Database Application Development in .NET: C#, Entity Framework, Linq

CS430/630
for Lecture 15
Microsoft .NET: has a VM much like Java's

- Java's JVM
  - You know that Java runs in its JVM, Java virtual machine.
  - Other languages run there as well ([Wikipedia list](https://en.wikipedia.org/wiki/List_of_Java_programs))
  - The JVM provides a wall between the Java world and the OS “underneath”
  - It can be called a “sandbox” ([Wikipedia](https://en.wikipedia.org/wiki/Java_virtual_machine))
  - The JVM is key to portability across hardware and OSs

- .NET is the Microsoft platform providing its own virtual machine, the CLR, for Common Language Runtime ([Wikipedia](https://en.wikipedia.org/wiki/Common_Language_Runtime))
  - Similarly providing a sandbox for safe execution, portability
  - But portability is limited to Microsoft OSs. (maybe not in future?)
  - Open source .NET project Mono, but not widespread in use
  - Multiple languages are provided for this environment: VB, C#, ..., not Java
  - C# is much like Java
Like Java, .NET compilation is to “bytecode”, then executed in the virtual machine.
In a C# .NET program, how can we access a DB?

• ODBC (old API, model for JDBC), connects to many DBs
• ADO.NET (OO approaches)
  • Introduced with .NET (‘02): Basic objects
  • Uses completely new model over older ActiveX ADO (history)
  • Command (query), DataSet object (or DataReader for large data)
  • Can use underlying OLDB connection, so works with many DBs
• LINQ (‘07)
  • LINQ to Datasets (i.e. DataSets of plain ADO.NET)
  • LINQ to SQL (‘07), Limited to MS SQL Server DB currently
• Entity Framework (ORM: object-relational mapping, like Java’s JPA) (‘08)
  • LINQ to Entities (of the Entity Framework)
  • Can access other databases (Oracle article)
  • .NET Core: 2017, cross-platform, some .NET features missing
• NHibernate (about ‘07): port of premier Java ORM Hibernate
  • Open source, connects to many databases
Microsoft Data Access History from www.c-sharpcorner.com

Accessing before ‘90s
- RAW Data, Direct APIs

Accessing data in 1990
- ODBC (abstract call-level)

Accessing pre .Net
- OLE DB, ADO (object level)

Accessing data in .Net
- ADO.NET, Datasets, DataReaders

Today
- Linq, Entity Framework... O/RM?
LINQ to SQL vs. Entity Framework

• Two groups at Microsoft developed these two approaches in parallel
  • Some competition, confusion in having competing solutions
  • Fusion of two in LINQ to Entities, but that left LINQ to SQL the odd one
  • Note that LINQ to SQL is only supported on MS SQL Server
  • LINQ to Entities can use Oracle, other DBs

• Some evidence MS is downplaying LINQ to SQL
  • From MSDN ASP.NET Data Access Options (surprised to see NHibernate here!)
    • The most commonly used ORMs that work with ASP.NET are the following:
      ➢ The ADO.NET Entity Framework is the main ORM that Microsoft provides for the .NET Framework.
      ➢ LINQ to SQL is a legacy ORM that Microsoft provides. (Don't confuse LINQ to SQL with LINQ. For information about LINQ, see LINQ versus SQL later in this topic.)
      ➢ NHibernate is an open source ORM for the .NET Framework.
Language trends: from https://insights.stackoverflow.com/trends
DB Access Frameworks/Libraries
What is LINQ?

• From MSDN:
• General-purpose query facilities added to the .NET Framework apply to all sources of information, not just relational or XML data.
• This means we can use LINQ to process date in lists, maps, or arrays, as well as DB data and XML data.
• It’s an extension of the languages themselves, not “just” an API into extra classes.
• Also no preprocessor is needed. The one compiler step is all that’s needed.
• This ability to change the underlying language is of course helped by the fact that Microsoft provides all the software involved, except external databases.
• This approach has inspired Java support for similar features: see Java Object Oriented Queries (jOOQ) and in Wikipedia.
LINQ used with C# program data structures

using...

class app {
    static void Main() {
        string[] names = { "Burke", "Connor", "Frank", "Everett", "Albert", "George", "Harris", "David" };

        IEnumerable<string> query = from s in names
                                      where s.Length == 5
                                      orderby s
                                      select s.ToUpper();

        foreach (string item in query)
            Console.WriteLine(item);
    }
}

Output: BURKE DAVID FRANK
Lambda Expressions (functions without names, aka anonymous functions)

Func<string, bool> filter = s => s.Length == 5;

// function filter takes a string s, executes
// “s.Length == 5” on it, returns the bool
// However, we can use “s => s.Length == 5”
// as a function without giving it a name

Func<string, string> extract = s => s;
Func<string, string> project = s => s.ToUpper();
// same query as on last slide:
IEnumerable<string> query = names.Where(filter)
    .OrderBy(extract)
    .Select(project);
Using Lambda Expressions

Same query again:

```csharp
IEnumerable<string> query =
    names.Where(s => s.Length == 5)
    .OrderBy(s => s)
    .Select(s => s.ToUpper());
```

• Note that Where appears as a method of names, itself an array. The methodology of providing that capability is another feature of LINQ.

• Also, Where returns IEnumerable<string>, which has OrderBy, and that also returns IEnumerable<string>, as does Select: this style of programming is called “method chaining” or a “train wreck”
Nameless objects too!

var value = new {
    Name = "Brian Smith", Age = 31, CanCode = false
};

Note that these objects have no “behavior”, only data members, like C structs.
These objects do have Equals and GetHashCode methods.
LINQ to SQL: marked up C# classes

```csharp
public class Person {
    [Table(Name="People")]
    public string Name;

    [Column(DbType="nvarchar(32) not null", Id=true)]
    public int Age;

    [Column]
    public bool CanCode;
}

public class Order {
    [Table(Name="Orders")]
    public string OrderID;

    [Column( DbType="nvarchar(32) not null", Id=true)]
    public string Customer;

    [Column]
    public int? Amount;
}
```
Application using these tables...

// establish a query context over ADO.NET sql connection
DataContext context = new DataContext(
    "Initial Catalog=petdb;Integrated Security=sspi");

// grab variables that represent the remote tables that // correspond to the Person and Order CLR types
Table<Person> custs = context.GetTable<Person> ();
Table<Order> orders = context.GetTable<Order> ();
// build the query, a join--
var query = from c in custs
            from o in orders
            where o.Customer == c.Name
            select new {
                c.Name,
                o.OrderID,
                o.Amount,
                c.Age
            };

// execute the query
foreach (var item in query)
    Console.WriteLine("{0} {1} {2} {3}",
                     item.Name, item.OrderID,
                     item.Amount, item.Age);
LINQ to Entities Example from stackoverflow

**Tables**

Travel_Request (Request_ID, Resource_ID) – FK to Resource  
Resource(Resource_ID, Resource_Name)

**Query:** I would like to add the Resource_Name to the list when returning all the TRAVEL_REQUESTS

**Solution:**

```csharp
var data = from t in Travel_Request
            join r in Resource on t.Resource_ID equals r.Resource_ID
            select new
            {
              RequestId = t.Request_ID,
              ResourceId = t.Resource_ID,
              ResourceName = r.Resource_Name
            };
```
LINQ to Entities Example

Tables
Travel_Request (Request_ID, Resource_ID) – FK to Resource
Resource(Resource_ID, Resource_Name)

Query: I would like to add the Resource_Name to the list when returning all the TRAVEL_REQUESTS

Second Solution: If you already have an EF association then it could simply be:

```csharp
var data = from t in Travel_Request
            select new
            {
                RequestId = t.Request_ID,
                ResourceId = t.Resource_ID,
                ResourceName = t.Resource.Resource_Name
            };

This hints at the power of ORM systems to provide a whole object graph for the convenience of programmers, from DB data.