Connection , Command , DataAdapter , DataReader is DataProvider specific => all are sealed class, inherited from DbConnection class, and implement the ICloneable interface

**DataSet:**

the output will be stored in dataset and work independently of the data provider

we use dataset in the application

**ADO.NET connection class :**

System.Data.SqlClient namespace

establish the connection to the server

Always close the connection explicitly

Connection is the most expensive resources

sealed class, inherited from DbConnection class, and implement the ICloneable interface

**constructor**:

SqlConnection(): = (used to initialize the new instance of connection) empty connection string

SqlConnection(String connectionString): = return the conncetion

SqlConnection(String connectionString, SqlCredential credential): = if in connection string (Integrated Security = true ) is not true then we have to give the credential in credential

**In connectionString :**

Data source=. ; =>(local host (means = .)), if server is on network then (NEED NAME OR IP ADDRESS)

**problem of hard-coding the connection string**

* move your database to a different server => change applicationcode => rebuild the application => redeployment (timeconsuming)
* multiple places => changes conn string at all places => maintenance overhead ( error-prone)
* In real-time applications, while developing you may point to your Development database. While moving to UAT, you may have a different server for UAT and in a production environment, you need to point to the production database

**Solve the problem of hard-coding the connection string**

store the connection string in web.config.( appsetting.json)

**In Web.config** ( ConfigurationManager is used to access)

<connectionStrings>

<add name="ConnectionString"

connectionString="data source=.; database=student; integrated security=SSPI"

providerName="System.Data.SqlClient" />

</connectionStrings>

ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;

**ADO.NET SqlCommand class**

used to execute the sql statements against sql database

**constructor**

* SqlCommand(): initialize a new instance of the System.Data.SqlClient.SqlCommand class.
* SqlCommand(string cmdText):
* SqlCommand(string cmdText, SqlConnection connection):
* SqlCommand(string cmdText, SqlConnection connection, SqlTransaction transaction):
* SqlCommand(string cmdText, SqlConnection connection, SqlTransaction transaction, SqlCommandColumnEncryptionSetting columnEncryptionSetting):

1. cmdText = query or stored procedure name
2. connection = connection string
3. transaction = SqlTransaction (Commit , RollBack)
4. columnEncryptionSetting = encryption setting

**Methods**

1. BeginExecuteNonQuery(): Asynchronous Execution of the sql statement
2. Cancel(): cancel the execution
3. Clone(): a new System.Data.SqlClient.SqlCommand object is a copy of the current instance.
4. CreateParameter(): creates a new instance of the paramaeter
5. ExecuteReader(): command send to connection and builds sqlDataReader(for reading the output data) (When your T-SQL statement returns more than a single value)
6. ExecuteScalar(): fetched all but returns the first column of the first row(return type is object)
7. ExecuteNonQuery(): exucute the query and return the numbers of rows affected ( Insert, Update or Delete)
8. Prepare(): ??
9. ResetCommandTimeout(): CommandTimeout property to its default value. ??

**ADO.NET SqlDataReader Class (increases the performance , reduces the system overheads)**

used to read data from database

read in forward-only direction(once read then no way to back to read previous record)

connection-oriented( equires an open or active connection to data source)

The data is available as long as the connection with the database exists

read-only(not possible to change the data)

SqlDataReader sdr = new SqlDataReadere();

sdr[0] = first column of the select query

**properties(Variables)**

1. Connection: to get the connection
2. Depth: depth of nesting for the current row ??
3. FieldCount: number of columns in the current row
4. HasRows:to check does it contains one or more rows
5. IsClosed: returns boolean value to show instance is closed or not
6. RecordsAffected: number of the rows affected by query
7. VisibleFieldCount: count of visible fields
8. Item[String]: for getting value of column in string
9. Item[Int32]: for getting value of column in integer

**Methods:**

1. Close(): close datareader
2. GetBoolean(int i): value of the specified column as a Boolean( i is the number of column starting from 0 index)
3. GetByte(int i): Value of specified column in byte
4. GetChar(int i): Value of specified column in a single character
5. GetDateTime(int i): Value of specified column in System.DateTime object
6. GetDecimal(int i): Value of specified column in System.Decimal objec
7. GetDouble(int i): Value of specified column in a double-precision floating-point number (ex : 10.20)
8. GetFloat(int i): Value of specified column in a single-precision floating-point number( ex: 10.2)
9. GetName(int i): get the name of specified column
10. GetSchemaTable(): Value of specified column in byte
11. GetValue(int i): get the value of parameter in the native formate
12. GetValues(object[] values): an array of objects with the column values of the current row(to get all column values of a raw)
13. NextResult(): advances the data reader to the next sqlstatement  
      
     returns the true or false as read()  
    (   
       
    select \* from tblUser;select \* from tblAdmin;  
      
    nextresult send the pointer to next statement (in our example it will send the pointer to get all the data of admin table)  
      
     while (reader.NextResult())  
     {  
     Console.WriteLine("\nSecond Result Set:");  
     //Looping through each record  
     while (reader.Read())  
     {  })
14. Read(): REad the next row and return the true if there is a row next to it.

**ADO.NET SqlDataAdapter**

bridge between a DataSet or DataTable and a Data Source

used to retrieve the data

fill the DataSet or DataTable and also update the data source

**Constructor**

1. SqlDataAdapter(): ==> a new instance of the SqlDataAdapter class
2. SqlDataAdapter(SqlCommand selectCommand): ==> new instance of the sqlDataAdpter with given command
3. SqlDataAdapter(string selectCommandText, string selectConnectionString): ==> new instance of the sqlDataAdpter with given command and connection string
4. SqlDataAdapter(string selectCommandText, SqlConnection selectConnection): ==> same as above but here we use SqlConnection class to add userId and Password more securely compare to above.

* selectCommand = select command or stored procedure
* selectConnectionString = connection string
* SqlConnectionClass to set userId and password(if IntegratedSecurity = true )

**Methods of SqlDataAdapter:**

1. CloneInternals(): create the copy of this instance of DataAdapter
2. Dispose(Boolean): release Unmanaged resources used by the DataAdapter ??
3. Fill(DataSet): add rows in the DataSet , automatically open , excute, retrieve, fill and close the connection,
4. FillSchema(DataSet, SchemaType, String, IDataReader):
5. GetFillParameters(): get the parameters set by the user while executing select statement
6. ResetFillLoadOption(): reset the FillLoadOption to its default state
7. ShouldSerializeAcceptChangesDuringFill():
8. ShouldSerializeFillLoadOption():
9. ShouldSerializeTableMappings():
10. Update(DataSet): to call the respective INSERT, UPDATE, or DELETE statements.

**DataAdapter => copied it to DataTable => DataAdapter.Fill(DataTable)**

**(open ,execute command , retrieve result, fill result in dataTable,close)**

**all are done automatically by fill method. => Datarow to take one row from dataTable => DataSet => DataAdaptor.Fill(DatasetName, Datatable(Table) ) => it is set the table into Dataset for future use**

**Once Dataset or dataTable is filled no need to open connection.**

using (SqlConnection connection = new SqlConnection(ConString))

{

SqlDataAdapter da = new SqlDataAdapter("select \* from student", connection);

//Using Data Table

DataTable dt = new DataTable();

da.Fill(dt);

//The following things are done by the Fill method

//1. Open the connection

//2. Execute Command

//3. Retrieve the Result

//4. Fill/Store the Retrieve Result in the Data table

//5. Close the connection

Console.WriteLine("Using Data Table");

//Active and Open connection is not required

//dt.Rows: Gets the collection of rows that belong to this table

//DataRow: Represents a row of data in a DataTable.

foreach (DataRow row in dt.Rows)

{

//Accessing using string Key Name

Console.WriteLine(row["Name"] + ", " + row["Email"] + ", " + row["Mobile"]);

//Accessing using integer index position

//Console.WriteLine(row[0] + ", " + row[1] + ", " + row[2]);

}

Console.WriteLine("---------------");

//Using DataSet

DataSet ds = new DataSet();

da.Fill(ds, "student"); //Here, the datatable student will be stored in Index position 0

Console.WriteLine("Using Data Set");

//Tables: Gets the collection of tables contained in the System.Data.DataSet.

//Accessing the datatable from the dataset using the datatable name

foreach (DataRow row in ds.Tables["student"].Rows)

{

//Accessing the data using string Key Name

Console.WriteLine(row["Name"] + ", " + row["Email"] + ", " + row["Mobile"]);

**ADO.NET DataTable :**

**System.Data namespace**

DataTable in C# is similar to table in Sql server

Data is going to Store in memory

central object which can be used independently (can be used by other object such as Dataset and DataView)

**Constructor:**

1. **DataTable():** initialize a new instance of the System.Data.DataTable
2. **DataTable(string tableName):**  initialize the instance with the specified table name(you want to provide while storing it in System.Data.DataTableCollection otherwise default name will be provided )
3. **DataTable(SerializationInfo info, StreamingContext context):??**
4. **DataTable(string tableName, string tableNamespace):??**

**Proporties:**

1. Columns: collection of columns of the table (DataTable.Columns.Add)
2. Constraints: collections of the constrained of the table
3. DataSet: get the dataset to which this table belongs
4. DefaultView: used to get a customized view of the table ??
5. HasErrors: get value indicating whether error in any row of the table
6. MinimumCapacity: to get or set the initial starting size for table
7. PrimaryKey: use to get or set an array of columns that function as primary keys
8. Rows: get the collections of the rows(DataTable.Rows.Add)
9. TableName: used to set the name of the datatable

**Methods :**

1. AcceptChanges(): to commit changes
2. RejectChanges() : to Rollback the Changes(copy the original datarow version to the current Data row version )
3. Remove() : Remove the row from DataRowCollections originalDataTable.Rows.Remove(row);
4. Delete(): Remove the row from DataRowCollections But actual removal occure after AcceptChanges() called. => row.Delete();
5. Clear(): to clear the all the data from dataTable
6. Clone(): to clone the structure of the dataTable (Just Schema)  
    DataTable cloneDataTable = originalDataTable.Clone();
7. Copy(): copy both structure and data of the dataTable (Schema + data) DataTable copyDataTable = originalDataTable.Copy();
8. CreateDataReader():
9. CreateInstance(): create new instance of the datatable
10. GetRowType(): . to get the row type
11. GetSchema(): . to get the schema of the table
12. ImportRow(DataRow): to copy the dataRow into a DataTable
13. Load(IDataReader): . used to fill dataTable with values from data source
14. Merge(DataTable, Boolean): used to merge Datatable with current DataTable
15. NewRow(): used to create new DataRow
16. Select(): used to get an arrray of all DataRow objects.

(Data Columns = columns of the table)

**DataColumn Properties (column ni properties : not null,data type,unique etc..)**

**does schema means template or base of the table ???**

1. AllowDBNull: used to get or set value that indicate whether column will accept null values or not
2. Autoincrement: like identity. to turn on auto-increment in column
3. AutoincrementSeed : starting value of the auto-increment
4. AutoincrementStep: This property is used to get or set the increment used by a column with its Autoincrement property set to true ???
5. Caption: used to get or set the column caption
6. ColumnName: to get or set the name of the column
7. Expression: used to filter rows, an aggregate column (min, max, average)
8. MaxLength: used to set or get maximum length of the text column
9. Unique: to get or set the column property unique

**ADO.NET DataSet :**

**System.Data namespace.**

Core component for accessing data from distributed and disconnected enviornment

represent the subset of the database in memory

collection relational data (table) in memory in tabular form

doesn't require contineuous open or active connection

based on disconnected Architecture

can have multiple tables ( **dataset.Tables.Add(table name)**)

**fetching the table from dataSet**

**// by indexing and also we can use by table name (dataSet.Tables[Orders].Rows)**

foreach (DataRow row in dataSet.Tables[0].Rows)

{

//Accessing the data using string column name

Console.WriteLine(row["ID"] + ", " + row["Name"] + ", " + row["Mobile"]);

//Accessing the data using integer index position

//Console.WriteLine(row[0] + ", " + row[1] + ", " + row[2]);

}

**Constructors:**

1. **DataSet():**  instance of the System.Data.DataSet class
2. **DataSet(string dataSetName):**  it specify the name of the instance of the dataset
3. **DataSet(SerializationInfo info, StreamingContext context): ??**
4. **DataSet(SerializationInfo info, StreamingContext context, bool ConstructSchema): ??**

**Properties (Varibles or attributes)**

1. **CaseSensitive:** specify the comparision of the string within System.Data.Datatable Object is Case Sensitive or not. if case-sensitive then it return the true.
2. **DefaultViewManager :**
3. **DataSetName :** to get or set the name of the current dataset
4. **EnforceConstraints:** to get or set a value indicating constraint should be followed or not during any update operation
5. **HasErrors:** to get a value indicating the errors in any of the datatable object
6. **IsInitialized :**  true => indicate complete initialization of object
7. **Prefix:** get or set an XML Prefix that aliases the namespace of the dataset
8. **Locate:**  get or set the locale info. used to compare string in the table
9. Namespace: to get or set the namespaces
10. Site: to get or set System.ComponentModel.ISite for Dataset
11. Relations: to get Collection of relations that link table and allow the navigation from parent table to child
12. Tables: to get the collection of tables

**Methods:**

1. **BeginInit() :** begins the initialization of dataset that can be used on a form or by another component
2. Clear(): clear the Dataset (all rows of all tables)
3. Clone(): copies the structure + Schemas + Relations + Constrains(But doesn't copy the data)
4. Copy(): both data + structure
5. CreateDataReader(): create the data reader for each table in dataset (we can do it by reader.NextResult())
6. CreateDataReader(params DataTable[] dataTables) : return the reader for the alll the dataset but return the reader by order in which parameter passed.
7. EndInit() : end the initialization of the Dataset that used on a form or used by another component. occurs at runtime
8. GetXml(): return XML representation of the dataset
9. GetXmlSchema() : return the XML Schema for Xml Representaion of the data stored in the dataset.

Which one to use DataReader or DataSet?

DataSet to use:

When you want to cache the data locally in your application so that you can manipulate the data.

When you want to work with disconnected architecture.

DataReader to use:

If you do not want to cache the data locally, then you need to use DataReader which will improve the performance of your application.

DataReader works on connected-oriented architecture i.e. it requires an open connection to the database.