**Oriette Ramos**

**Lesson Plan 33**

**ISTA-220**

1. What is the difference in the purposes of SQL and LINQ? In other words, why do we need two different query languages? Does LINQ replace SQL? Does SQL make LINQ unnecessary?  
   **LINQ provides syntax and semantics very reminiscent of SQL and with many of the same advantages. You can change the underlying structure of the data being queried without needing to change the code that actually performs the queries. You should be aware that although LINQ looks similar to SQL, it is far more flexible and can handle a wider variety of logical data structures. For example, LINQ can handle data organized hierarchically, such as that found in an XML document. They are both necessary because SQL queries a database (storage), and LINQ queries data in memory.**
2. What is the one essential requirement for the data structures used with LNQ? Why is this requirement important?  
   **LINQ requires the data to be stored in a data structure that implements the IEnumerable or IEnumerable<T> interface. This is important because the LINQ methods are extension methods of the Enumerable class.**
3. Were does the LINQ Select() method live?  
   **The LINQ Select() method is an extension method of the Enumerable class. The Enumerable class is located in the System.Linq namespace and provides a substantial set of static methods for querying objects that implement the generic IEnumerable<T> interface.**
4. (Select) Explain, token by token, each token in this line of code:

IEnumerable<string> customerFirstNames = customers.Select(cust => cust.FirstName);

**IEnumerable<string> is declaring an enumerable collection of strings and customerFirstNames is the variable name for the collection. The "customers" part of customers.Select is an array of customer objects. The "Select" part of customers.Select is the LINQ Select method. The variable cust is the parameter passed in to the method. You can think of cust as an alias for each row in the customers array. The compiler deduces this from the fact that you are calling the Select method on the customers array. The => is a lambda expression used to define an anonymous method. cust.FirstName returns the value of the FirstName property of each customer object. To sum it up, the code uses the Select method of the customers array to generate an IEnumerable object named customerFirstNames that contains the first names of each customer.**

1. (Filter) Explain, token by token, each token in this line of code:
2. IEnumerable<string> usCompanies = addresses.Where(addr =>

String.Equals(addr.Country, "United States")).Select(usComp => usComp.CompanyName);

**Creates an enumerable collection of strings named usCompanies and populates it with the data in the CompanyName property of each object in the addresses array with a filter applied where the Country property of each object returned equals "United States".**

1. (OrderBy) Explain, token by token, each token in this line of code:
2. IEnumerable<string> companyNames = addresses.OrderBy(addr =>

addr.CompanyName).Select(comp => comp.CompanyName);

**Creates an enumerable collection of strings named usCompanies and populates it with the data in the CompanyName property of each object in the addresses array ordered alphabetically.**

1. (Group) Explain, token by token, each token in this line of code:

var companiesGroupedByCountry = addresses.GroupBy(addrs => addrs.Country);

**Creates an implicitly typed collection named companiesGroupedByCountry and populates it with an enermable set that contains all the fields in the original source collection, but the objects are ordered into a set of enumerable collections based on the Country property.**

1. (Distinct) Explain, token by token, each token in this line of code:

int numberOfCountries = addresses.Select(addr => addr.Country).Distinct().Count();

**Creates an integer named numberOfCountries and assigns it the value returned by counting the distinct number of values found in the Country property of each object in the source.**

1. (Join) Explain, token by token, each token in this line of code:
2. var companiesAndCustomers =
3. customers.Select(c =>
4. new { c.FirstName, c.LastName, c.CompanyName }).Join(addresses, custs =>
5. custs.CompanyName, addrs => addrs.CompanyName, (custs, addrs) =>

new {custs.FirstName, custs.LastName, addrs.Country });

**Creates an implicitly typed collection named companiesAndCustomers and populates it with the values from the FirstName and LastName properties of the customers array and the Country property of the addresses array where the CompanyName properties match in the two objects.**

1. Explain the difference between a deferred collection and a static, cached collection.  
   **When you use LINQ to define an enumerable collection, either by using the LINQ extension methods or by using query operators, you should remember that the application does not actually build the collection at the time that the LINQ extension method is executed; the collection is enumerated only when you iterate over it. This means that the data in the original collection can change in the time between the execution of a LINQ query and when the data that the query identifies is retrieved; you will always fetch the most up-to-date data. You can force the evaluation of a LINQ query when it is defined and generate a static, cached collection. This collection is a copy of the original data and will not change if the data in the collection changes.**