Introduction to LINQ (2):

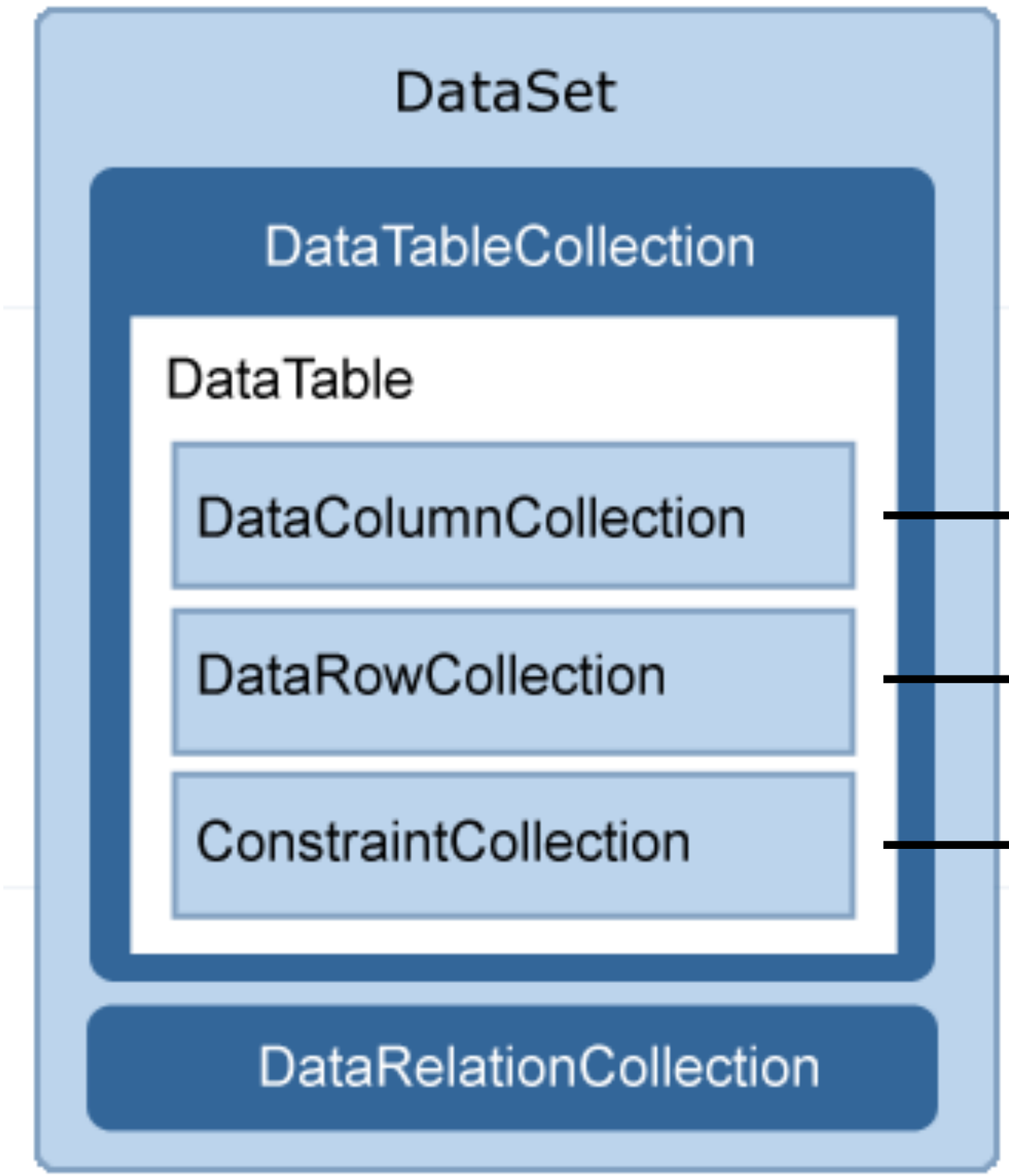
Language-Integrated Query

We have seen: LINQ to OBJECTS

now:

LINQ to DataSets

DataSets



Libraries:

```
System.Data.DLL
System.Data.DataSetExtensions.DLL
```

Namespaces:

```
System.Linq
System.Data
```

The DataSet can be seen as a collection of data residing in memory and organized in a relational way.

Creation of a DataTable

```
//Create the table
DataTable DT = new DataTable(“DT_Name”);

//Create the columns (fields)
DT.Columns.Add(“Name”,typeof(String));
DT.Columns.Add(“Year”,typeof(Int16));
… …

//Add data
DT.Rows.Add(“John”,1977);
DT.Rows.Add(“Jack”,1983);
… …

//Create a DataSet
DataSet myData = new DataSet(“DS_Name”);
myData.Tables.Add(DT);
```

LINQ Query on DataSets

with reference to the example in the previous slide:

```
//Query (. notation + lambda expression in this case)
var data = myData.Tables[“DT_Name”].Rows.Cast<DataRow>()
            .Where(ThisEntry =>
                    ThisEntry.ItemArray[0].ToString()==“John”);

//Display the results:
foreach (DataRow row in data){
    …
}
```

Exercise: rewrite the query in the other LINQ style (from-where-select)

Display Results with DataGridView

```
EnumerableRowCollection<DataRow> query =

data.Tables["datatable"].AsEnumerable().Where(entry =>
entry.ItemArray[1].ToString() == "John");

DataView view = data.AsDataView();
```

```
my_datagridview.DataSource = view;
```

Copying a DataTable

A common operation is to query a table and then copy the result into another table. Modifying the example in the last slide:

```
EnumerableRowCollection<DataRow> query =

data.Tables["datatable"].AsEnumerable().Where(entry =>
entry.ItemArray[1].ToString() == "John");

//Copy the results into a new Table
DataTable results_table = query.CopyToDataTable();

//The table can be further modified:
results_table.Columns.Remove(“col_name”);
results_table.Columns[“Name”].ColumnName = “FirstName”;

DataView view = results_table.AsDataView();

my_datagridview.DataSource = view;
```

Multiple Tables and Comparisons

Let's suppose we have two DataTables: **`table1`** and **`table2`** in a dataset **`data`**.

```
//Query combining tables:
IEnumerable<DataRow> combined_table =
    data.Tables["table1"].AsEnumerable()
    .Concat(data.Tables["table2"].AsEnumerable())
    .OrderBy(row => row.Field<String>("Name"));

//Perform the comparison
IEnumerable<DataRow> result =
    combined_table.Distinct(DataRowComparer.Default);

//Display
foreach (DataRow row in result)
    mydatagridview.Rows.Add(row.ItemArray);
```

The comparison operators:

`Distinct, Except, Intersect, Union, SequenceEqual`

Except and Intersect

```
//Except query: filters out the different rows
IEnumerable<DataRow> combined_table =
    data.Tables[“table1”].AsEnumerable()
    .Except(data.Tables[“table2”].AsEnumerable(),
    DataRowComparer.Default);
```

```
//Intersect query: filters out the common rows
IEnumerable<DataRow> combined_table =
    data.Tables[“table1”].AsEnumerable()
    .Intersect(data.Tables[“table2”].AsEnumerable(),
    DataRowComparer.Default);
```

Union and SequenceEqual

```
//Union query: like Distinct, but no concatenation needed
IEnumerable<DataRow> combined_table =
    data.Tables[“table1”].AsEnumerable()
    .Union(data.Tables[“table2”].AsEnumerable(),
    DataRowComparer.Default).
    OrderBy(row => row.Field<String>(“Name”));
```

```
//Intersect query: outputs a boolean describing
//                 if the two row sequences were equal.
IEnumerable<DataRow> combined_table =
    data.Tables[“table1”].AsEnumerable()
    .SequenceEqual(data.Tables[“table2”].AsEnumerable(),
    DataRowComparer.Default);
```