LINQ AND DATABASE

LINQ to ADO.NET

LINQ to ADO.NET includes different flavors of LINQ to query data from different databases like as Microsoft SQL Server, Oracle, and others.

- LINQ to SQL
- **LINQ to DataSet**
- **LINQ to Entities**

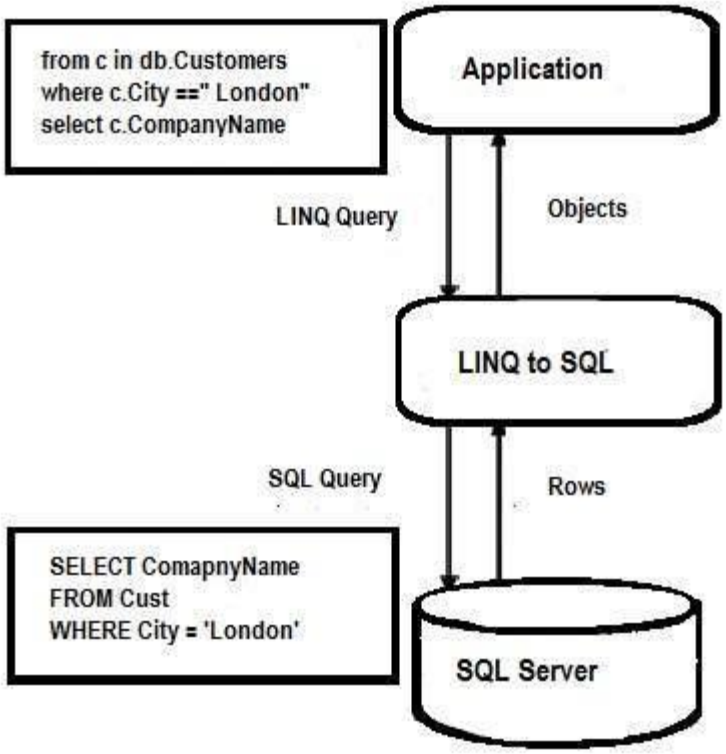
Other

- **LINQ to XML**
- **PLINQ (Parallel LINQ)**

LINQ to SQL

- It is specifically designed for working with Sql Server database.
- It provides run-time infrastructure for managing relational data as objects.
- It also supports transactions, views and stored procedures.
- It is an object-relational mapping (ORM) framework that allow 1-1 mapping of Sql Server database to .net classes.
- In this mapping the classes that match the database table are created automatically from the database itself and we can use these classes immediately.

LINQ to SQL


from c in db.Customers
where c.City ==" London"
select c.CompanyName
Application
LINQ Query
Objects
LINQ to SQL
SQL Query
Rows
SELECT ComapnyName
FROM Cust
WHERE City = 'London'
SQL Server

LINQ to SQL

- The most fundamental elements in the LINQ to SQL object model and their relationship to elements in the relational data model are summarized in the following table:

LINQ to SQL object model	Relational data model
Entity Class	Table
Class member	Column
Association	Foreign key relationship
Method	stored Procedure or Function

LINQ to SQL

- Objects are linked to relational data by decorating normal classes with attributes.
- Two of the most important attributes are Table and Column:
  - The name of the class will be used for the name of the table.
  - The Column attribute is used to decorate fields or properties of an entity class.

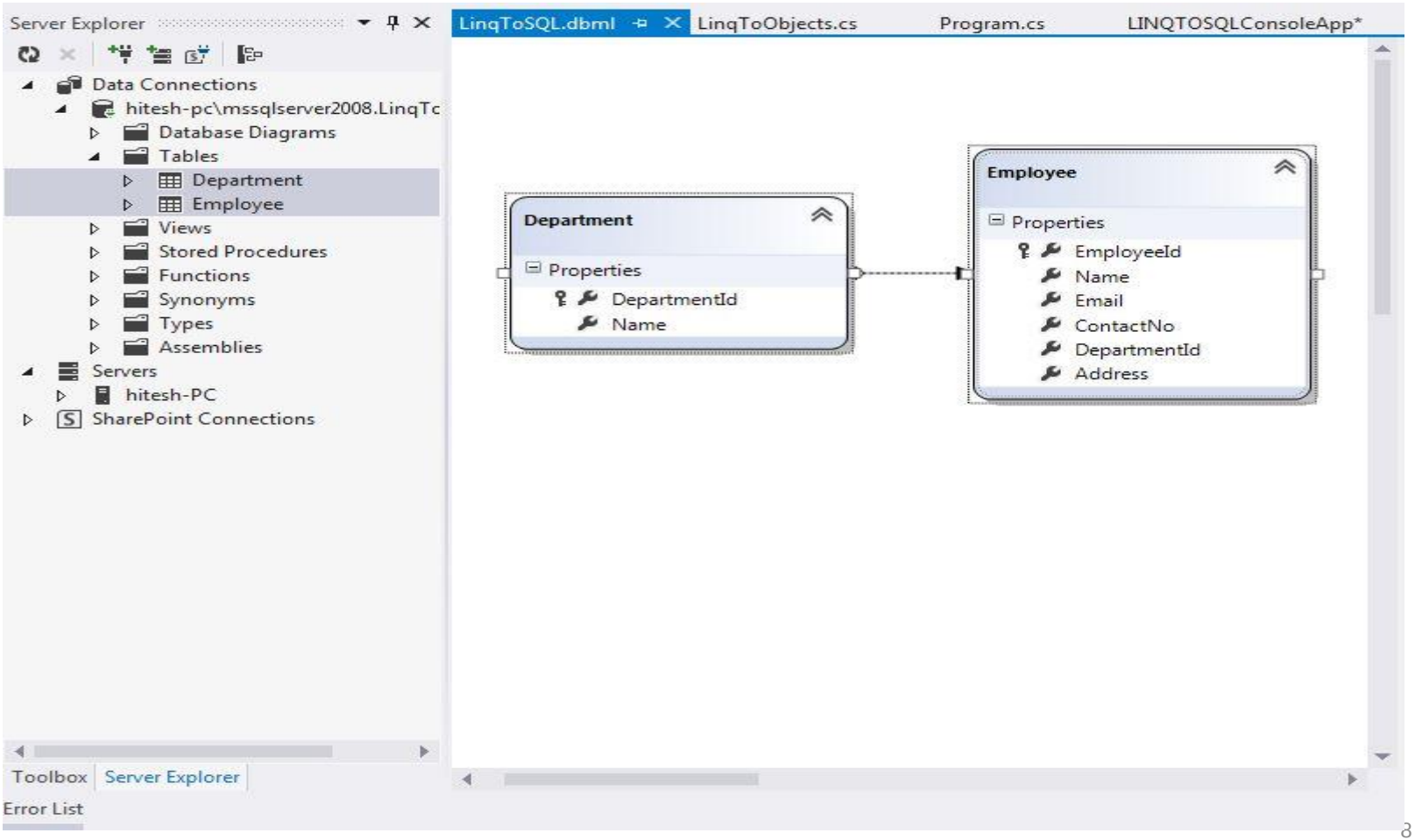
LINQ to SQL

`DataContext` Class

- All LINQ to SQL queries occur via a DataContext class, which controls the flow of data between the program and the database.
- A specific `DataContext` derived class, which inherits from the class `System.Data.Linq.DataContext`, is created when the LINQ to SQL classes representing each row of the table are generated by the IDE.
- This derived class has properties for each table in the database, which can be used as data sources in LINQ queries.
- Any changes made to the `DataContext` can be saved back to the database using the `DataContext`'s SubmitChanges method, so with LINQ to SQL you can modify the database's contents.

LINQ to SQL

Tables to Classes



Insert, Update and Delete using LINQ To SQL

```
LinqToSQLDataContext db = new LinqToSQLDataContext(connectString);

//Create new Employee
Employee newEmployee = new Employee();
newEmployee.Name = "Michael";
newEmployee.Email = "yourname@companyname.com";
newEmployee.ContactNo = "343434343";
newEmployee.DepartmentId = 3;
newEmployee.Address = "Michael - USA";

//Add new Employee to database
db.Employees.InsertOnSubmit(newEmployee);

//Save changes to Database.
db.SubmitChanges();
```

Insert, Update and Delete using LINQ To SQL

```
        //Get new Inserted Employee
        Employee insertedEmployee = db.Employees.FirstOrDefault(e
=>e.Name.Equals("Michael"));


        Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2},
ContactNo = {3}, Address = {4}",
                          insertedEmployee.EmployeeId,
insertedEmployee.Name, insertedEmployee.Email,
                          insertedEmployee.ContactNo,
insertedEmployee.Address);
```

Update using LINQ To SQL

```
  string connectString =
System.Configuration.ConfigurationManager.ConnectionStrings["LinqToSQLDBConnectionString"].ToString(
);


      LinqToSQLDataContext db = new LinqToSQLDataContext(connectString);

      //Get Employee for update
      Employee employee = db.Employees.FirstOrDefault(e =>e.Name.Equals("Michael"));

      employee.Name = "George Michael";
      employee.Email = "yourname@companyname.com";
      employee.ContactNo = "99999999";
      employee.DepartmentId = 2;
      employee.Address = "Michael George - UK";

      //Save changes to Database.
      db.SubmitChanges();
```

Update using LINQ To SQL

```
//Get Updated Employee
    Employee updatedEmployee =
db.Employees.FirstOrDefault(e =>e.Name.Equals("George
Michael"));

    Console.WriteLine("Employee Id = {0} , Name = {1}, Email
= {2}, ContactNo = {3}, Address = {4}",
                updatedEmployee.EmployeeId,
updatedEmployee.Name, updatedEmployee.Email,
                updatedEmployee.ContactNo,
updatedEmployee.Address);
```

Delete using LINQ To SQL

```
  string connectString =
System.Configuration.ConfigurationManager.ConnectionStrings["LinqToSQLDBCon
nectionString"].ToString();

      LinqToSQLDataContext db = newLinqToSQLDataContext(connectString);

      //Get Employee to Delete
      Employee deleteEmployee = db.Employees.FirstOrDefault(e
=>e.Name.Equals("George Michael"));

      //Delete Employee
      db.Employees.DeleteOnSubmit(deleteEmployee);

      //Save changes to Database.
      db.SubmitChanges();
```

Delete using LINQ To SQL

```
//Get All Employee from Database
    var employeeList = db.Employees;
    foreach (Employee employee in employeeList)
    {
      Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3}",
                        employee.EmployeeId, employee.Name, employee.Email, employee.ContactNo);
```

LINQ to Entities

- In many ways it is very similar to LINQ to SQL.
- It uses a conceptual Entity Data Model (EDM)
- The ADO.NET Entity Framework has been improved in .NET framework 4.0 to query any database like Sql Server, Oracle, MySql, DB2 and many more.

Insert, Update and Delete using LINQ To Entities

```
using (LinqToSQLDBEntities context = new LinqToSQLDBEntities())
      {
        //Get the List of Departments from Database
        var departmentList = from d in context.Departments
        select d;

        foreach (var dept in departmentList)
        {
          Console.WriteLine("Department Id = {0} , Department Name = {1}",
                    dept.DepartmentId, dept.Name);
        }

        //Add new Department
        DataAccess.Department department = new DataAccess.Department();
        department.Name = "Support";

        context.Departments.Add(department);
        context.SaveChanges();
```

Insert, Update and Delete using LINQ To Entities

```
Console.WriteLine("Department Name = Support is inserted in Database");


        //Update existing Department
        DataAccess.Department updateDepartment = context.Departments.FirstOrDefault(d
=>d.DepartmentId == 1);
        updateDepartment.Name = "Account updated";
        context.SaveChanges();


        Console.WriteLine("Department Name = Account is updated in Database");


        //Delete existing Department
        DataAccess.Department deleteDepartment = context.Departments.FirstOrDefault(d
=>d.DepartmentId == 3);
        context.Departments.Remove(deleteDepartment);
        context.SaveChanges();


        Console.WriteLine("Department Name = Pre-Sales is deleted in Database");
```

Insert, Update and Delete using LINQ To Entities

```
//Get the Updated List of Departments from Database
        departmentList = from d in context.Departments
        select d;

        foreach (var dept in departmentList)
        {
          Console.WriteLine("Department Id = {0} ,
Department Name = {1}",
                        dept.DepartmentId, dept.Name);
        }
      }
```

LINQ to DataSet

- It is an easy and faster way to query data cached in a DataSet object. It also allow LINQ to query over any database that can be query with Ado.Net.

LINQ to XML

- It provides an improved XML programming interface.
- Using this we can query, modify xml document and also save document after modification.

PLINQ (Parallel LINQ)

- It extends LINQ to Objects with a new parallel programming library.
- Using this, we can break/split up a query to execute simultaneously/parallel on different processors.