Microsoft® ADO.NET Summary

Abhishek SHARMA

Contents

- What is ADO.Net?
- What happened to ADO?
- The ADO.Net object structure
- Connecting
- Commanding
- Readers and DataSets

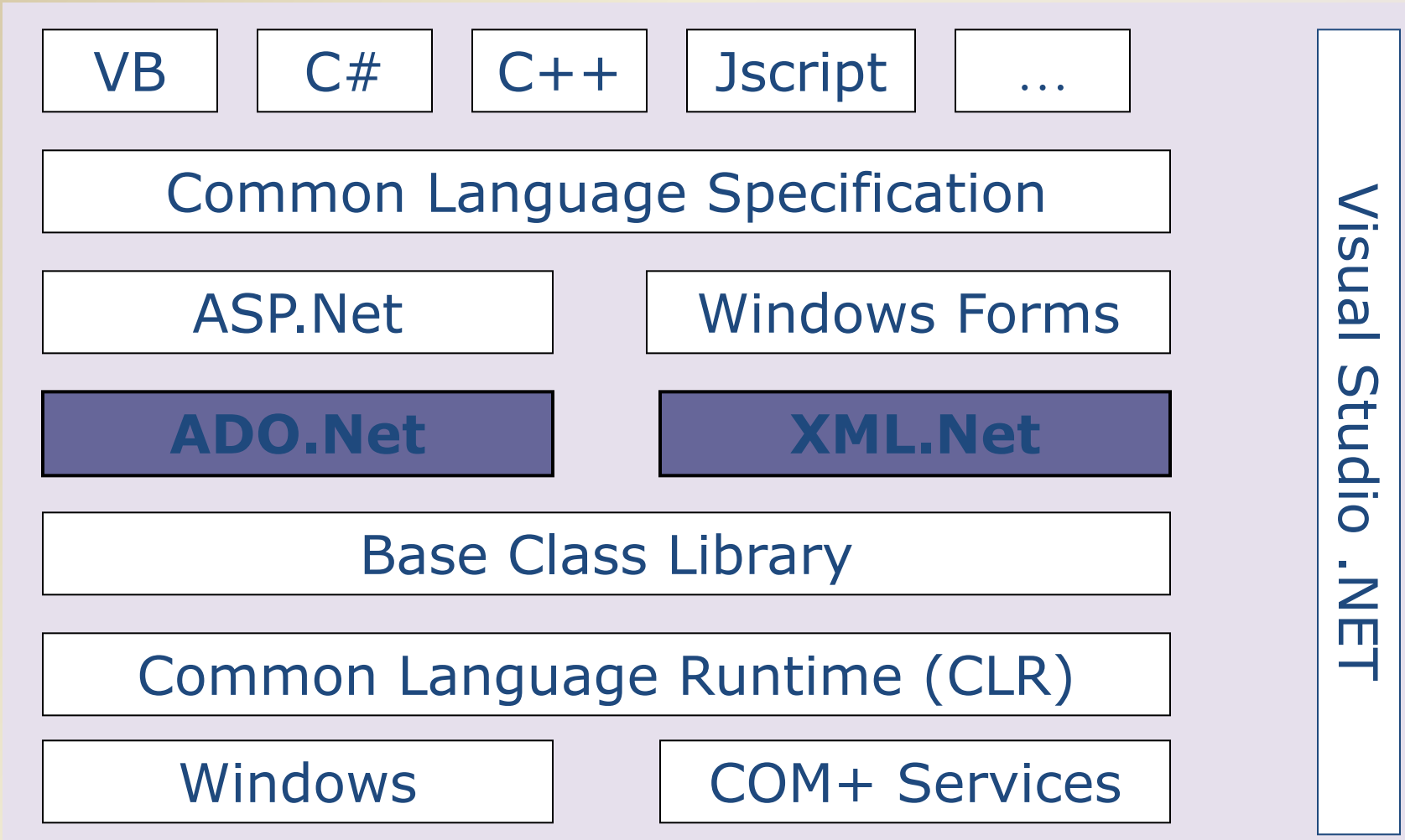
What is ADO.Net?

- The data access classes for the .Net framework
- Designed for highly efficient data access
- Support for XML and disconnected record sets

And the .Net framework?

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

Where does ADO sit?



What happened to ADO?

- 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

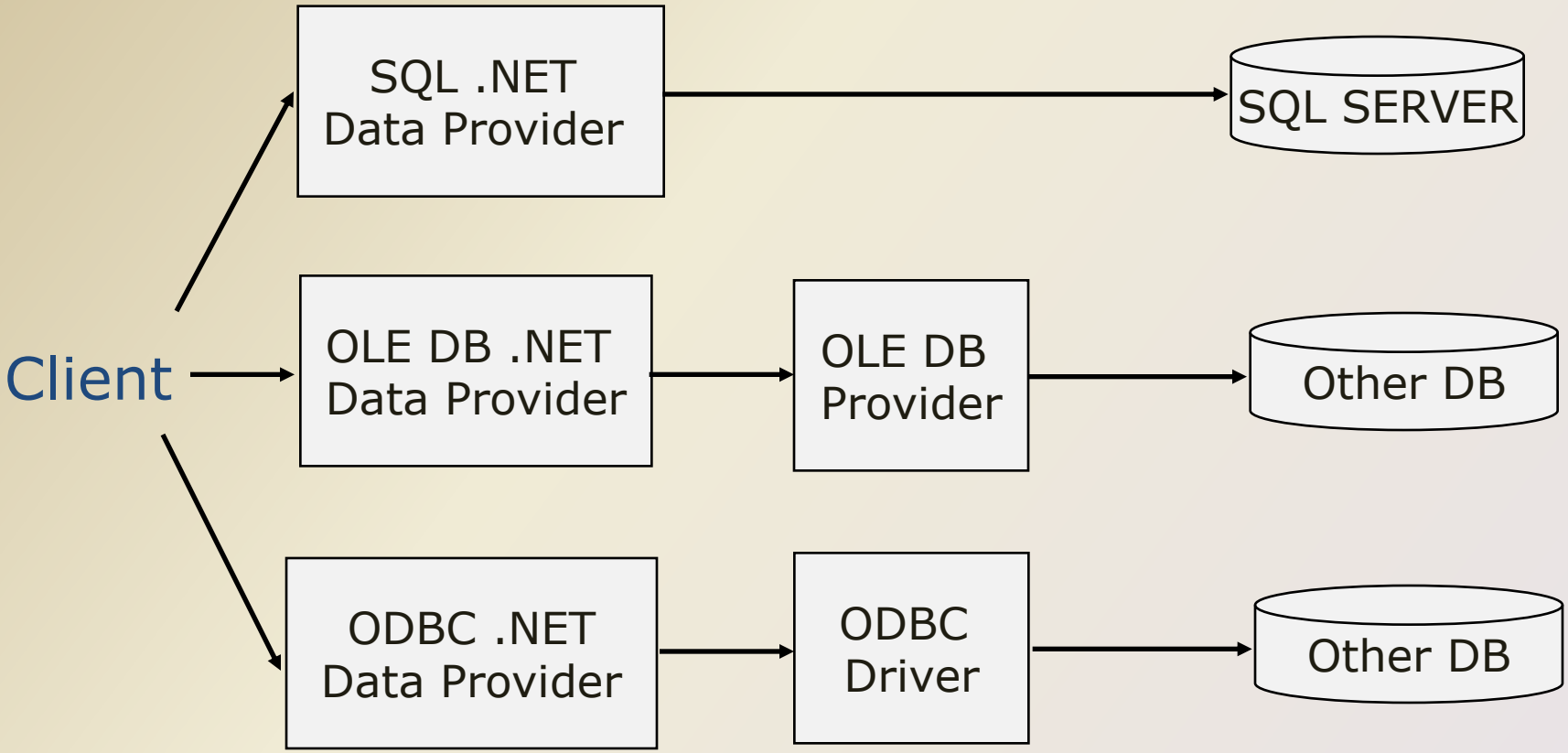
ADO / ADO.Net Comparisons

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

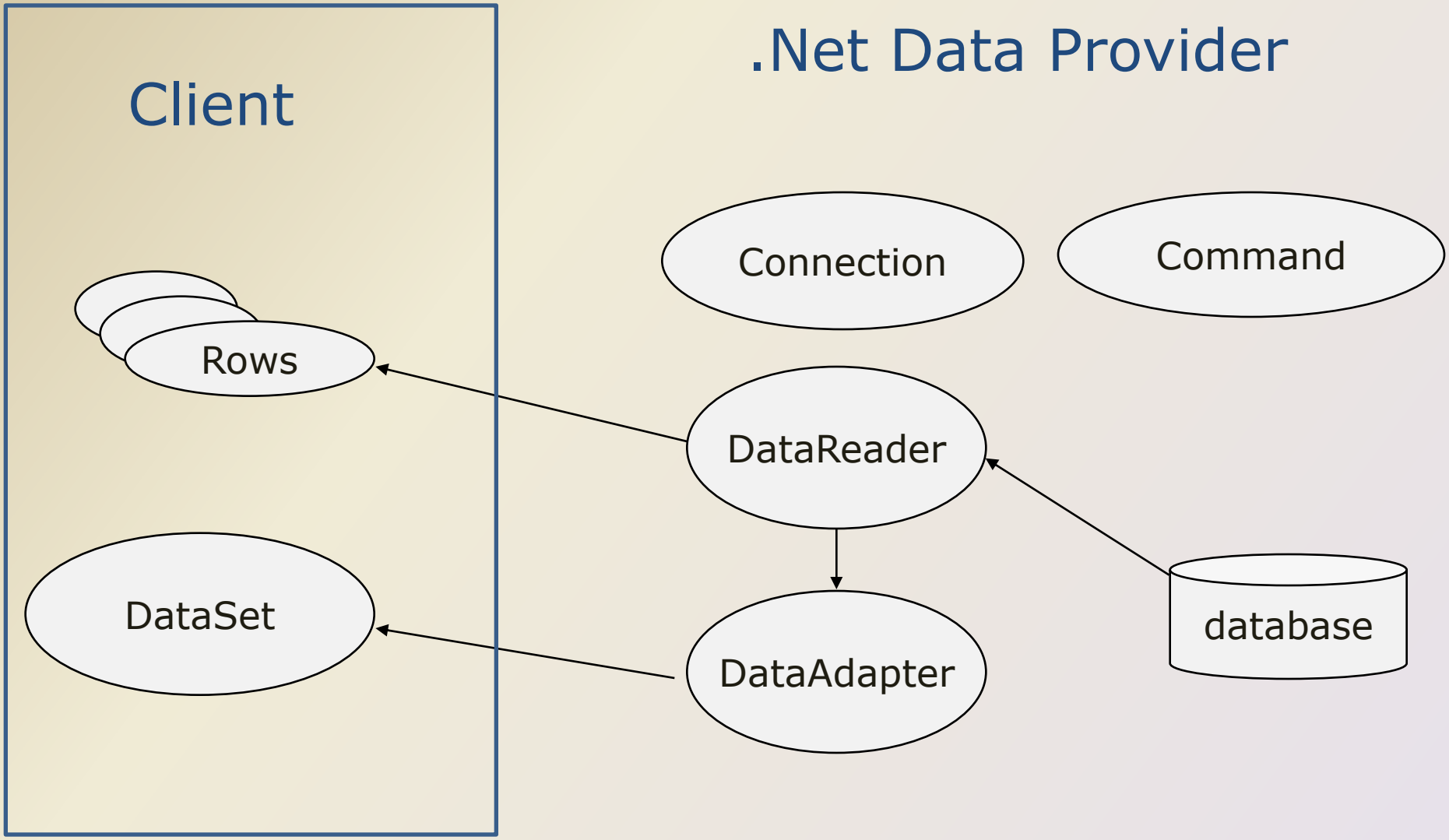
ADO / ADO.Net Comparisons

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

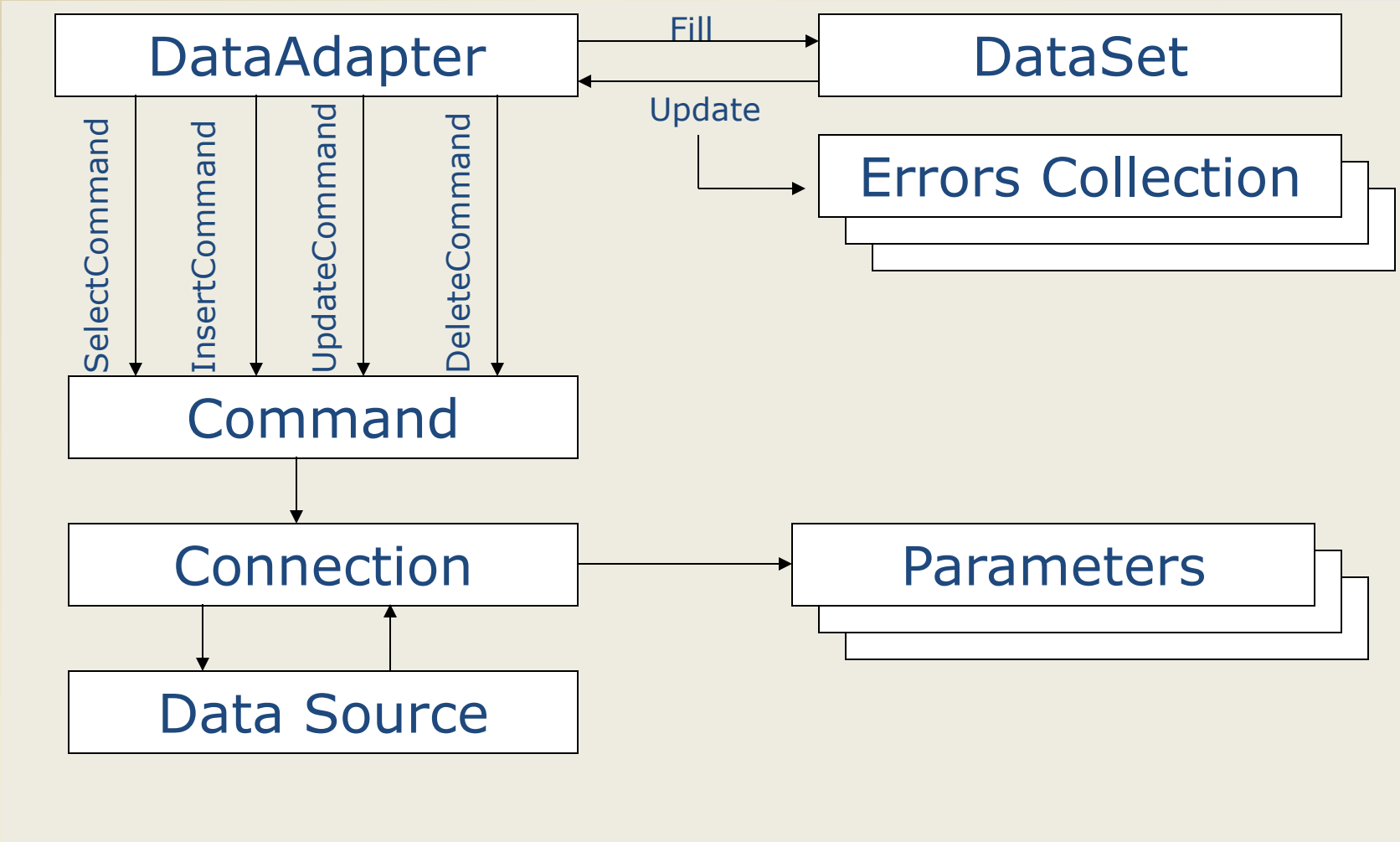
.NET Data Providers



Data Provider Functionality



ADO.Net object model



Namespaces

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

Using Namespaces

- 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();
```

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=northwind";
  conn.Open();  // Pool A is created.
  ```
- ```
  SqlConnection conn = new SqlConnection();
  conn.ConnectionString =
    "Integrated Security=SSPI;Initial Catalog=pubs";
  conn.Open();
  // Pool B is created because the connection strings differ.
  ```
- ```
  SqlConnection conn = new SqlConnection();
  conn.ConnectionString =
    "Integrated Security=SSPI;Initial Catalog=northwind";
  conn.Open(); // The connection string matches pool A.
  ```
  ```

Getting data

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

Using the command object

- SqlCommand
  Multiple constructors
- `New()`
- `New(cmdText)`
- `New(cmdText, connection)`
- `New(cmdText, connection, transaction)`

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

The DataReader object

- Access to data is on a per record basis.
- Forward only
- Read only
- Does support multiple recordsets

Creating a data reader

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

Other Methods

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

DataSets

- In-memory representation of data contained in a database/XML
- Operations are performed on the DataSet, not the data source
- Can be created programmatically, using a DataAdapter or XML schema and document (or any mixture)

Creating DataSets

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

DataAdapters

- Pipeline between DataSets and data sources
- Geared towards functionality rather than speed
- Disconnected by design
- Supports select, insert, delete, update commands and methods

DataAdapters

- Must always specify a select command
- All other commands can be generated or specified

Using the DataAdapter

```
SQLDataAdapter sqlDA =
  new SqlDataAdapter();

sqlDA.SelectCommand =
  new SqlCommand ("select * from
  authors“, sqlConnection);

DataSet sqlDS = new
  DataSet("authorsTable");
sqlDA.Fill(sqlDS, "authorsTable");
```

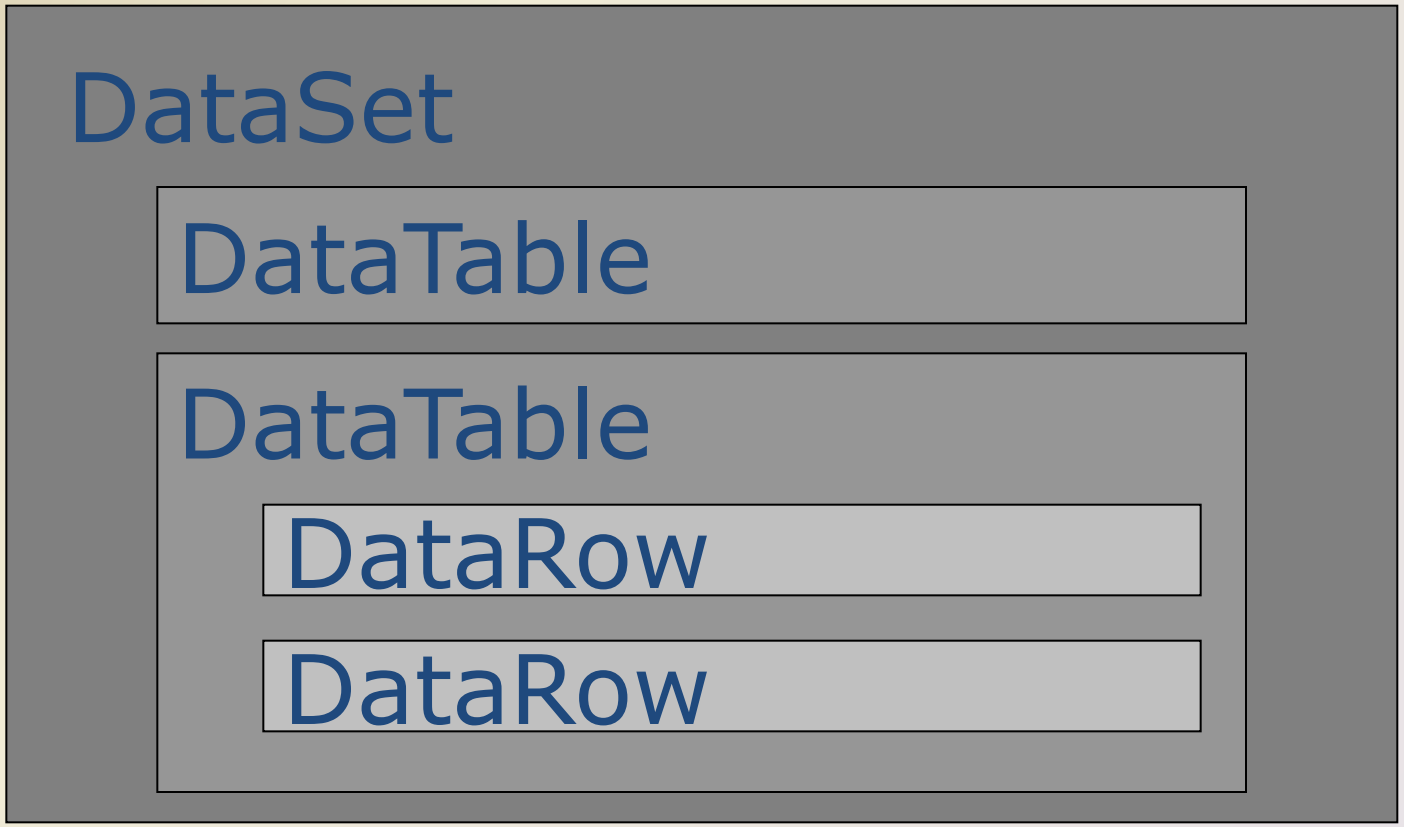
DataAdapters

- 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. Then sync each type.

DataTables

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

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.