ADO.NET is an object-oriented set of libraries that allows you to interact with data sources.

The SqlDataAdapter holds the SQL commands and connection object for reading and writing data.

DataSet

DataSet is a disconnected orient architecture that means there is no need of active connections during work with datasets and it is a collection of DataTables and relations between tables. It is used to hold multiple tables with data. You can select data form tables, create views based on table and ask child rows over relations.

protected void BindGridview()

{

SqlConnection conn = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");

conn.Open();

SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);

SqlDataAdapter sda = new SqlDataAdapter(cmd);

DataSet ds = new DataSet();

sda.Fill(ds);

gvUserInfo.DataSource = ds;

gvUserInfo.DataBind();

}

DataTable

DataTable represents a single table in the database. It has rows and columns. There is no much difference between dataset and datatable, dataset is simply the collection of datatables.

protected void BindGridview()

{

SqlConnection con = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");

conn.Open();

SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);

SqlDataAdapter sda = new SqlDataAdapter(cmd);

DataTable dt = new DataTable();

da.Fill(dt);

gridview1.DataSource = dt;

gvidview1.DataBind();

}

DataAdapter

DataAdapter will acts as a Bridge between DataSet and database. This dataadapter object is used to read the data from database and bind that data to dataset. Dataadapter is a disconnected oriented architecture. Check below sample code to see how to use DataAdapter in code:

protected void BindGridview()

{

SqlConnection con = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");

conn.Open();

SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);

SqlDataAdapter sda = new SqlDataAdapter(cmd);

DataSet ds = new DataSet();

da.Fill(ds);

gvUserInfo.DataSource = ds;

gvUserInfo.DataBind();

}

Lets you close the connection as soon it's done loading data, and may even close it for you automatically

All of the results are available in memory

You can iterate over it as many times as you need, or even look up a specific record by index

Has some built-in faculties for updating back to the database.

DataReader

DataReader is used to read the data from database and it is a read and forward only connection oriented architecture during fetch the data from database. DataReader will fetch the data very fast when compared with dataset. Generally we will use ExecuteReader object to bind data to datareader.

Protected void BindGridview()

{

using (SqlConnection conn = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test"))

{

con.Open();

SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);

SqlDataReader sdr = cmd.ExecuteReader();

gvUserInfo.DataSource = sdr;

gvUserInfo.DataBind();

conn.Close();

}

}

Holds the connection open until you are finished (don't forget to close it!).

Can typically only be iterated over once

Is not as useful for updating back to the database

DataView

A dataview is a view on a datatable, a bit like a sql view. It allows you to filter and sort the rows - often for binding to a windows form control.

Additionally, a DataView can be customized to present a subset of data from the DataTable. This capability allows

you to have two controls bound to the same DataTable, but showing different versions of the data. For example, one

control may be bound to a DataView showing all of the rows in the table, while a second may be configured to display

only the rows that have been deleted from the DataTable. The DataTable also has a DefaultView property which returns

the default DataView for the table.

DataView custDV = new DataView(custDS.Tables["Customers"],

"Country = 'USA'",

"ContactName",

DataViewRowState.CurrentRows);

Transactions in ADO.Net

private static void Demo1()

{

SqlConnection db = new SqlConnection("connstringhere");

SqlTransaction transaction;

db.Open();

transaction = db.BeginTransaction();

try

{

new SqlCommand("INSERT INTO TransactionDemo " +

"(Text) VALUES ('Row1');", db, transaction)

.ExecuteNonQuery();

new SqlCommand("INSERT INTO TransactionDemo " +

"(Text) VALUES ('Row2');", db, transaction)

.ExecuteNonQuery();

new SqlCommand("INSERT INTO CrashMeNow VALUES " +

"('Die', 'Die', 'Die');", db, transaction)

.ExecuteNonQuery();

transaction.Commit();

}

catch (SqlException sqlError)

{

transaction.Rollback();

}

db.Close();

}

Transactions Isolation Levels

1. ReadUncommitted – This is, essentially, no isolation. Anyone can read the data placed in a table or updated immediately after the SQL statement causes the change – no commit is required. This could lead to a process having out-of-date data: it may be using a version of the data that was then rolled back out of the table!

2. ReadCommitted – This is slightly more isolated. In this case, a transaction can only read data from the table that has already been committed. When a transaction wants to update data, it acquires a shared lock on that data and (if successful getting the lock) updates the data. Transactions outside of that transaction cannot update the data in that table until the locking transaction commits. This is only slightly more isolated, however: a SQL statement executed twice within a transaction could return a different result-set if a second transaction changes and commits the data the SQL statement executes on between the two statements. This is the default isolation level for SqlTransaction.

3. RepeatableRead – Slowly getting more isolated. In this case, a shared lock is applied on all data queried within a transaction. This means that no other transaction can alter the data used in your transaction. This prevents the case where data you had queried once changes on subsequent queries. It does not, though, prevent the case where rows are added to the table that may be returned in subsequent queries.

4. Serializable – Locks are placed on ranges of the tables you are using, preventing other users from changing your data or adding new rows underneath you. This is the most isolated isolation level, but it does come with the drawback of locking more data than your transaction may strictly need.