Systems Programming & Scripting

Lecture 12: Database access in C# and LINQ

ADO.NET

- ADO.NET provides a direct interface to a database.
- The interface is database-specific.
- ADO.NET uses a conventional, shallow embedding of SQL commands into C# as host language, i.e. SQL commands are composed as strings
- A more advanced, deep embedding of SQL commands is provided by LINQ, i.e. SQL commands a language constructs

Structure of database access

- To access a database with ADO.NET the following steps are necessary:
  - Connect to a database
  - Compose an SQL query
  - Issue the query
  - Retrieve and process the results
  - Disconnect from the database.

ADO.NET Example

- To connect to a database, a connection string has to specify location, account, password etc. (fill in user id and pwd)

```
using MySql.Data.MySqlClient;
string cstr = "Server=anubis;Database=test;User ID=;Password=";
MySqlConnection dbcon;
       try {
         dbcon = new MySqlConnection(cstr);
         dbcon.Open();
       }
       catch (MySql.Data.MySqlClient.MySqlException ex) { … }
```

ADO.NET Example (cont'd)

- Next, compose an SQL query as a string
- This can be any SQL operation
- Depending on the underlying database, SQL extensions might be available.

```
MySqlCommand dbcmd = dbcon.CreateCommand();

string sql =
            "SELECT A_ID, A_FNAME, A_LNAME " +
            "FROM authors";
dbcmd.CommandText = sql;
```

ADO.NET Example (cont'd)

- Next, issue the query, and process the result, typically in a while loop.

```
MySqlDataReader reader = dbcmd.ExecuteReader();

while(reader.Read()) {
  string FirstName = (string) reader["A_FNAME"];
  string LastName = (string) reader["A_LNAME"];
  Console.WriteLine("Name: " + FirstName + " " + LastName);
}
```

ADO.NET Example (cont'd)

- Finally, clean-up and disconnect.

```
reader.Close();
reader = null;
dbcmd.Dispose();
dbcmd = null;
dbcon.Close();
dbcon = null;
```

LINQ

- Language Integrated Query (LINQ) is a more advanced way to interact with databases.
- It's a new feature with C# 3.0 onwards.
- It provides SQL-like commands as language extensions, rather than composing SQL queries as strings (*deep embedding*)
- It can also be used to access other forms of data, such as XML data or compound C# data structures.

LINQ Example

- The same example as before, written in LINQ is much simpler.
- First, classes, representing the tables of the database are defined.

```
[Table(Name = "authors")]
public class Authors
{
  [Column]
  public int A_ID   { get ; set ; }
  [Column]
  public string A_FNAME  { get ; set ; }
  [Column]
  public string A_LNAME  { get ; set ; }
}
```

LINQ Example (cont'd)

- Next, a connection is established, using a connection string similar to ADO.NET.

```
DataContext db = new DataContext("Data Source = .\\MySql;" +
                    "Initial Catalog=test;Integrated Security=True");
```

```
DataContext db = new DataContext(connStr);
```

LINQ Example (cont'd)

- The main advantage of LINQ is the simplified way of performing queries.
- Note, that SQL-like commands such as select, from etc are directly available

```
Table<Authors> AuthorTable = db.GetTable<Authors>();
List<Authors> dbQuery = from author in Authors select author ;

foreach (var author in dbQuery) {
   Console.WriteLine("Author: "+author.A_FNAME+" “+
                     author.A_LNAME);
}
```

Querying in-memory Data

- LINQ can also be used to query in-memory data, such as XML data or compound C# data structures.
- This results in more uniform and succinct code.
- Using LINQ in this way requires several advanced language features.
- It is an alternative to using standard mechanisms of traversing data structures such as iterators

Example

- Assume we have a list of books:

```
List<Book> booklist = new List<Book> {
        new Book { Title = "Learning C#"
                , Author = "Jesse Liberty"
                , Publisher = "O'Reilly"
                , Year = 2008
                },
        new Book { Title = "Programming C#"
                , Author = "Jesse Liberty"
                , Publisher = "O'Reilly"
                , Year = 2008
                },
        new Book { Title = "Programming PHP"
                , Author = "Rasmus Lerdorf, Kevin Tatroe"
                , Publisher = "O'Reilly"
                , Year = 2006
                },
        };
```

Example

- The conventional way to iterate over the list looks like this:

```
foreach (Book b in booklist) {
    if (b.Author ==  "Jesse Liberty") {
      Console.WriteLine(b.Title + " by " + b.Author);
    }
       }
```

Example

In contrast, the LINQ-style iteration looks like an SQL query and is shorter:

```
IEnumerable<Book> resultsAuthor =
     from b in booklist
     where b.Author == "Jesse Liberty"
     select b;

     Console.WriteLine("LINQ query: find by author ...");
     // process the result
     foreach (Book r in resultsAuthor) {
       Console.WriteLine(r.Title + " by " + r.Author);
     }
```

Example

To avoid returning entire book results from the query we can use anonymous types and just return title and author:

```
var resultsAuthor1 =// NB: this needs to infer the type (anonymous!)
     from b in booklist
     where b.Author == "Jesse Liberty"
     select new { b.Title, b.Author} ; // NB: anonymous type here!


// process the result
foreach (var r in resultsAuthor1) {
   Console.WriteLine(r.Title + " by " + r.Author);
}
```

Example

Lambda expressions can be used to shorten the query even further:

```
var resultsAuthor2 =  // NB: lambda expression here
 booklist.Where(bookEval => bookEval.Author == "Jesse Liberty");

 // process the result
 foreach (var r in resultsAuthor2) {
    Console.WriteLine(r.Title + " by " + r.Author);
 }
```

Example

We can sort the result by author:

```
var resultsAuthor3 =
     from b in booklist
     orderby b.Author
     select new { b.Title, b.Author} ; // NB: anonymous type here!

Console.WriteLine("LINQ query: ordered by author ...");
// process the result
foreach (var r in resultsAuthor3) {
    Console.WriteLine(r.Title + " by " + r.Author);
}
```

Example

We can join tables like this:

```
var resultList4 =
         from b in booklist
         join p in purchaselist on b.Title equals p.Title
         where p.Quantity >=2
     select new { b.Title, b.Author, p.Quantity } ;

Console.WriteLine("LINQ query: ordered by author ...");
// process the result
foreach (var r in resultList4) {
  Console.WriteLine(r.Quantity + " items of " + r.Title
                    + " by " + r.Author);
}
```

Summary

- C# supports 2 ways of querying databases:
  - ADO.NET with SQL queries as strings
  - LINQ with SQL commands embedded into the language
- ADO.NET is older and more robust
- LINQ is newer and easier to use
- LINQ can also be used to traverse in memory data structures.