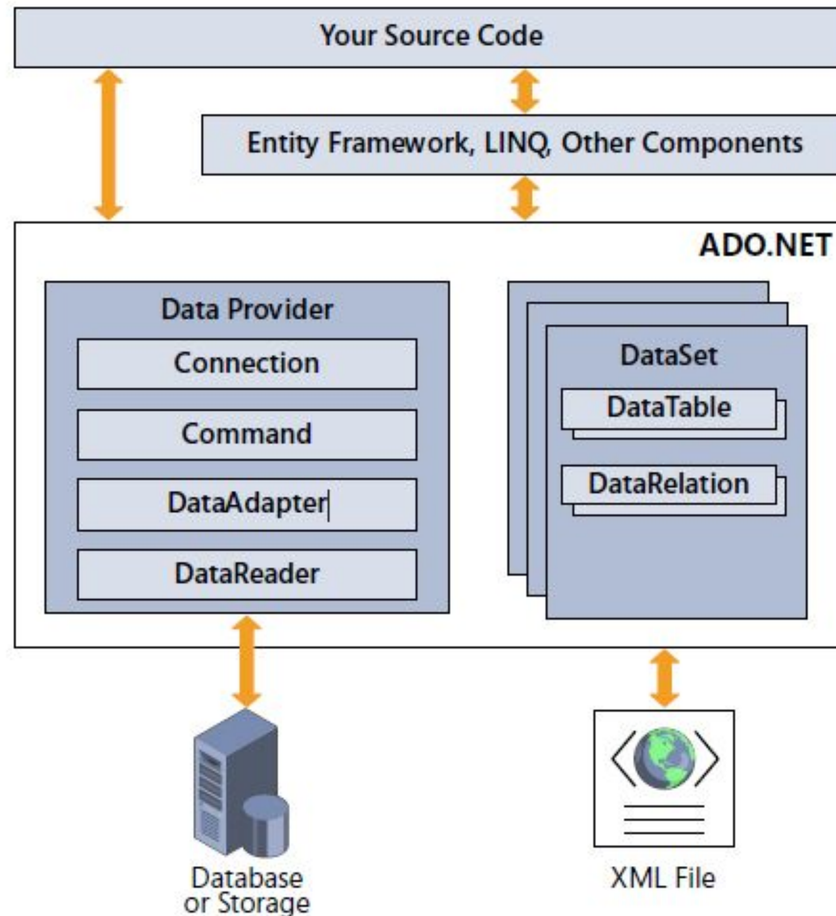
ADO.net

ADO.NET, ընդհանուր սկզբունքներն ու կառուցվածքը

- Ado.net-ը ապահովում է միևնույն ծրագրավորման մոդելը տարբեր տվյալների աղբյուրների հետ աշխատելու համար:
- Տարբեր տվյալների բազաներին միանալու հնարավորություն ապահովելու համար օգտագործվում են տարբեր *պրովայդերներ:*
- Արտաքին տվյալների աղբյուրի հետ կոմունիակցիան ապահովվում է *Connection* օբյեկտի միջոցով:
- SQL հրամանների ուղարկումն ու արդյունքների ստացումը կատարվում է *Command* օբյեկտի միջոցով: *Command* օբյեկտը նաև օժտված է *Parameter* հատկանիշով, ինչը թույլ է տալիս փոխանցել պարամետրեր SQL հարցումներին, ինչպես նաև ստեղծել և օգտագործել Stored Procedure-ներ:
- *DataAdapter* օբյեկտները թույլ են տալիս կապ հաստատել բազայի և այնպիսի ADO.net օբյեկտների միջև, ինչպիսիք են DataSet-ը և DataTable-ը:
- *DataReader*-ը ապահովում է SQL հարցման արդյունքի read-only, արագացված ընթերցում:

ADO.NET, ընդհանուր սկզբունքներն ու կառուցվածքը



Միացումը արտաքին տվյալներին:

- Ado.net-ի connection ստեղծելու համար անհրաժեշտ է Ado.net-ին փոխանցել մի շարք ինֆորմացիաներ: Դրանցից են.
  - որտեղ է գտնվում տվյալների աղբյուրը
  - ինչ անուն ունի աղբյուրը
  - ինչպիսի ֆորմատի տվյալներ են այնտեղ պարունակվում
  - security պարամետրեր, որոնք թույլ են տալիս միանալ բազային
  - տվյալների բազայի ղեկավարման համակարգին փոխանցվող պարամետրեր և այլն...
- Այս տվյալները փոխանցում են միացման տողի (connection string) միջոցով: Դրա սինտաքսիսն է. key1=value1; key2=value2;...;keyN=valueN

ConnectionString-երի օրինակներ

- Data Source=(local)\SQLEXPRESS;Initial Catalog=Sample; Integrated Security=True – միանում է լոկալ համակարգչի վրա գտնվող SQLEXPRESS սերվերի Sample անվամբ բազային օգտագործելով Windows security:
- Server=myServerName; Database=myDataBase; User Id=myUsername; Password=myPassword; - միանում է myServerName սերվերի default instance-ին:
- Server=192.168.172.80,1433; Database=myDataBase; User Id=myUsername; Password=myPassword; - միանում է նշված IP-ով սերվերի սերվերի default instance-ին:

System.Data.SqlClient.SqlConnectionStringBuilder դասը

- *SqlConnectionStringBuilder դասը թույլ է տալիս դինամիկ սահմանել connectionstring*

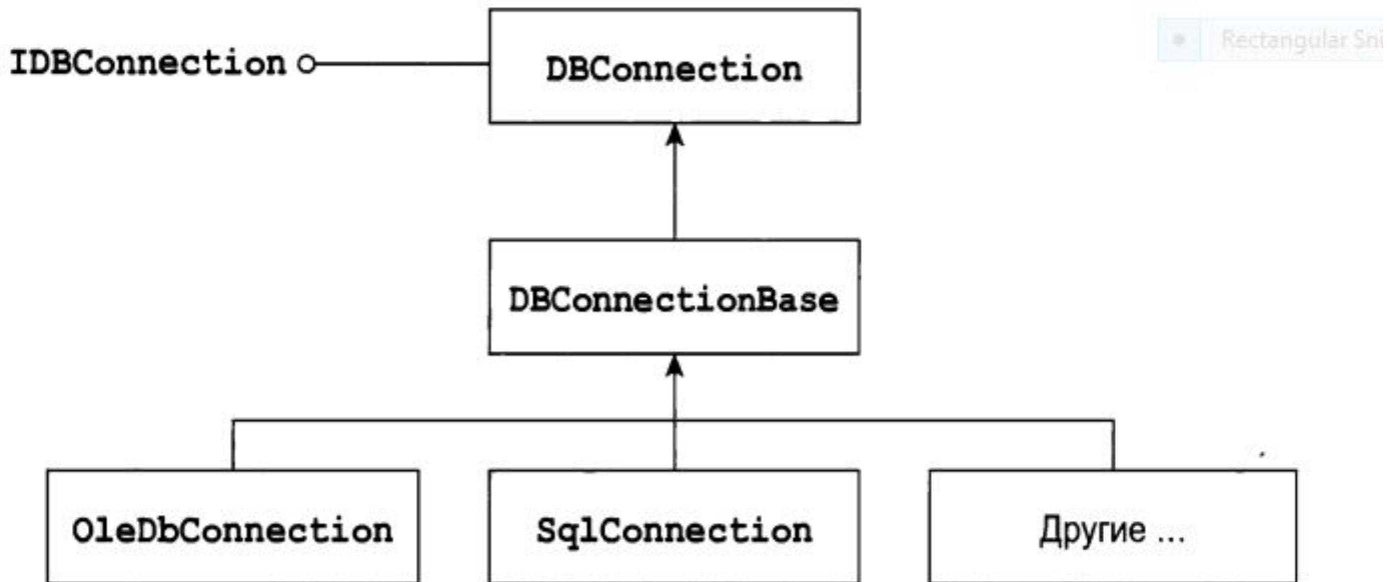
օր.

```
  SqlClient.SqlConnectionStringBuilder builder =
new SqlClient.SqlConnectionStringBuilder();
builder.DataSource = @"(local)\SQLEXPRESS";
builder.InitialCatalog = “Sample";
builder.IntegratedSecurity = true;

return builder.ConnectionString;
```

Connection

Connection դասերը



Connection օբյեկտի ստեղծումը և բացումը

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
  connection.Open();

  // ինչ որ խելոք բաներ
}
```

- Connection օբյեկտի ստեղծումը չի բացում կապը, հետևաբար անհրաժեշտ է դա կատարել բացահայտ՝ Open() մեթոդի միջոցով:

ADO.NET Connected Model

Command օբյեկտը

- Դեպի տվյալների բազա SQL հրամանի ուղարկման և արդյունքի ստացման համար օգտագործվում են Command դասերը:
- Command դասի կոնստրուկտորը որպես արգումենտ ընդունում է Connection օբյեկտ և հրամանի տողը: Որպես հրաման կարող է հանդես գալ նաև Stored Procedure անվանումը:

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();

    SqlCommand command =
        new SqlCommand("Select * from product",  connection);
}
```

- կորեկտ աշխատանքի համար command օբյեկտին անհրաժեշտ է փոխանցել բացված connection

Command-ի կատարումը

- Կախված սպասվող տվյալների տիպից, Command դասը առաջարկում է հետևյալ մեթոդները.
- ExecuteNonQuery() – կատարում է նշված հրամանը և վերադարձնում այն տողերի քանակը, որոնք ենթարկվել են փոփոխության: Որպես կանոն օգտագործվում է Update, Delete և Insert SQL հրամանների հետ:
- ExecuteScalar() – կատարում է հրամանը և վերադարձնում է արդյունքի առաջին տողի առաջին սյան արժեքը:
- ExecuteReader() – որպես արդյունք վերադարձնում է IDataReader ինտերֆեյսի տիպավորված իրականացում:
- ExecuteXmlReader() – վերադարձնում է XmlReader տիպի օբյեկտ, որը թույլ է տալիս կարդալ տվյալների բազայից ստացված Xml արդյունքը:

SqlDataReader դասը

- SqlDataReader օբյեկտը իրենից ներկայացնում է բացված connection-ով բազայից տվյալների կարդացման արագ և պարզագույն միջոց:
- SqlDataReader օբյեկտի անմիջական ստեղծում հնարավոր չէ: Օբյեկտի ստացման միակ հնարավոր միջոցը SqlCommand.ExecuteReader() մեթոդն է:
- SqlDataReader դասը իրականացնում է IDisposable ինտերֆեյսը:
- *Սահմանափակումները*
  - DataReader-ը թույլ է տալիս կարդալ տվյալներ միայն մեկ ուղղությամբ
  - Կատարել փոփոխություններ տվյալների բազայում DataReader օբյեկտի միջոցով հնարավոր չէ

Բազայի հետ կապի ավարտը

- DataReader օբյեկտի հետ աշխատանքը ավարտելուց անմիջապես հետո անհրաժեշտ է այն բացահայտ փակել Close() մեթոդի միջոցով:
- DataReader-ի փակումը չի ապահովում Connection-ի փակում, հետևավար անհրաժեշտ է բացահայտ կերպով փակել Connection-ը DataReader-ի հետ աշխատանքի ավարտից անմիջապես հետո:
- SqlCommand.ExecuteReader մեթոդի գերբեռնումներից մեկը ընդունում է որպես արգումենտ CommandBehavior տիպի օբյեկտ, որի CommandBehavior.CloseConnection արժեքը ավտոմատ կերպով փակում է connection-ը՝ reader-ի փակումից անմիջապես հետո:

```
string connectionString = @"Data Source=NOTEBOOK\SQLSERVER;Initial Catalog=northwind;Integrated
   Security=True";
using(SqlConnection connection = new SqlConnection(connectionString))
{
   connection.Open();
   string sqlString = "Select * From Products";
   SqlCommand command = new SqlCommand(sqlString, connection);
   using(SqlDataReader reader = command.ExecuteReader())
   {
     // data reading code
   }
}
```

Տվյալների unsafe ընթերցումը Reader-ի միջոցով

- SqlDataReader.Read() մեթոդը բեռնում է Reader օբյեկտի մեջ բազայի հերթական տողը: Վերադարձնում է true, եթե տողը հաջողվել է ընթերցել, հակառակ դեպքում (հիմնականում՝ եթե այլևս նոր տողեր չկան)՝ false:
- SqlDataReader դասում սահմանված է ինդեքսատոր, որի գերբեռնումները թույլ են տալիս ստանալ տիպերի տեսանկյունից անապահով հասանելիություն տվյալներին.
  օր. object obj = reader[“ProductName”];
  object obj = reader[0];
- Առաջին դեպքում կատարվում է փնտրում reader-ի բոլոր սյունակներով՝ մինչև համապատասխան անվամբ սյունակի հայտնաբերումը: Արդյունքում առաջին տարբերակը (this[string] գերբեռնումը) զգալիորեն դանդաղ է this[Int32] գերբեռնումից:

Տվյալների տիպավորված ընթերցումը Reader-ի միջոցով:

- DataReader-ից տվյալների տիպավորված ընթերցումը կարելի է կազմակերպել SqlDataReader.GetXXX(int index) մեթոդների միջոցով, որտեղ xxx-ը այն տիպն է, որի արժեքը պատրաստվում ենք ստանալ: օր. reader.GetString(1), reader.GetInt32(4):

- reader.GetOrdinal(string colName) մեթոդի միջոցով կարելի է ստանալ համապատասխան անվամբ սյունակի հերթական համարը, որը կարելի է օգտագործել GetXXX մեթոդներում:

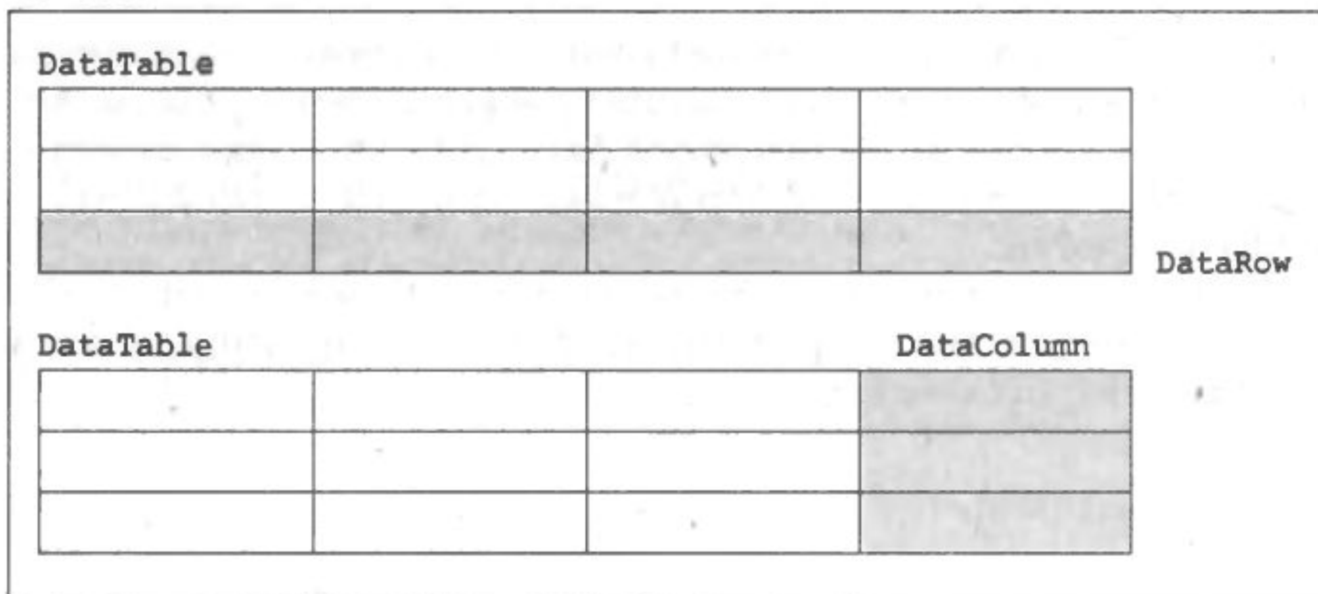
```
string connectionString = @"Data Source=NOTEBOOK\SQLSERVER;Initial Catalog=northwind;Integrated
    Security=True";
using(SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string sqlString = "Select * From Products";
    SqlCommand command = new SqlCommand(sqlString, connection);

    using(SqlDataReader reader = command.ExecuteReader())
    {
     while (reader.Read())
          {
           int productName = reader.GetInt32(reader.GetOrdinal("ProductName"));
                        // data reading code
     }
    }
}
```

ADO.NET Disconnected Model

DataSet

- DataSet-ը իրենից ներկայացնում է ավտոնոմ (անջատված) տվյալների կոնտեյներ: Դրա ստրուկտուրան գրեթե նույնությամբ կրկնում է ռելացիոն բազայի ստրուկտուրան, սակայն DataSet-ը կարող է պարունակել տվյալներ գործնականում ցանկացած աղբյուրից (XML, կոդով նշված տվյալներ և այլն):



DataTable

- DataSet-ը оժտված է DataTable օբյեկտների կոլեկցիայով: Յուրաքանչյուր DataTable ունի խիստ տիպավորված սյուներ, որոնք արտահայտվում են DataColumns կոլեկցիայով:
- DataTable-ի յուրաքանչյուր տող DataRow տիպի է:

DataColumn

- DataColumn օբյեկտ ստեղծելու համար պետք է նշել նրա անունը և այն տվյալների տիպը, որը պահվելու է նրա մեջ.

```
 DataColumn column = new DataColumn("Name",
typeof(string));
```

- DataColumn տիպի հիմնական հատկանիշներն են.
- AllowDBNull
- AutoIncrement
- AutoIncrementSeed
- AutoIncrementStep
- Caption
- ColumnName
- DataType
- DefaultValue

DataRow

- DataRow տիպը իրենից ներկայացնում է կոնկրետ սխեմայով արտահայտված տվյալների տող:
- DataRow-ի ինֆորմացիան կարդալու համար օգտագործվում է ինդեքսատոր, որի գերբեռնումը թույլ է տալիս ստանալ արժեքի կոնկրետ վերսիան.

```
row["PersonName"] = "John";
Console.WriteLine(row["PersonName", DataRowVersion.Proposed]);
```

- `DataRowVersion` կարող է ընդունել հետևյալ արժեքներից մեկը.
  - Original
  - Current
  - Proposed
  - Default

- Ամբողջ տողի ընթացիք վիճակը DataTable-ի ներսում կարելի է ստանալ DataRowState հատկանիշի միջոցով:
  - Detached
  - Unchanged
  - Added
  - Deleted
  - Modified

DataTable-ների միջև կապերի սահմանումը DataSet-ում

- Ado.net-ում աղյուսակների միջև կապերը սահմանվում են *DataRelation* դասի միջոցով: DataRelation կոնստրուկտորը արպես պարամետր ընդունում է կապի անունը (կամ null default անվան համար) և այն սյուները, որոնց միջոցով պիտի հաստատվի կապը:
- Կարելի է ստեղծել կապ նաև DataSet.Relations.Add() մեթոդի միջոցով, որը ոչ բացահայտ կերպով ստեղծում է կապ Table-ների միջև:
  օր.

```
DataSet set = new DataSet();
set.Tables.Add(Person);
set.Tables.Add(Car);

set.Relations.Add(null,
 Person.Columns["ID"], Car.Columns["PersonID"]);
```

DataSet-ի “ձեռքով” ստեղծման օրինակ

```
DataTable dataTable = new DataTable("Person");
     dataTable.Columns.Add("ID", typeof(int));
     dataTable.Columns["ID"].AutoIncrement = true;
     dataTable.Columns["ID"].AutoIncrementSeed = 1;
     dataTable.Columns["ID"].AutoIncrementStep = 1;

     dataTable.Columns.Add("PersonName", typeof(string));
     dataTable.Columns.Add("Phone", typeof(string));

     DataRow row = dataTable.NewRow();
     dataTable.Rows.Add(row);

     row["PersonName"] = "John";
     row["Phone"] = "094 83 47 26";

     DataSet set = new DataSet();
     set.Tables.Add(dataTable);
```

DataSet-ի արժեքավորումը տվյալների բազայից: DataAdapter

- Տվյալների բազայից դեպի DataSet տվյալների ավելացումը կատարվում է XXXDataAdapter դասի միջոցով, որի կոնստրուկտորը որպես արգումենտ ընդունում է բազայի Connection-ը և SQL հրաման՝ որի հիման վրա պետք է կատարել տվյալների ստացումը:
- Տվյալների լցոնումը կատարվում է Fill մեթոդի միջոցով, որը որպես արգումենտ ընդունում է DataSet կամ DataTable օբյեկտ:

DataAdapter-ի կիրառման օրինակ

```
string connectionString = @"Data Source=NOTEBOOK\SQLSERVER;Initial
  Catalog=northwind;Integrated Security=True";

using(SqlConnection connection = new SqlConnection(connectionString))
{
   connection.Open();
   string sqlString = "Select * from Products INNER JOIN Categories ON
  Products.CategoryID = Categories.CategoryID";

   SqlDataAdapter adapter = new SqlDataAdapter(sqlString, connection);
       DataSet dataSet = new DataSet();
   adapter.Fill(dataSet);

   this.dataGridView1.DataSource = dataSet.Tables[0];
}
```

DataSet-ում փոփոխված տվյալների պահպանումը բազայում

- SqlDataAdapter դասը օժտված է SelectCommand, UpdateCommand, InsertCommand, DeleteCommand հատկություններով: SelectCommand-ը օգտագործվում է բազայից տվյալների ստացման համար և հաճախ տրվում է կոնստրուկտորի միջոցով: Վերջին երեքը նախատեսված են հիշողության մեջ պահպանված տվյալները դեպի բազա ուղարկելու համար:
- SqlDataAdapter.Update հրամանը գտնելով փոփոխված տողերը, փոփոխությունները գրանցում է Connection օբյեկտով սահմանված բազայում՝ օգտագործելով XXXCommand հրամանները:

Insert, Update և Delete հրամանների ավտոմատ գեներացումը

SqlCommandBuilder դասը հնարավորություն է տալիս ավտոմատ կերպով գեներացնել հրամաններ՝ Select հրամանի հիման վրա.

```
using(SqlConnection connection = new SqlConnection(connectionString))
{
   connection.Open();
   string sqlString = "Select * from Products";
   SqlDataAdapter adapter = new SqlDataAdapter(sqlString, connection);
   DataSet dataSet = new DataSet();
   adapter.Fill(dataSet);

   this.dataGridView1.DataSource = dataSet.Tables[0];

   SqlCommandBuilder builder = new SqlCommandBuilder(adapter);
   adapter.UpdateCommand = builder.GetUpdateCommand();
   adapter.InsertCommand = builder.GetInsertCommand();
   adapter.DeleteCommand = builder.GetDeleteCommand();

   dataSet.Tables[0].Rows[0]["ProductName"] = "Lavash";

   adapter.Update(dataSet);
}
```

Transaction

Տրանզակցիա, ACID հատկությունը

- Տրանզակցիայի հատկությունների բնութագրման համար օգտագործվում է ACID տերմինը, որը նշանակում է Atomicity, Consistency, Isolation, Durability:
- **Atomicity** – ատոմարություն: Այս հատկությունը նշանակում է, որ տրանզակցիայի տակ կատարվող գործը պետք է լինի անբաժանելի: Այսինքն՝ կամ կատարվում է գործը ամբողջությամբ, կամ ոչինչ չի կատարվում:
- **Consistency** – համաձայնեցվածություն: Այս հատկությունը պահանջում է, որպեսզի համակարգի վիճակը տրանզակցիայից առաջ և հետո լինի իրական: Տրանզակցիայի կատարման ընթացքում թույլատրվում է միջանկյալ վիճակներ:
- **Isolation** – մեկուսացվածություն: Դա նշանակում է որ միաժամանակ կատարվող տրանզակցիաները չպետք է տեսնեն միմյանց միջանկյալ վիճակները:
- **Durability** – կայունություն: Տրանզակցիայի ավարտից հետո արդյունքը պարտադիր պետք է պահպանվի որևէ կայուն եղանակով: Օրինակ կոշտ սկավառակի վրա և այլն:

- *Ոչ բոլոր նշված հատկանիշներն են պարտադիր բոլոր տրանզայցիաների համար:*

Լոկալ տրանզակցիայի ստեղծումը Connection օբյեկտից

- DbConnection դասը օժտված է BeginTransaction() մեթոդով որը վերադարձնում է DbTransaction տիպի օբյեկտ:
- Յուրաքանչյուր DbCommand, որը պետք է կատարվի տրանզակցիայի մեջ, պետք է ասոցացվի տրանզակցիայի օբյեկտի հետ DbCommand.Transaction հատկանիշի միջոցով:

```
 string connectionString = @"Data Source=NOTEBOOK\SQLSERVER;Initial Catalog=northwind;Integrated Security=True";
using(SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();

    SqlTransaction transaction = connection.BeginTransaction(); // բացել տրանզակցիա տվյալ connection-ի համար
    try
    {
     SqlCommand command =
     new SqlCommand("UPDATE Products Set productName = 'Matnaqash' where productName = 'Lavash'", connection);

     command.Transaction = transaction; // կատարել տվյալ հրամանը տրանզակցիայով
     command.ExecuteNonQuery();

     transaction.Commit(); // եթե հասել ենք այստեղ, ուրեմն ամեն ինչ նորմալ է անցել, իրականացնել տրանզակցիան
    }
    catch
    {
     MessageBox.Show("Exception!");
     transaction.Rollback(); // Առաջացել է խնդիր, չեղարկել բոլոր փոփոխությունները բազայում
    }
}
```

System.Transactions.Transaction դասը

- System.Transactions.Transaction տիրույթում սահմանված են XXXTransaction անուններով դասեր, որոնք թույլ են տալիս ստեղծել և օգտագործել տարբեր հատկություններով օժտված տրանզակցիաներ: Բոլոր դասերի համար բազային է Transaction դասը, որի հիմնական հատկանիշներն ու մեթոդներն են.
  - Current – վերադարձնում է ընթացիկ տրանզակցիան:
  - TransactionInformation – վերադարձնում է TransactionInformation տիպի օբյեկտ, որը պարունակում է մանրամասն ինֆորմացիա տվյալ տրանզակցիայի վերաբերյալ:
  - Rollback() – ընդհատում է ընթացիկ տրանզակցիան և համակարգը վերադարձնում է մինչև տրանզակցիայի սկիզբը ունեցած վիճակին:
  - TransactionComplated – Իրենից ներկայացնում է *event,* որը կանչվում է տրանզակցիայի ավարտման ժամանակ:

CommitableTransaction դասը

- CommitableTransaction դասը միայ դասն է System.Transactions տիրույթում, որն օժտված է ֆիքսման (commit) մեթոդով:
- Այս դասը հիմնականում օգտագործվում է բաշխված տրանզակցիաներ ստանալու համար:
- Բաշխված են կոչվում այն տրանզակցիաները, որոնք իրականացվում է մի քանի connection-ների միջև (մի քանի սերվերների):
- Զգուշացում: Որպեսզի բաշխված տրանզակցիաները աշխատեն, անհրաժեշտ է, որ օպերացիոն համակարգում միացված լինի **Distributed Transaction Coordinator** սերվիսը

CommitableTransaction դասի կիրառման օրինակ

```
 private void UpdateProducts(int id, CommittableTransaction transaction)
 {
string connectionString = @"Data Source=NOTEBOOK\SQLSERVER;Initial Catalog=northwind;Integrated Security=True";
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
      connection.Open();
      connection.EnlistTransaction(transaction);
      SqlCommand command = new SqlCommand("UPDATE Products Set productName    = 'Matnaqash' where ProductID = " + id,
    connection);

      command.ExecuteNonQuery();
    }
}
 CommittableTransaction transaction = new CommittableTransaction();
transaction.TransactionCompleted += transaction_TransactionCompleted;
try
{
    for (int i = 1; i < 10; i++)
    {
      UpdateProducts(i, transaction);
    }

    transaction.Commit();
    }
    catch
    {
      transaction.Rollback();
    }
}
```

Ներառող (ambient) տրանզակցիա

- Ներառող տրանզակցիան թույլ է տալիս նշել աշխատանքային տիրույթ, որի ներսում իրականացվող ցանկացած միացում (connection) ավտոմատ կերպով ապահովվում է տրանզակցիայով:
- Ներառող տրանզակցիա ստանալու համար նախատեսված է TransactionScope դասը, որի հիմնական մեթոդներն են Complate() և Dispose():
- Complate() մեթոդը թույլ է տալիս արժեքավորել, այսպես կոչված, հաջող ավարտի բիթը (happy bit):
- Dispose() մեթոդի կանչի ժամանակ կատարվում է տրանզակցիայի ֆիքսում, եթե happy bit-ը արժեքավորված է, կամ կատարում է փոփոխությունների չեղարկում (Rollback)

TransactionScope դասի կիրառման օրինակ

```
private void UpdateProducts(int id)
{
    string connectionString = @"Data Source=NOTEBOOK\SQLSERVER;Initial Catalog=northwind;Integrated
    Security=True";
 using (SqlConnection connection = new SqlConnection(connectionString))
  {
     connection.Open();
     SqlCommand command = new SqlCommand("UPDATE Products Set productName = 'Matnaqash' where ProductID = "
    + id, connection);

     command.ExecuteNonQuery();
    }
}



using (TransactionScope scope = new TransactionScope())
{
    for (int i = 1; i < 10; i++)
     {
     UpdateProducts(i);
    }

    scope.Complete();
}
```

Ներդրված ներառող տրանզակցիաներ

- Եթե մեկ TransactionScope-ը ներառվում է մեկ այլ TransactionScope-ի մեջ, ապա դրանց փոխհարաբերությունները կարգավորվում են TransactionScopeOption թվարկման միջոցով, որի արժեքը փոխանցվում է TransactionScope դասի կոնստրուկտորին: Այն ընդունում է հետևյալ արժեքներից մեկը
- Required – եթե գոյություն ունի արտաքին տրանզակցիա, ապա օգտագործվում է այն, հակառակ դեպքում՝ ստեղծվում է նորը: Այդ դեպքում ֆիքսացիա տեղի է ունենում այն և միայն այն դեպքում, եթե բոլոր ներդրված TransactionScope օբյեկտները կանչում է Complate() մեթոդը:
- RequiresNew – անկախ արտաքին տրանզակցիայից ստեղծվում է նոր տրանզակցիա: Ե՛վ արտաքին և՛ ներքին տրանզակցիաները գործում են միմյանցից անկախ:
- Suppress – ներքին տրանզակցիան աշխատում է անկախ արտաքին տրանզակցիայից:

Տրանզակցիաների իզոլացումը

Տրանզակցիայի կիրառման ժամանակ առաջացող պոտենցիալ պրոբլեմներ

- Մի քանի տրանզակցիաների կիրառման դեպքում առաջացող խնդիրները բաժանվում են երեք խմբի:

1. **Կեղտոտ ընթերցում** (dirty read): Քանի որ տրանզակցիայի ժամանակ փոփոխվող տվյալները պոտենցյալ կարող են վերադարձվել իրենց նախորդ վիճակին, ապա այդ տվյալների կարդացումը տրանզակցիայի միջանկյալ իրավիճակում անվանում են “կեղտոտ”:
2. **Չվերականգնվող ընթերցում** (nonrepeatable reads): Այսպիսի ընթերցում տեղի է ունենում այն ժամանակ, երբ մեկ տրանզակցիայի ընթացքում տվյալները կարդացվում են, իսկ մեկ այլ տրանզակցիայի ընթացքում կատարվում են փոփոխություններ: Հետևաբար այդ տվյալները դարնում են չվերականգնվող:
3. **Ֆանտոմային ընթերցում** (phantom reading): Ֆանտոմային ընթերցում տեղի է ունենում այն դեպքում, երբ տրանզակցիայի ընթացքում կատարվում է որոշակի դիապազոնի մշակում (հիմնականում where օպերատորի դեպքում) իսկ այդ ընթացքում այլ տրանզակցիա կատարում է փոփոխություն այդ դիապազոնում:

- Նշված խնդիրների լուծման համար օգտագործվում է տրանզակցիաների իզոլացման մեխանիզմը:

Տրանզակցիաների իզոլացման տեսակները

- **ReadUncommitted** – այս դեպքում տրանզակցիաները միմյանցից չեն իզոլացվում: Այս դեպքում կարող են տեղի ունենալ նշված բոլոր պրոբլեմները:
- **ReadCommitted** – այս դեպքում կատարվում է տվյալ պահին փոփոխվող տվյալների բլոկավորում գրելու և կարդալու համար: Արդյունքում "կեղտոտ ընթերցման" պրոբլեմը չի առաջանում: Սակայն երբ կատարվում է տվյալների հերթական ընթերցում, ամեն հաջորդ տվյալի ընթերցման ժամանակ նախորդի բլոկավորումը հանվում է: Արդյունքում կարող է առաջանալ չկրկնվող ընթերցման պրոբլեմը:
- **RepeatableRead** – այս իզոլյացիայի դեպքում տվյալների կարդացումը բլոկավորվում է մինչև տրանզակցիայի ավարտը: "Կեղտոտ" ընթերցումը և չկրկնվող ընթերցումը բացառվում են, սակայն մնում է ֆանտոմային ընթերցման խնդիրը:
- **Serializable –** տվյալների դիապազոնը բլոկավորվում է այնքան ժամանակ, քանի դեռ տրանզակցիան չի ավարտվել: Արդյունքում բոլոր երեք խնդիրները բացառվում են:
- **Snapshot** – այս դեպքում բլոկավորումը արդեն փոփոխված տվյալներից հանվոմ է, և այլ տրանզակցիաները կարող են կարդալ դրանք:
- **Unspecified –** այլ իզոլյացիայի տարբերակ: Օգտագործվում է այն դեպքում, երբ տվալների պրովայդերը ունի իզոլացման իր սեփական տարբերակը :
- **Chaos -** նման է **ReadUncommitted** տարբերակին, սակայն այս դեպքում կարող են կարդացվել նաև տվյալ պահին բլոկավորված տվյալները: