**Entity Framework**

Code First – CRUD

* Entities & DbContext API (note: all have auto property backing field)

    public class Blog

    {

        public int BlogId { get; set; }  // primary key

        public string Title { get; set; }

        public string Blogger { get; set; }

        public string Email { get; set; }

        public string Url { get; set; }

        public DateTime DateCreated { get; set; }

**// 1-m Relationship**

        public virtual **ICollection**<Post> Posts { get; set; }

    }

    public class Post

    {

        public int PostId { get; set; } // primary key

        public string Title { get; set; }

        public DateTime DateCreated { get; set; }

        public string Content { get; set; }

**// Foreign Key, 2 different ways to navigate**

        public virtual int BlogId { get; set; }

        public virtual Blog Blog { get; set; }

        // 1-m Relationship

        public virtual ICollection<Comment> Comments { get; set; }

    }

    public class Comment

    {

        public int CommentId { get; set; }  // primary key

        public string Content { get; set; }

        public DateTime DateCreated { get; set; }

        public string User { get; set; }

        // Foreign Key, 2 different ways to navigate

        public virtual int PostId { get; set; }

        public virtual Post Post { get; set; }

    }

    public class BlogSiteContext : **DbContext**

    {

        public **DbSet**<Blog> Blogs { get; set; }

        public DbSet<Post> Posts { get; set; }

        public DbSet<Comment> Comments { get; set; }

    }

* Usage
  + **Begin**: **using (var ctx = new BlogSiteContext())** { … CRUD operations … }
  + **C**: When newing up, **NO NEED** to set **int PrimaryKey, ForeignKey & Relationship**!
    - ctx.**Blogs.Add**( new Blog { … });
    - ctx.Posts.Add(new Post { …});
    - ctx.Comments.Add(new Comment { … });
  + **R**: Use **Find** by key / **LINQ** to Entity.
    - Post thePost = Ctx.Posts**.Find(theKey)**;
    - var queryResult = **from p in ctx.Posts** where …. select p;
  + **U**
    - **thePost.MyProperty** = …;
  + **D**
    - ctx.**Posts.Remove(thePost)**;
  + **End**: **var recordsAffected = ctx.SaveChanges();**

DB Migration

* Make sure EF 4.3+
* Package Manager Console
  + PM> Eanble-Migrations
    - Create a folder “Migration”. One class for seeding Database. Another class w/ Up() / Down(). Up: the current DB State.
  + Any Model C# class changes
    - PM> Add-Migration someDesc
    - PM> Update-Database
  + Seed data at individual Migration
    - void Up() { … Sql(“UPDATE myTable SET myColumn=…”); }
  + Rollback DB Schema Changes
    - PM> Update-Database –TargetMigration:”someDesc”
    - Roll ALL THE WAY BACK
      * PM> Update-Database –TargetMigration:$InitialDatabase
  + Generate SQL for deployment
    - PM> Update-Database –Script – SourceMigration:$InitialDatabase –TargetMigration:”someDesc”
    - // if you left out TargetMigration, it will use the latest migration as the target!
  + Custom Migration
    - <http://blogs.msdn.com/b/adonet/archive/2012/02/09/ef-4-3-code-based-migrations-walkthrough.aspx>
    - “Customizing Migration” Section

Re-create DB

Db Connection

* <http://blogs.msdn.com/b/adonet/archive/2011/01/27/using-dbcontext-in-ef-feature-ctp5-part-2-connections-and-models.aspx>

Configure over Convention

* Add DataAnnotations Reference
* [Key], []
* <http://msdn.microsoft.com/en-us/data/gg193958>

Validation

* <http://msdn.microsoft.com/en-us/data/gg193959>
* <http://blogs.msdn.com/b/adonet/archive/2011/05/27/ef-4-1-validation.aspx>

DbContext API

* <http://msdn.microsoft.com/en-us/data/gg192989>

Stored Procedure

* <http://msdn.microsoft.com/en-us/data/gg699321>