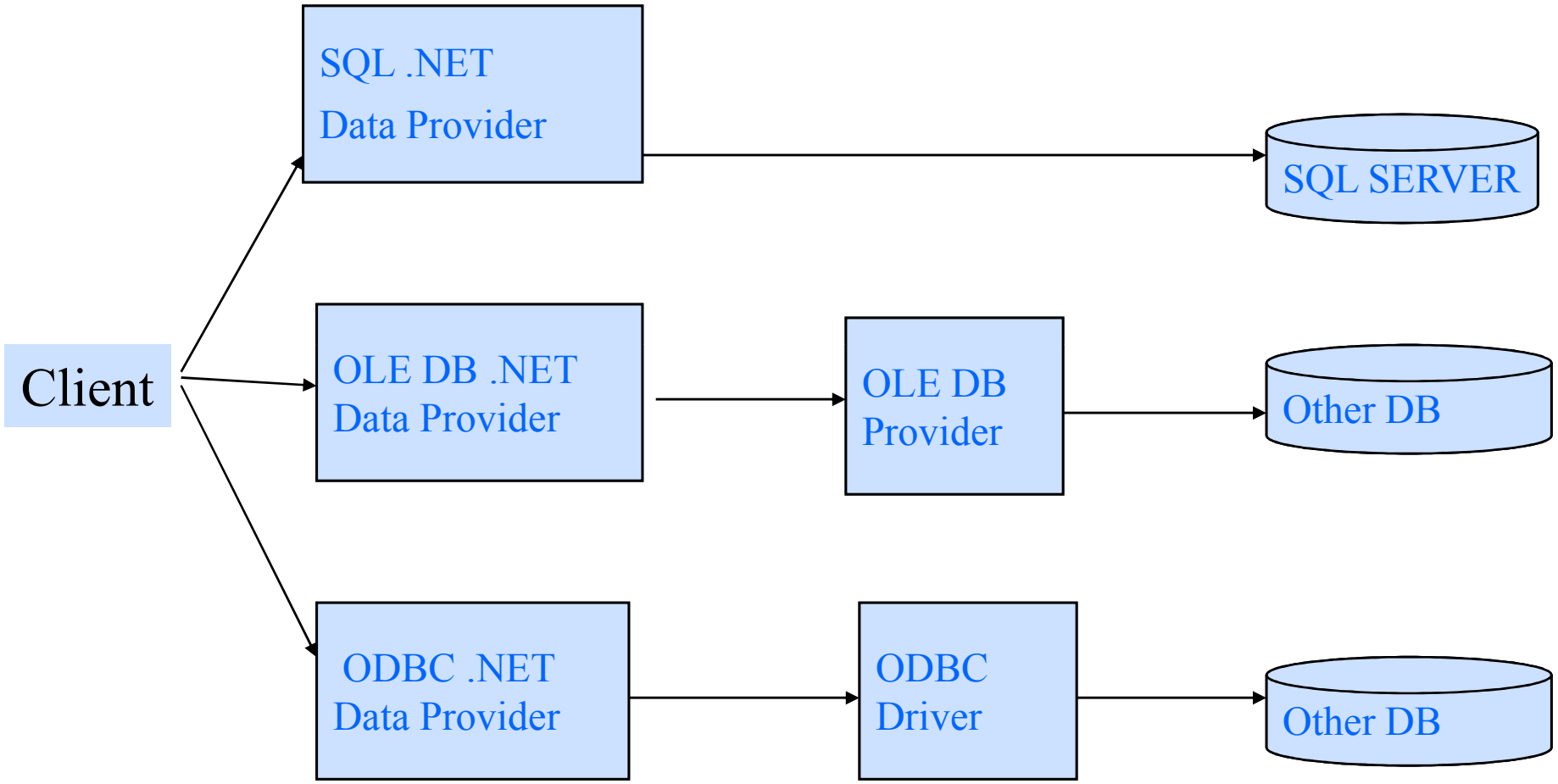
ADO Vs ADO.Net (Comparison)

Feature	**ADO**	**ADO.Net**
In memory data storage	Recordset object Mimics single table	Dataset object Contains DataTables
Data Reads	Sequential	Sequential or non-sequential
Data Sources	OLE/DB via the Connection object	Managed provider calls the SQL APIs

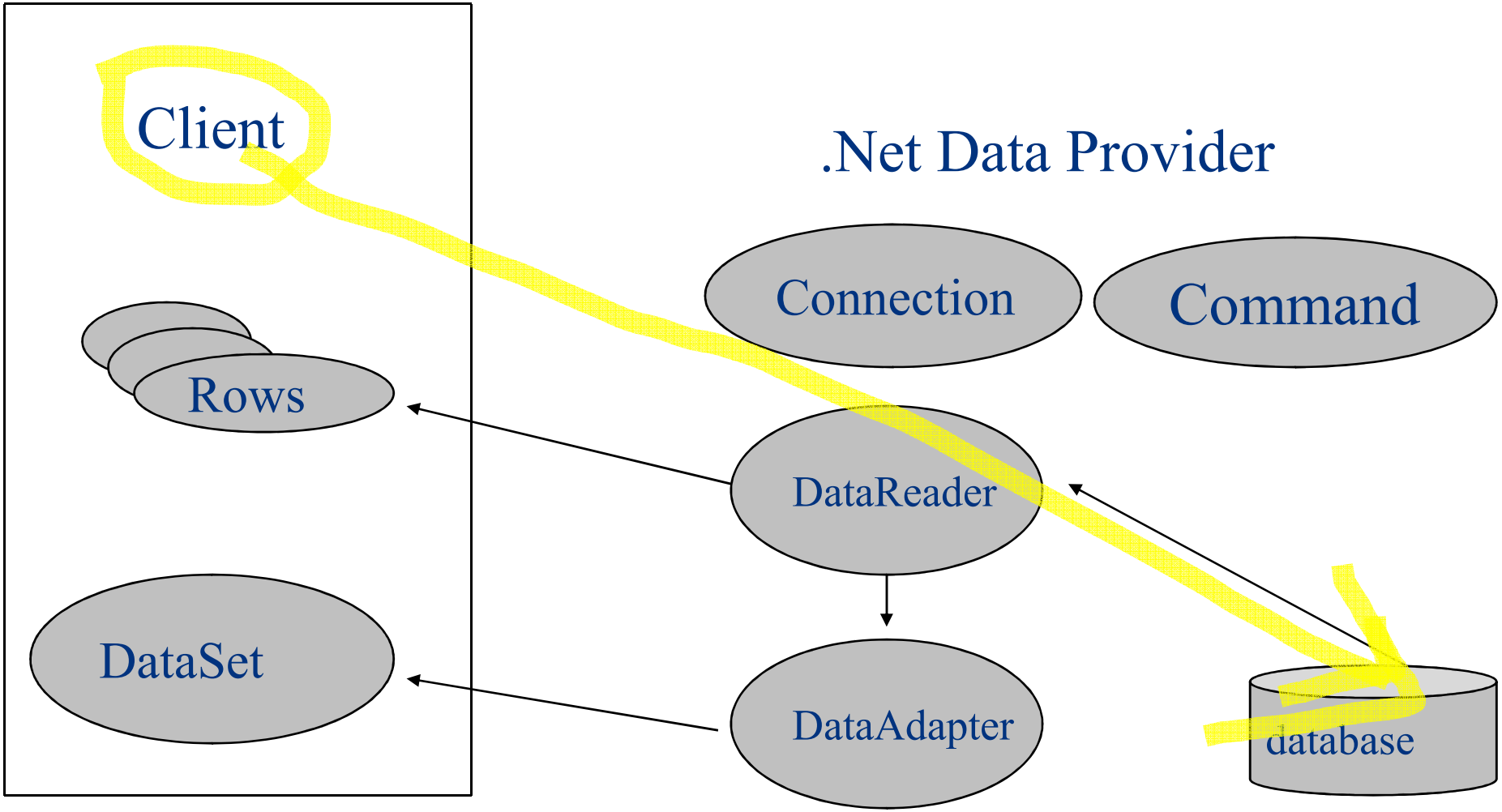
ADO Vs ADO.Net (Comparison)

Feature	**ADO**	**ADO.Net**
Disconnected data	Limited support, suitable for R/O	Strong support, with updating
Passing datasets	COM marshalling	DataSet support for XML passing
Scalability	Limited	Disconnected access provides scalability

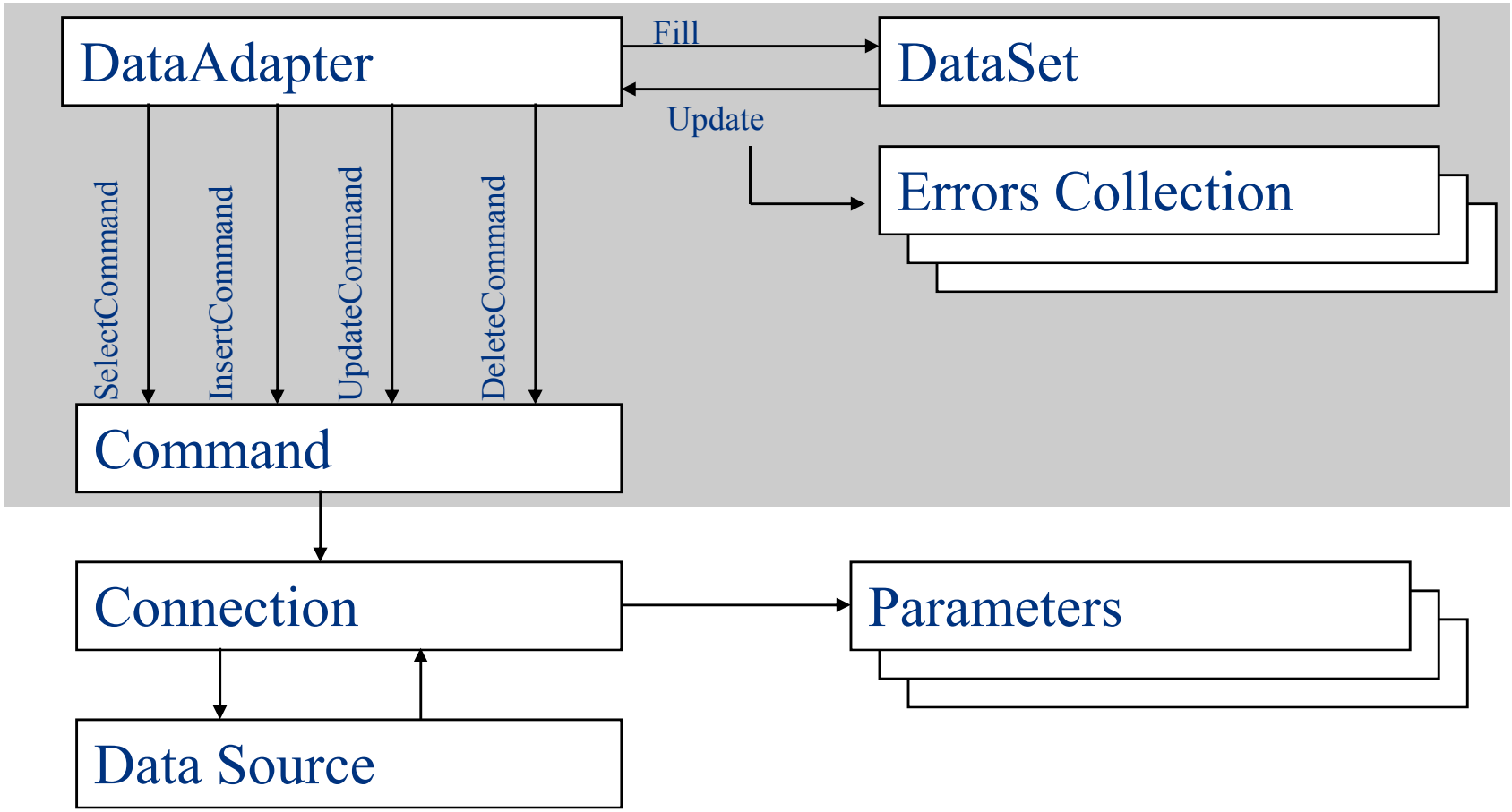
5.2 .NET Data Providers



Data Providers Functionality



ADO.Net object model



5.3 Connecting to a Database

- **Open Database Connectivity** (**ODBC**): a standard that allows ODBC-compliant applications to access any data source for which there is an ODBC driver
- ODBC uses SQL commands to access a database
  - ODBC then translates the SQL commands into a format that the database understands
- .NET includes strong support for ODBC
- .NET also allows to work directly with SQL Server and Oracle databases
  - Working directly provides faster access

Access SQL Server Databases

- **ActiveX Data Objects** (**ADO**): a Microsoft database connectivity technology that allows ASP and other Web development tools to access ODBC- and OLE-compliant databases
- **OLE DB**: a data source connectivity standard promoted by Microsoft (Eg. Microsoft Access DB)
  - Supports both relational and nonrelational data sources
- **ADO.NET**: most recent version of ADO that allows access to OLE DB-compliant data sources and XML

MS-Access Databases

- **Microsoft Data Access Components** (**MDAC**): components that make up Microsoft's Universal Data Access technology
  - Include ADO and OLE DB
- MDAC is installed with many Microsoft products, including Internet Explorer, Internet Information Services, Visual Studio, and the .NET Framework SDK

Namespaces in C#

- System.Data & System.Data.Common
- System.Data.SqlClient
- System.Data.OleDB
- System.Data.SqlTypes
- System.XML & System.XML.Schema

Using Namespaces in C#

- **VB.Net**

```
Imports System.Data
Imports System.Data.SqlClient
Dim sqlAdp as SqlDataAdapter
```

- **C#**

```
using System.Data;
using System.Data.SqlClient;
SqlDataAdapter sqlAdp= new
SqlDataAdapter();
```

- Use classes in the **`System.Data.SqlClient`** namespace to access and manipulate SQL Server databases

Understanding the `System.Data.SqlClient` Namespace (cont'd.)

Object	Description
`DataSet`	Represents data retrieved from a data source
`SqlCommand`	Executes a command, such as a SQL command, against a SQL Server database
`SqlConnection`	Provides access to a SQL Server database
`SqlDataAdapter`	Controls the interaction of a `DataSet` object with a SQL Server database
`SqlDataReader`	Returns read-only, forward-only data from a SQL Server database
`SqlError`	Contains error information returned from SQL Server
`SqlException`	Represents the exception that is thrown when an error or warning is returned from SQL Server

Core ADO.NET objects

Connecting to an SQL Server Database

- **`SqlConnection`** class: used to connect to an SQL Server database
  - Create an object from this class, passing in a connection string
- Connection string must include the **Data Source** parameter with the name of the SQL Server instance you wish to use

Connecting to an SQL Server Database (cont'd.)

Method	Description
`BeginTransaction()`	Begins a transaction
`ChangeDatabase()`	Changes the currently opened database
`Close()`	Closes a data source connection
`CreateCommand()`	Creates and returns a `Command` object associated with the `SqlConnection` object
`GetSchema()`	Returns schema information from the data source
`Open()`	Opens a data source connection
`ClearPool()`	Empties the `SqlConnection` object pool for the specified connection
`ClearAllPool()`	Empties all `SqlConnection` object pools

`SqlConnection` class methods

Connecting to an SQL Server Database (cont'd.)

Property	Description
`ConnectionString`	The string used to open a SQL Server database
`ConnectionTimeout`	The time to wait before abandoning a SQL Server database connection attempt
`Database`	The name of the current SQL Server database to use after a connection has been established
`DataSource`	The name of the SQL Server instance
`ServerVersion`	The SQL Server version to which the database is connected
`State`	A string indicating the current status of the SQL Server database connection

SQL Namespace Objects

- `using System.Data.SqlClient;`
    - SqlConnection
    - SqlCommand
    - SqlDataReader
    - SqlDataAdapter
    - SqlParameter
    - SqlParameterCollection
    - SqlError
    - SqlErrorCollection
    - SqlException
    - SqlTransaction
    - SqlDbType

Connecting to SQL

- ```
  using System.Data.SqlClient;

  string sConnectionString =
    "Initial Catalog=Northwind;
     Data Source=localhost;
     Integrated Security=SSPI;";

  SqlDataAdapter sqlAdp= new
  SqlDataAdapter(sConnectionString);

  sqlAdp.Close();
  sqlAdp.Dispose();
  ```
  ```

Connection Pooling

- ADO.Net pools connections.
  When you close a connection it is released back into a pool.
- ```
  SqlConnection conn = new SqlConnection();
  conn.ConnectionString =
    "Integrated Security=SSPI;Initial Catalog=student";
  conn.Open();  // Pool A is created.
  ```
- ```
  SqlConnection conn = new SqlConnection();
  conn.ConnectionString =
    "Integrated Security=SSPI;Initial Catalog=bsc";
  conn.Open();
  // Pool B is created because the connection strings differ.
  ```
- ```
  SqlConnection conn = new SqlConnection();
  conn.ConnectionString =
    "Integrated Security=SSPI;Initial Catalog=student";
  conn.Open(); // The connection string matches pool A.
  ```
  ```

Connecting to Database : Example

- **MS-Access : Example**

```
string conString
   = "Provider=Microsoft.Jet.OLEDB.4.0;Data
   Source=D:/essentials/jerald-
   notes/c#/practical/ex9/crud/course.mdb;";
OleDbConnection con
   = new OleDbConnection(conString);
con.Open();
```

- **Refer Example (Exercise-9)**

http://www.csresources.in/csharpnotes/csharp_ex9.htm

Opening and Closing a Data Source

- After creating a **`SqlConnection`** object, **`Open()`** method is used to open the specified SQL Server database instance
- **`Close()`** method disconnects the database connection
  - Database connections do not automatically close when a C# program ends
- Must verify that program has successfully connected to a database before attempting to use it
- **`State`** property of the **`SqlConnection`** class: indicates the current status of the database connection

Checking the Database Connection (cont'd.)

Value	Description
Broken	The connection is broken
Closed	The connection is closed
Connecting	The `Connection` object is connecting to the data source
Executing	The connection is executing a command
Fetching	The connection is retrieving data
Open	The connection is open

`SqlConnection` class **`State`** property values

5.4 Using the command object

- SqlCommand
  Multiple constructors

```
1.New()
2.New(cmdText)
3.New(cmdText, connection)
4.New(cmdText, connection,
      transaction)
5.And many more
```

Using the command object

```
string sSelectQuery =
  "SELECT * FROM Categories ORDER BY CategoryID";
string sConnectionString =
  "Initial Catalog=Northwind;
   Data Source=localhost;
   Integrated Security=SSPI;";
SqlConnection objConnect = new SqlConnection(sConnectString);
SqlCommand objCommand = new     SqlCommand(sSelectQuery,
                                           objConnect);
/*
objCommand.CommandTimeout = 15;
objCommand.CommandType = CommandType.Text;
*/

objConnect.Open();

SqlDataReader drResults;
drResults = objCommand.ExecuteReader()

drResults.Close();
objConnect.Dispose();
```

Command Methods

- .ExecuteReader() - Returns DataReader
- .ExecuteNonQuery() - **Returns # of Rows Affected**
- .ExecuteXMLReader() - Returns XMLReader Object to Read XML documentation
- .ExecuteScaler() - Returns a Single Value e.g. SQL SUM function.

The DataReader object

- DataReader objects are highly optimised for fast, forward only enumeration of data from a data command
- A DataReader is **not** disconnected
- Access to data is on a per record basis.
- Forward only
- Read only
- Does support multiple recordsets

5.5 Reading data

- SqlCommand
  - ExecuteReader
  - ExecuteNonQuery
  - ExecuteScalar
  - ExecuteXMLReader
- SqlDataAdapter
  - DataSet

Retrieving Records with the `SqlDataReader` Class

- **`SqlCommand`** class: used to execute commands against Microsoft SQL Server version 7.0 or later
- Syntax:

```
SqlCommand object = new SqlCommand
                    ("command", connection)
```

  - ***`command`*** parameter: contains the SQL command to be executed
  - ***`connection`*** parameter: represents the **`SqlConnection`** object used to connect to the database

Retrieving Records with the `SqlDataReader` Class (cont'd.)

- **`DataReader`** object: used to retrieve read-only, forward-only data from a data source
- **Forward-only**: the program can only move forward sequentially through the records in the returned data from the first to the last
- Use a **`DataReader`** object when you want to read data but not add, delete, or modify records
- **`SqlDataReader`** class: used to retrieve data from SQL Server

Retrieving Records with the `SqlDataReader` Class (cont'd.)

- **`ExecuteReader()`** method of the **`SqlCommand`** class: creates a **`SqlDataReader`** object
  - Must assign the **`SqlDataReader`** object to a variable
- **`Read()`** method of the **`SqlDataReader`** class: advances the **`SqlDataReader`** object to the next record
- **Cursor**: your position within the recordset
  - Initially placed before the first row in the recordset
  - First use of the **`Read()`** method places the cursor in the first row of the recordset

Creating a data reader

```
SqlDataReader sqlReader;
sqlReader =
  sqlCommand.ExecuteReader();
while (sqlReader.Read())
{
  // process, sqlReader("field")
}
sqlReader.Dispose();
```

Other Methods for Reading

- GetString(), GetInt() etc.
- GetSqlString(), GetSqlInt32() etc.
- GetValues()
- IsDBNull()
- GetSchemaTable()

5.6 DataSets

- In-memory representation of data contained in a database/XML
- Operations are performed on the DataSet, not the data source
- Client Programs(C#) accesses the Dataset
- Can be created programmatically, using a DataAdapter or XML schema and document (or any mixture)
- Query from a table will create a dataset

Creating DataSets

- Setup SqlConnection
- Setup a SqlDataAdapter
- Create a DataSet
- Call the .Fill() method on the DataAdapter

5.7 DataAdapters

- Pipeline between DataSets and data sources
- Geared towards functionality rather than speed
- Disconnected by design
- Supports select, insert, delete, update commands and methods
- Must always specify a select command
- All other commands can be generated or specified

Using the DataAdapter

```
 string conString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=D:/ex10/crud/course.mdb;";
OleDbConnection con = new OleDbConnection(conString);
OleDbCommand cmd;
OleDbDataAdapter adapter;
DataTable dt = new DataTable();
con.Open();
adapter = new OleDbDataAdapter(cmd);
UpdateCommand = con.CreateCommand();
adapter.UpdateCommand.CommandText = sql;
if (adapter.UpdateCommand.ExecuteNonQuery() > 0)
                    {
                    clearTxts();
                    MessageBox.Show(@"Successfully Updated");
                }
```

Refer (Click)
Exercise 10

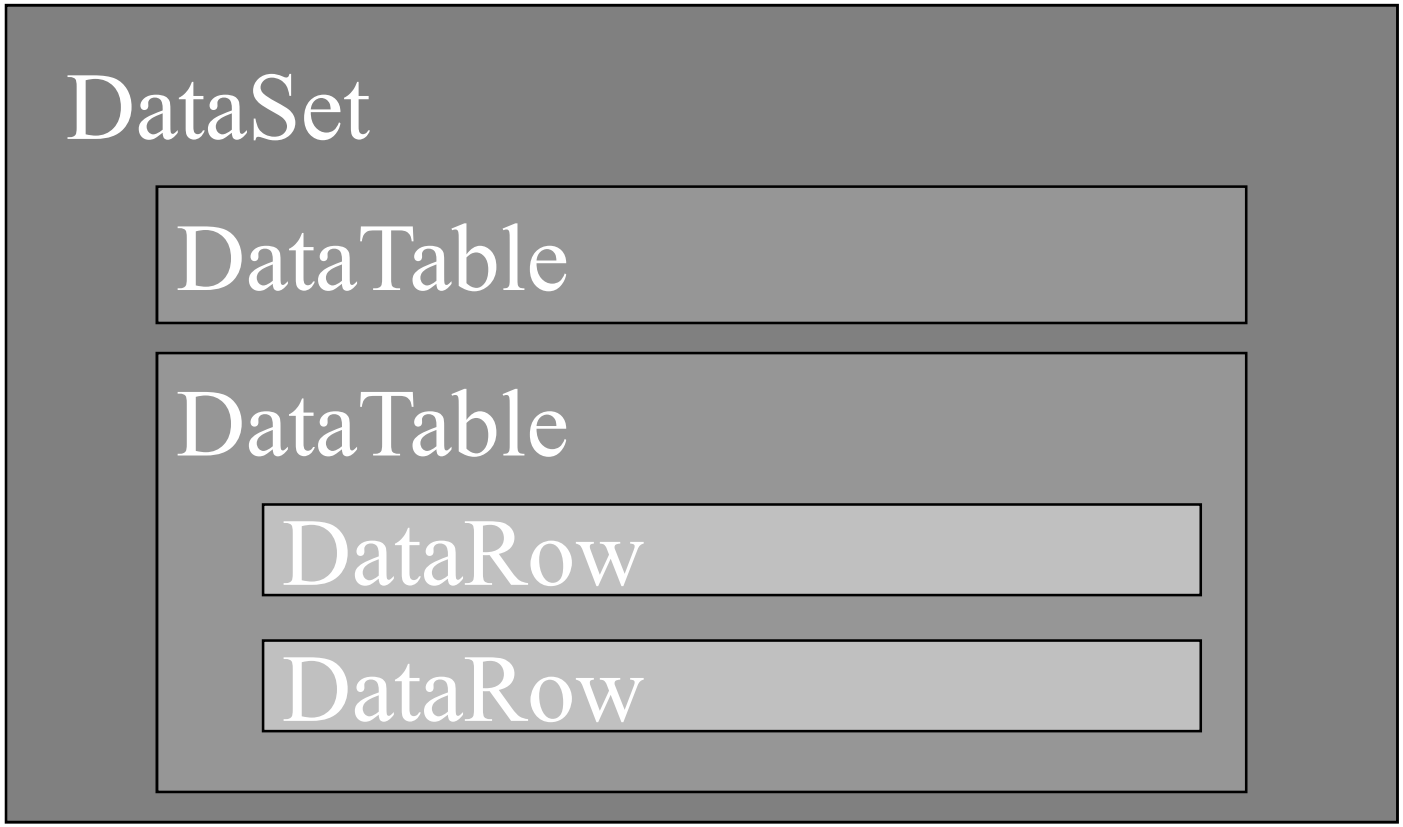
DataAdapters

- For speed and efficiency you should set your own InsertCommand, UpdateCommand and DeleteCommand
- Call **GetChanges** to seperates the updates, adds and deletes since the last sync.

DataTables

- A DataSet contains one or more DataTables.
- Fields are held within the DataTable.
- And in DataRows(Records), DataColumns(Fields).

Sets, Tables and Rows



Using DataTables

With a DataTable we can

- Insert, modify and update
- Search
- Apply views
- Compare
- Clear
- Clone and Copy

DataRelations

- New to ADO.Net
- Tables within a DataSet can now have relationships, with integrity.
- Supports cascading updates and deletes.

DataViews

- Like a SQL view
- Single, or multiple tables
- Normally used with GUI applications via Data Binding.

Create Database

- **Prerequisites:** Microsoft SQL Server Management Studio
- Open Microsoft SQL Server Management Studio and write the below script to create a database and table in it.
  - *create database Demodb;*
  - *use Demodb;*
  - *CREATE TABLE demo(articleID varchar(30) NOT NULL PRIMARY KEY, articleName varchar(30) NOT NULL,);*
  - *insert into demo values(1, 'C#');*
  - *insert into demo values(2, 'C++');*

Connecting with a Database

- **Connecting C# with Database:** The connection to a database normally consists of the following parameters.
  - ***Database name or Data Source:*** The database name to which the connection needs to be set up and connection can be made or you can say only work with one database at a time.
  - ***Credentials****:* The username and password which needs to be used to establish a connection to the database.
  - ***Optional Parameters****:* optional parameters provide more information on how .NET should connect to the database to handle the data.

Connection – Sample Code

```
// C# code to connect the database
using System;
using System.Data.SqlClient;
namespace Database_Operation {
class DBConn {
    // Main Method
    static void Main()
    {  Connect();
        Console.ReadKey();
    }

    static void Connect()
    {     string constr;
          // for the connection to sql server database
        SqlConnection conn;
          // Data Source is the name of the  server
          // The Initial Catalog is used to specify
          // the name of the database
// The UserID and Password are
     credentials  required to connect to
     database.
       constr = @"Data Source=DESKTOP-
     GP8F496;Initial Catalog=Demodb;User
     ID=sa;Password=abc123";

       conn = new SqlConnection(constr);
         // to open the connection
       conn.Open();
  Console.WriteLine("Connection Open!");
  // to close the connection
       conn.Close();
    }
}
}
```

Insert – Example Code

```
using System;
using System.Data.SqlClient;
namespace Database_Operation {
class InsertStatement {
    static void Main()
    {   Insert();
        Console.ReadKey();
    }

    static void Insert()
    {
string constr;
// connection to sql server database
SqlConnection conn;
constr = @"Data Source=D:/ex1/GP8F496;
Initial Catalog=Demodb;User ID=sa;Password=abc123";
conn = new SqlConnection(constr);
  // to open the connection
       conn.Open();
```

```
//to perform read and write operations
       SqlCommand cmd;

// data adapter object is use to
// insert, update or delete commands
SqlDataAdapter adap = new SqlDataAdapter();
string sql = "";

sql = "insert into demo values(3, 'Python')";
//execute the sql command
cmd = new SqlCommand(sql, conn);

// associate the insert SQL command to adapter
adap.InsertCommand = new SqlCommand(sql,
      conn);

//execute the DML statement in database
adap.InsertCommand.ExecuteNonQuery();
// closing all the objects
cmd.Dispose();
conn.Close();
} }}
```

Update – Code Example

```
sql = "update demo set articleName=chemical' where
    articleID=3";
cmd = new SqlCommand(sql, conn);

// associate the insert SQL  command to adapter
adap.InsertCommand = new SqlCommand(sql, conn);

// use to execute the DML statement against
adap.InsertCommand.ExecuteNonQuery();
```

Delete – Code Example

```
SqlConnection conn;

constr = @"Data Source=D:/ex1/GP8F496;
Initial Catalog=Demodb;User
      ID=sa;Password=abc123";
conn = new SqlConnection(constr);

// to open the connection
 conn.Open();
SqlCommand cmd;

// data adapter object is use to
// insert, update or delete commands
SqlDataAdapter adap = new SqlDataAdapter();
string sql = "";
```

```
// define SQL statement
sql = "delete from demo where articleID=3";

// execute the sql command
       cmd = new SqlCommand(sql, conn);

// associate the insert SQL
// command to adapter
adap.InsertCommand = new SqlCommand(sql, conn);

//execute the DML statement
adap.InsertCommand.ExecuteNonQuery();
```

Handling SQL Server Errors

- Must handle situations that occur when you cannot connect to a database server
- Connection may fail because:
  - The database server is not running
  - You have insufficient privileges to access the data source
  - You entered an invalid username and password
- Other causes of errors:
  - You are trying to open a nonexistent database
  - You entered an invalid SQL statement

Using Exception Handling to Control SQL Server Errors

- Place the **`Open()`** method within a **`try`**…**`catch`** block to trap connection errors
- **`SqlException`** class:
  - Part of the **`System.Data.SqlClient`** namespace
  - Represents the exception that is thrown when SQL Server returns an error or warning
  - **`Number`** and **`Message`** properties provide an error code and message for the exception

5.8 Executing SQL Commands

- **`System.Data.SqlClient`** namespace contains classes to access and manipulate SQL Server databases:
  - **`SqlDataReader`** class
  - **`SqlCommand`** class

Retrieving Records with the `SqlDataReader` Class (cont'd.)

- Use the **`Read()`** method to determine if a next record is available
  - Returns true if there is another row in the recordset
- Field names in a database table are assigned as variables in a **`SqlDataReader`** object collection
  - Content of each variable changes when the cursor position moves to a new row

Retrieving Records with the `SqlDataReader` Class (cont'd.)

- Use the **`Close()`** method of the **`SqlDataReader`** class to close it when you are finished working with it
  - **`SqlDataReader`** has exclusive access to the connection object
  - You cannot access any other commands until the **`SqlDataReader`** object is closed

Executing SQL Commands with the `SqlCommand` Object

- **`ExecuteNonQuery()`** method of the **`SqlCommand`** object: executes commands against a database
  - Used for inserting, updating, or deleting rows in a SQL Server database
  - Does not return a recordset of data

CRUD USING C#

Creating and Deleting Databases

- Use the **`CREATE DATABASE`** statement with the **`ExecuteNonQuery()`** method to create a new database
  - If database already exists, an error will occur
- Can test if the database exists with the **`ChangeDatabase()`** method in a **`try`**…**`catch`** block
  - If unsuccessful, can create the database in the **`catch`** block
- Use the **`DROP DATABASE`** statement with the **`ExecuteNonQuery()`** method to delete a database

Creating and Deleting Tables

- Use the **`CREATE TABLE`** statement with the **`ExecuteNonQuery()`** method to create a new table
- Must select the correct database with the **`SqlConnection`** constructor or with the **`ChangeDatabase()`** method before executing the **`CREATE TABLE`** statement
- Can use the **`ExecuteReader()`** or **`ExecuteNonQuery()`** methods to determine whether the table already exists

Creating and Deleting Tables (cont'd.)

- IDENTITY keyword: used with a primary key to generate a unique ID for each row in a new table
  - First row's identity value is 1
  - Each subsequent row's identity value increases by 1
- You can specify a start value and the increment value if desired
- When adding records to a table with an IDENTITY field, do not include a field value for the IDENTITY field
- Use the **`DROP TABLE`** statement with the **`ExecuteNonQuery()`** function to delete a table

Adding, Deleting, and Updating Records

- Use the **`INSERT`** and **`VALUES`** keyword with the **`ExecuteNonQuery()`** method to add a record
  - Values in the **`VALUES`** list must be in the same order in which the fields were defined in the table
  - Specify **`NULL`** in any field for which you do not have a value
- Use the **`BULK INSERT`** statement and the **`ExecuteNonQuery()`** method to add multiple records using data in a local text file

Adding, Deleting, and Updating Records (cont'd.)

- Use the **`UPDATE`**, **`SET`**, and **`WHERE`** keywords with the **`ExecuteNonQuery()`** method to update records in a table
  - **`UPDATE`** keyword specifies the table name
  - **`SET`** keyword assigns values to fields
  - **`WHERE`** keyword specifies which records to update
- Use the **`DELETE`** and **`WHERE`** keywords with the **`ExecuteNonQuery()`** method to delete records in a table
  - To delete all records in a table, omit the **`WHERE`** keyword

Summary

- Open Database Connectivity (ODBC) allows ODBC-compliant applications to access any data source
- ODBC driver is required for data access
- ActiveX Data Objects (ADO) technology that allows C# to access ODBC and OLE DB-compliant databases
- **`System.Data.SqlClient`** namespace is used to access and manipulate SQL Server databases
- **`SqlConnection`** class is used to connect to a SQL Server database
- **`State`** property of the **`SqlConnection`** class is used to determine the current status of the database connection

Summary (cont'd.)

- Use the **`SqlException`** class to handle errors
- Use the **`SqlCommand`** class to execute commands against SQL Server
- **`ExecuteReader()`** method with a **`DataReader`** object is used to retrieve data from a SQL Server data source
- **`ExecuteNonQuery()`** method of the **`SqlCommand`** class executes commands against a database
- **`CREATE DATABASE`** statement with **`ExecuteNonQuery()`** method is used to create a new database
- Use the **`CREATE TABLE`** statement with the **`ExecuteNonQuery()`** method to create a new table

Summary (cont'd.)

- Use the **`IDENTITY`** keyword with a primary key to generate a unique ID for each new row in a table
- Use the **`DROP TABLE`** statement with the **`ExecuteNonQuery()`** method to delete a table
- Use the **`INSERT`** and **`VALUES`** keywords with the **`ExecuteNonQuery()`** method to add a new record to a table
- Use the **`BULK INSERT`** statement with the **`ExecuteNonQuery()`** method and a local text file to add multiple new records to a table

Summary (cont'd.)

- Use the **`UPDATE`**, **`SET`**, and **`WHERE`** keywords with the **`ExecuteNonQuery()`** method to update records in a table
- Use the **`DELETE`** and **`WHERE`** keywords with the **`ExecuteNonQuery()`** method to delete records in a table

REFERENCES

- ADO.Net Programmer's Reference
  Bilbija, Dickenson et al. Wrox Press
- http://msdn.microsoft.com
- http://www.csharphelp.com/
- http://www.csharp-station.com/
- http://www.csharpindex.com/
- http://www.c-sharpcorner.com/
- https://www.w3schools.com/cs/
- https://www.javatpoint.com/c-sharp-tutorial
- https://www.bestprog.net/en/site-map/c/