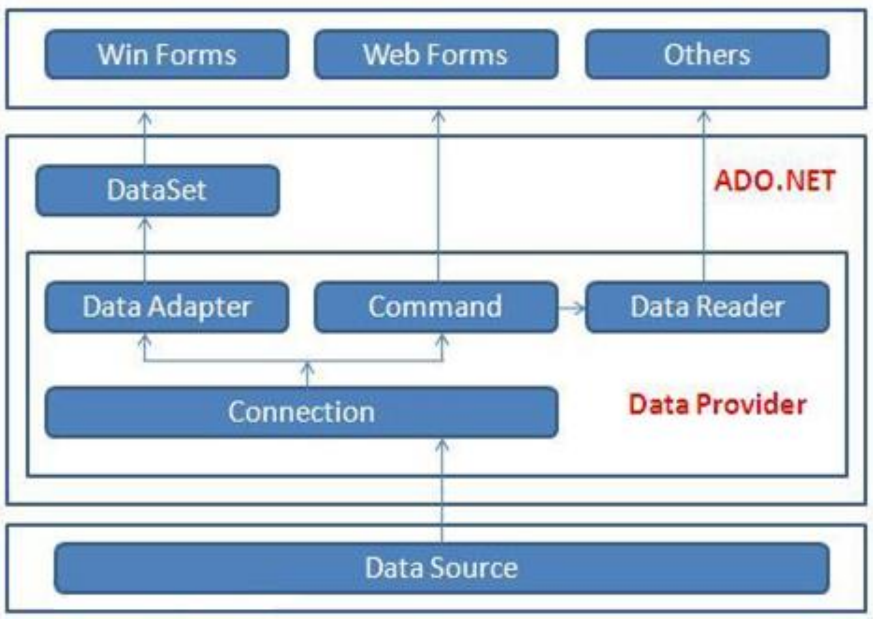
Data Access using ADO.NET

- ☞ Evolution of ADO.NET
- ☞ ADO .NET Architecture
- ☞ ADO.NET Connected and Disconnected Models
- ☞ Data Providers in ADO.NET
- ☞ Connection Object
- ☞ Building the Connection String
- ☞ Understanding:
    - o DataReader
    - o DataSet
    - o DataAdapter
    - o DataTable
    - o DataColumn
    - o DataRow
- ☞ Data Binding
- ☞ GridView Programming

R K University

- The first data access model, DAO (Data Access Object) was created for local databases with the built-in Jet engine which had performance and functionality issues.
- Next came **RDO (Remote Data Object)** and **ADO (Active Data Object)** which were designed for Client Server architectures but, soon ADO took over RDO.
- ADO was a good architecture but as the language changes so is the technology.
- With ADO, all the data is contained in a recordset object which had problems when implemented on the network and penetrating firewalls.
- ADO was a connected data access, which means that when a connection to the database is established the connection remains open until the application is closed.
- Leaving the connection open for the lifetime of the application raises concerns about database security and network traffic.
- Also, as databases are becoming increasingly important and as they are serving more people, a connected data access model makes us think about its productivity.
- For example, an application with connected data access may do well when connected to two clients, the same may do poorly when connected to 10 and might be unusable when connected to 100 or more.
- Also, open database connections use system resources to a maximum extent making the system performance less effective.
- Due to the above drawback, Microsoft introduced a separate data access object, which is also called as **ADO.NET**, which will be supporting both the architectures, which are Connected architecture and Disconnected architecture.
- ADO.NET provides connected access to a database connection using the .NET-managed providers and disconnected access using datasets, which are applications using the database connection only during retrieval of data or for data update.
- Dataset is the component helping to store the persistent data in memory to provide disconnected access for using the database resource efficiently and with better scalability.

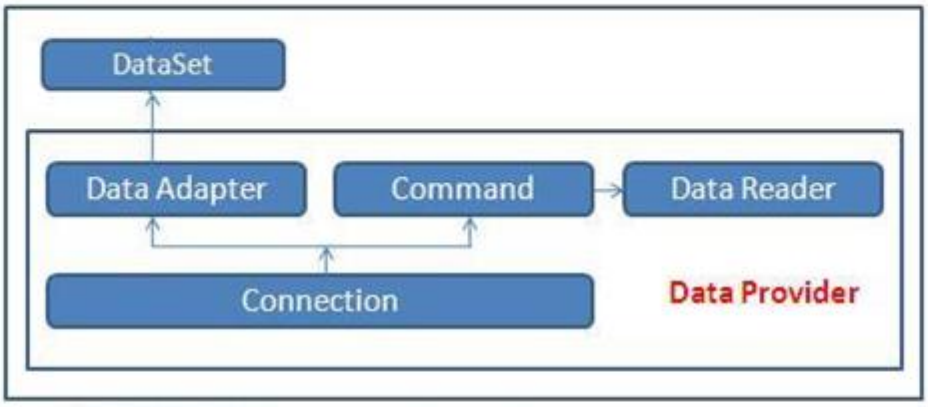
ADO.NET Architecture



- ADO.NET consists of a set of Objects that expose data access services to the .NET environment.

- It is a data access technology from Microsoft .Net Framework, which provides communication between relational and nonrelational systems through a common set of components.
- **System.Data** namespace is the core of ADO.NET and it contains classes used by all data providers.
- ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate ADO.NET data access code.

Data Providers and DataSet



- The two key components of ADO.NET are **Data Providers** and **DataSet**.
- The Data Provider classes are meant to work with different kinds of data sources.
- They are used to perform all data-management operations on specific databases.
- DataSet class provides mechanisms for managing data when it is disconnected from the data source.

Data Providers:

- The Data Provider is responsible for providing and maintaining the connection to the database.
- A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner.
- The .Net Framework includes mainly three Data Providers for ADO.NET.
- They are the **Microsoft SQL Server Data Provider**, **OLEDB Data Provider** and **ODBC Data Provider**.
- SQL Server uses the SqlConnection object, OLEDB uses the OleDbConnection Object and ODBC uses OdbcConnection Object respectively.
- A data provider contains Connection, Command, DataAdapter, and DataReader objects.
- These four objects provide the functionality of Data Providers in the ADO.NET.

Connection

- The Connection Object provides physical connection to the Data Source.
- Connection object needs the necessary information to recognize the data source and to log on to it properly, this information is provided through a connection string.

Command

- The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source.
- The command object provides a number of Execute methods that can be used to perform the SQL queries in a variety of fashions.

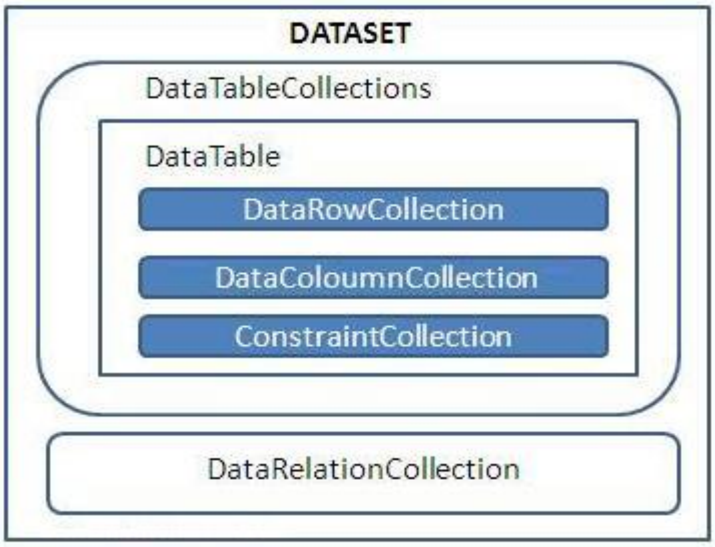
DataReader

- The DataReader Object is a stream-based, forward-only, read-only retrieval of query results from the Data Source, which do not update the data.
- DataReader requires a live connection with the database and provides a very intelligent way of consuming all or part of the result set.

DataAdapter

- DataAdapter Object populates a Dataset Object with results from a Data Source.
- It is a special class whose purpose is to bridge the gap between the disconnected Dataset objects and the physical data source.

DataSet:



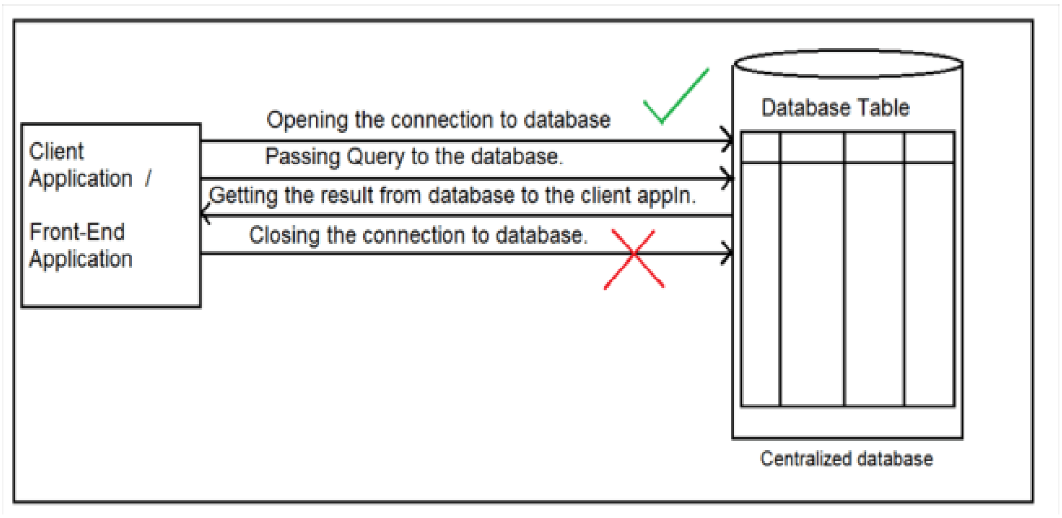
- The dataset is a disconnected, in-memory representation of data.
- It can be considered as a local copy of the relevant portions of the database.
- The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database.
- When the use of this DataSet is finished, changes can be made back to the central database for updating.
- The data in DataSet can be loaded from any valid data source like Microsoft SQL server database, an Oracle database or from a Microsoft Access database.
- DataSet provides a disconnected representation of result sets from the Data Source, and it is completely independent from the Data Source.
- DataSet provides much greater flexibility when dealing with related Result Sets.
- DataSet contains rows, columns, primary keys, constraints, and relations with other DataTable objects.
- It consists of a collection of DataTable objects that you can relate to each other with DataRelation objects.
- The DataAdapter Object provides a bridge between the DataSet and the Data Source.

Data Access Object Architecture

- In order to connect or communicate front end of the Application to the database, we have two different types of data access architecture, which acts as a mediator or an interface between the two components:
    - Connected oriented architecture.
    - Disconnected oriented architecture.

Connected Oriented Architecture

- In Connected oriented architecture, for each and every operation to the database like insert/update/delete, the client Application or front end will be performing an operation on the central database directly.
- In this architecture, the programmer or a developer needs to open SQL connection before performing the operation like inserting the record, updating the record or deleting the record from the database table, once the operations to a database are done then the programmer needs to close SQL connection explicitly.
- Diagram to understand connected oriented architecture


Client Application / Front-End Application
Opening the connection to database
Passing Query to the database.
Getting the result from database to the client appln.
Closing the connection to database.
Database Table
Centralized database

Advantages of Connected Oriented Architecture

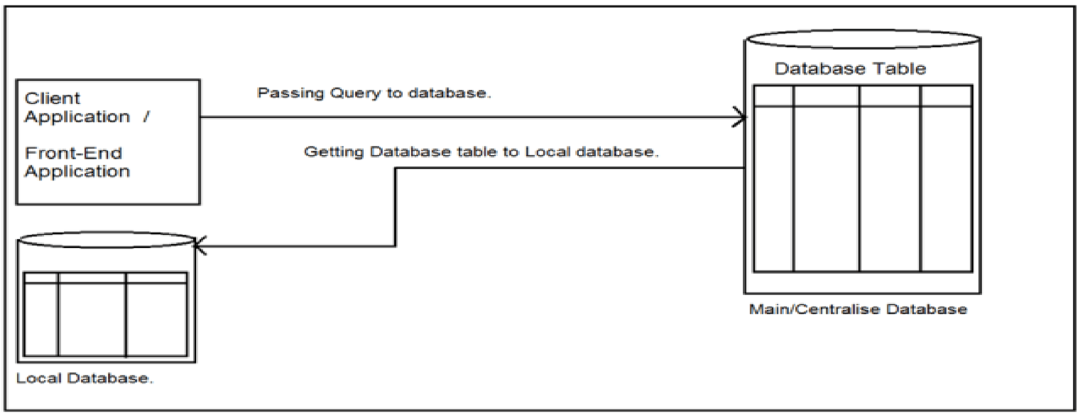
- In this architecture, as we perform database operations on the centralized database, accessing the data will be faster.
- The data will be more secure, as it is communicating with the database directly.

Disadvantages of Connected Oriented Architecture

- In this architecture, the programmer or developer needs to open the connection to the database before sending the queries and after the results are fetched, the programmer needs to close the connection explicitly.
- Every time connecting the centralized database will be more burden, as it degrades the database performance.
- It increases the network traffic.
- For every operation on the database, it will hit the database every time.

Disconnected Oriented Architecture

- In Disconnected Oriented Architecture, the client Application or frontend will not interact or communicate with the centralized database but it will interact with the local database, which will be a part of the client machine.
- Whatever operations are done to the local database, it will be reflected back to the centralized database automatically.
- In this architecture, to open the connection to database or to close the connection to database, programmer/developer doesn't require anything to write, which would be done internally.


Client Application / Front-End Application
Passing Query to database.
Getting Database table to Local database.
Database Table
Main/Centralise Database
Local Database.

Advantages of using Disconnected Oriented Architecture

- The programmer/developer does not need any code to write such as to open the connection or to close the connection to a database.
- It reduces the network traffic.
- It reduces the burden on the database (sql server).
- As database operations are done to a local database (client machine) there will be no database hitting for every request.

Disadvantages of using Disconnected Oriented Architecture

- As the data will be a part of local database or client machine, there will be less security, which can be hacked easily.
- For every request or operation to the database, local database will act as an interface between the user interface and the centralized database, the result of local database will be reflected to the centralized database, so in this case accessing/performance will be slow compared to the connected oriented architecture.
- In this architecture, a local database will be created within the client machine.
- Due to this reason, it will consume the client machine's memory, which effects the performance of the client machine.

ADO.NET Data Providers

- The classes responsible for the movement of data between the disconnected data classes in the client application and the data store are referred to as connected classes or provider classes.
- An ADO.NET data provider connects to a data source and provides a way to execute commands against that data source in a consistent manner that is independent of the data source and data source-specific functionality.
- However, aside from a core set of similar capabilities, there is no guarantee that identical functionality will be available in each data provider.
- This is due to differences between data sources and provider implementations
- The ADO.NET Framework comes with the following providers:
- **OLEDB:** The OLEDB provider, expressed through the System.Data.OleDb namespace.
- You can use this provider to access SQL Server 6.5 and earlier, SyBase, DB2/400, and Microsoft Access.
- **ODBC:** The ODBC provider, expressed through the System.Data.Odbc namespace.
- This provider is typically used when no newer provider is available.
- **SQL Server:** The Microsoft SQL Server provider, expressed through the System.Data.SqlClient namespace.

- It Contains classes that provide functionality similar to the generic OleDb provider.
- The difference is that these classes are tuned for SQL Server 7 and later data access.
- Prior to .NET Framework 4, Microsoft also included a functional **Oracle provider** with the .NET Framework.
- However, its classes have been marked as deprecated and obsolete in .NET Framework 4.
- Core objects that make up a .NET Framework data provider:
- **Connection:** Establishes a connection to a specific data source.
- The base class for all Connection objects is the DbConnection class.
- **Command:** Executes a command against a data source.
- Exposes Parameters and can execute in the scope of a Transaction from a Connection.
- The base class for all Command objects is the DbCommand class.
- **DataReader:** Reads a forward-only, read-only stream of data from a data source.
- The base class for all DataReader objects is the DbDataReader class.
- **DataAdapter:** Populates a DataSet and resolves updates with the data source.
- The base class for all DataAdapter objects is the DbDataAdapter class.
- In addition to the core classes .NET Framework data provider also contains the classes **Transaction**, **CommandBuilder**, **ConnectionStringBuilder**, **Parameter**, **Exception**, **Error** and **ClientPermission**
- Depending on the design and data source for your application, your choice of .NET Framework data provider can improve the performance, capability, and integrity of your application.
- The following table discusses the advantages and limitations of each .NET Framework data provider.

Provider	Notes
SQL Server	Recommended for middle-tier applications that use Microsoft SQL Server. Recommended for single-tier applications that use Microsoft Database Engine (MSDE) or SQL Server.
OLEDB	Recommended for single-tier applications that use Microsoft Access databases. Use of an Access database for a middle-tier application is not recommended.
ODBC	Recommended for middle and single-tier applications that use ODBC data sources.
Oracle	Recommended for middle and single-tier applications that use Oracle data sources.

- To communicate with SQL Server database, we need to use **system.data.sqlclient** as our namespace to access its classes, properties or the methods.
- We have various classes in the above namespace like:
    - SqlConnection
    - SqlCommand
    - SqlDataReader
    - SqlDataAdapter
- Data provider will have the responsibilities given below.
    - Maintaining the connection to the centralized database.
    - Executing the query into the centralized database.
    - Fetching the query result and forwarding to the front end Application.
    - Fetching the query result and filling into the local database.

- To use the above four classes i.e. **SqlConnection** class, **SqlCommand** class, **SqlDataReader** class and **SqlDataAdapter** class and their members; we need to create the object for each classes.

Connection Object

- **Syntax**
  - SqlConnection con=new SqlConnection();
- This object is used to maintain the connection to centralized database, which is opening the connection, maintaining the connection and closing the connection to the database, which has to be done by the developer/ programmer explicitly.
- To open the connection, we need to use open method.
  - example : con.Open();
- To close the connection, we need to use close method.
  - example : con.Close();
- **Open()** : This method is used to open the connection to the centralized database in order to perform certain operations or queries.
- **Close()** : Once the result is fetched, we need to close the connection to the database.

Command Object

- **Syntax**
  - SqlCommand cmd=new SqlCommand();
- This object is used for executing the given sql commands / queries within the sql server database.

SqlDataReader Object

- **Syntax**
  - SqlDataReader dr=cmd.ExecuteReader();
- This object will fetch the results of SQL command/ queries and will send to the client Application / front end Application.

SqlDataAdapter Object

- **Syntax**
  - SqlDataAdapter da=new SqlDataAdapter();
- This SqlDataAdapter is a part of System.data.SqlClient namespace and it is used for disconnected oriented architecture.
- This object is used to fetch the results of the executed SQL commands/ queries and will be filling into the local database.
- Data Adapter object acts as an interface/ mediator between the centralized database and the client Application.

DataSet

- Data Set is used as part of disconnected oriented architecture.
- In disconnected oriented architecture, we have a local database, where we perform the given SQL queries or SQL commands, which is a part of client machine.
- Data Set is a local database, which is a predefined class defined by Microsoft with the System.Data namespace.
- Dataset contains the fetched records from the centralized database and will be stored into local database in XML table format.
- **Syntax:**
  - Dataset ds=new Dataset();

- Objects to be used in Connected Oriented Architecture
    - SqlConnection object
    - SqlCommand object
    - SqlDataReader object.
- Objects to be used in Disconnected Oriented Architecture
    - SqlConnection object
    - SqlCommand object
    - SqlDataAdapter object
    - DataSet

DataTable

- A DataTable object resembles a database table and has a collection of DataColumns (i.e. fields) and DataRows (i.e. records).
- It can also have a primary key based on one or more columns and a collection of Constraint objects which are useful for enforcing the uniqueness of the values in a column.
- DataTable objects in a DataSet class are often tied to each other through relationships.
- The majority of applications will populate a DataTable directly from a database however it is possible to fill a datatable using code.

Creating a DataTable Object

- DataTable dt = new DataTable();
- DataSet ds = new DataSet();

Retrieving a DataTable from a DataSet

da.Fill(ds);

- dt = ds.Table[0];
- dataGridView1.DataSource = ds.Table[0];

Filtering, Searching and Sorting

- This performs a simple filter and returns an array of matching DataRow objects.
    - DataRows dr = new DataRows();
    - dr = objDataTable.Select("FirstName = 'Rohan'")

DataColumn

- The DataColumn is the fundamental building block for creating the schema of a DataTable.
- Each DataColumn has a DataType property that determines the kind of data the DataColumn contains.
- For example, you can restrict the data type to integers, or strings, or decimals.
- Because data that is contained by the DataTable is typically merged back into its original data source, you must match the data types to those in the data source.
- Properties such as **AllowDBNull**, **Unique**, and **ReadOnly** put restrictions on the entry and updating of data, thereby helping to guarantee data integrity.
- You can also use the **AutoIncrement**, **AutoIncrementSeed**, and **AutoIncrementStep** properties to control automatic data generation.
    - DataColumn col = new DataColumn();
    - col.ColumnName = "id";
    - col.DataType = System.Type.GetType("System.Int32");
    - col.AutoIncrement = true;
    - col.AutoIncrementSeed = 1;
    - col.AutoIncrementStep = 1;

- col.ReadOnly = true;
- table.Columns.Add(col);

DataRow

- The DataRow and DataColumn objects are primary components of a DataTable.
- Use the DataRow object and its properties and methods to retrieve and evaluate; and insert, delete, and update the values in the DataTable.
- The DataRowCollection represents the actual DataRow objects in the DataTable, and the DataColumnCollection contains the DataColumn objects that describe the schema of the DataTable.
- Use the overloaded **Item()** property to return or set the value of a DataColumn.
- To create a new DataRow, use the **NewRow** method of the DataTable object.
- After creating a new DataRow, use the Add method to add the new DataRow to the DataRowCollection.
- Finally, call the **AcceptChanges** method of the DataTable object to confirm the addition.
    - DataTable custTable = new DataTable("Customers")
    - DataRow row1 = custTable.NewRow()
    - custTable.Rows.Add(row1);

Steps to Communicate with SQL Server Database

Disconnected Oriented Architecture:

- **Step 1:** Including/Importing data provider namespace
    - **using System.Data.SqlClient;**
- **Step 2:** Declaring the connection string to connect to the database
    - string ConneString= "Server=Servername; database=databasename; userid=sa; password=abc";
- **Step 3:** In this step, we have to use SqlConnection object by using the connection string parameter and open the connection:
    - SqlConnection connection=new SqlConnection(conneString);
    - connetion.Open( );
- **Step 4:** In this step, we need to use the command object by initializing sql query/sql command with connection object reference:
    - SqlCommand cmd=new SqlCommand("Query to database like CRUD", connection);
- **Step 5:** In this step we need to create a local database by creating its object. Before creating any object, we need to import the namespace of **DataSet**, so that we can use all its classes, methods, properties etc.
    - using System.data;
    - DataSet dds=new DataSet();
- **Step 6:** In this step, we will be creating SqlDataAdapter object by initializing the SqlCommand object reference:
    - SqlDataAdapter daa=new SqlDataAdapter(cmd);
- **Step 7:** After getting the results of the executed SQLcommand/SQL queries from SqlDataAdapter, we will be filling into our local database:
    - daa.Fill(dds,"NameoftheTable");
- Here fill method is part of SqlDataAdapter which is used to fetch the records from centralized database and will fill into local database.

Connected Oriented Architecture:

- **Step 1:** Including data provider into namespace, as shown below.
    - using System.Data.SqlClient;
- **Step 2:** Declaring connection string for connecting to the database:
    - string ConneString= "Server=Servername;database=database-name;userid=sa;password=abc;";
- **Step 3:** In this step, we have to use SqlConnection object by using the connection string:
    - SqlConnection connection=new SqlConnection(ConneString);
    - connection.Open( );
- **Step 4:** In this step, we need to use command object by initializing SQL query/SQL command
    - SqlCommand cmd=new SqlCommand("Query to database like CRUD", connection);
- **Step 5:** In this step, we need to create datareader object:
    - SqlDataReader dr = cmd.ExecuteReader();
- **Step 6:** Once the data reader fetches the record from the centralized database, we will bind this data from DataReader object to the client Application.
    - GridView1.DataSource=dr;
- **Step 7:** After the execution is completed, the programmer needs to close the connection explicitly.
    - connection.Close();

GridView Programming

- Displaying data in a tabular format is a task you are likely to perform frequently.
- The **DataGridView** control is designed to be a complete solution for displaying tabular data with Windows Forms.
- The DataGridView control is highly configurable and extensible, and it provides many properties, methods, and events to customize its appearance and behavior.
- The DataGridView control makes it easy to define the basic appearance of cells and the display formatting of cell values.
- The cell is the fundamental unit of interaction for the DataGridView.
- All cells derive from the DataGridViewCell base class.
- Each cell within the DataGridView control can have its own style, such as text format, background color, foreground color, and font.
- Typically, however, multiple cells will share particular style characteristics.
- The data type for the cell's Value property by default is of type Object.
- Example:

```
private void Form1_Load(object sender, EventArgs e)
{
        string cs = "Connection String";
        SqlConnection cn = new SqlConnection(cs);
        cn.Open();

        SqlCommand cmd = new SqlCommand("select * from Student", cn);
        SqlDataAdapter da = new SqlDataAdapter(cmd);
        DataSet ds = new DataSet();
        da.Fill(ds);
```

```
            dataGridView1.DataSource = ds.Tables[0];
        }
```