# ORM and Entity Framework

1

## Agenda

- Object Relational Mapping (ORM)
- What is Entity Framework?
- Entity Framework Architecture
- Create First Entity Data Model (EDM)
- Querying with EDM
- Data Manipulation with DBContext
- Q&A

2

## Object Relational Mapping (ORM) – Why?

- Convert wrong data type is often happen in ADO.NET that's reason ORM is invented to resolve this problem and others
- Simplified development because it automates object-to-table and table-to-object conversion, resulting in lower development and maintenance costs

3

## Object Relational Mapping (ORM) – Why?

- Less code compared to embedded SQL and handwritten stored procedures
- Transparent object caching in the application tier, improving system performance
- An optimized solution making an application faster and easier to maintain

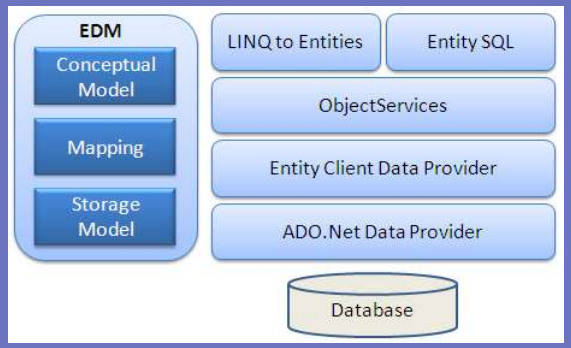
4

## What is Entity Framework (EF)?

- EF is an implementation of ORM in .Net
- EF Features
  - Provides mechanism for accessing and storing the data to database
  - Provides services as change tracking, lazy loading, query translation…etc

5

## Entity Framework Architecture



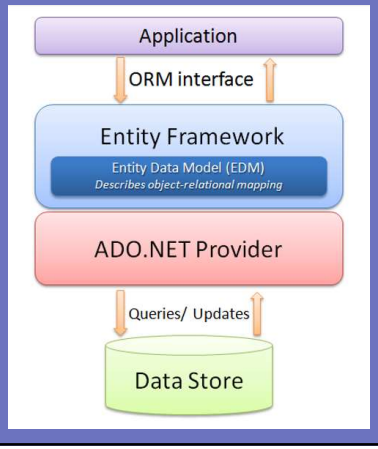

7

## What is Entity Framework (EF)?

- Using LINQ to Entities for CRUD operations (Create, Read, Update, Delete)
- Improves maintainable and extendable for application
  - Separate database design and domain class design
  - Multiple database support such as SQL server, Oracle…etc

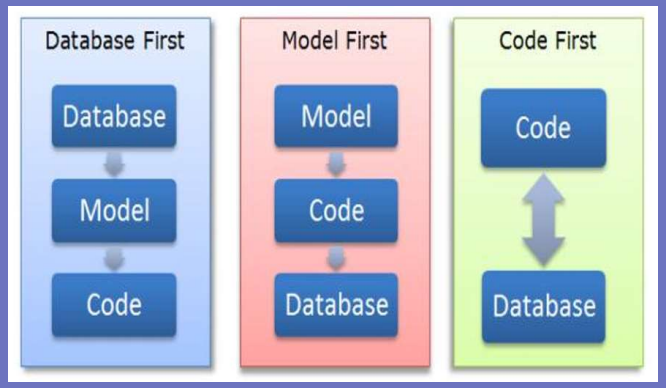
6

## Entity Framework Architecture



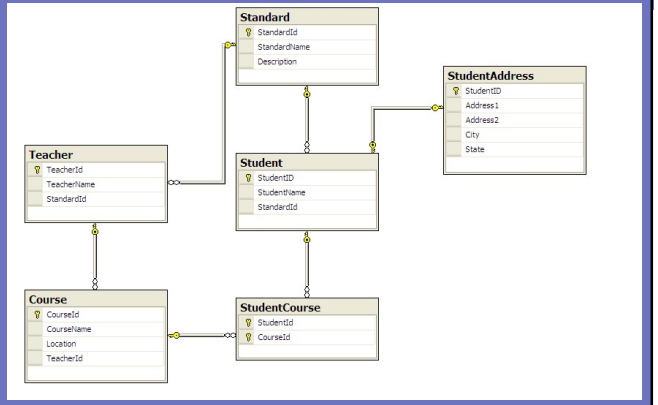

8

## Entity Framework Programming Model




9

## Create Database Schema




11

## Database First

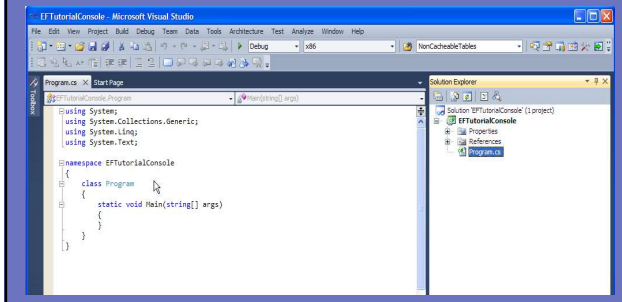
### Executed Steps:

1. Create Database Schema
2. Create New Project and EDM
3. EDM Configurations
4. EDM Generation

10

## Start New Project and EDM

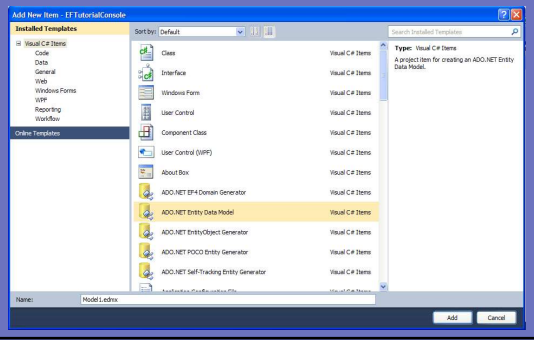
Create console project with named EFTutorialProject

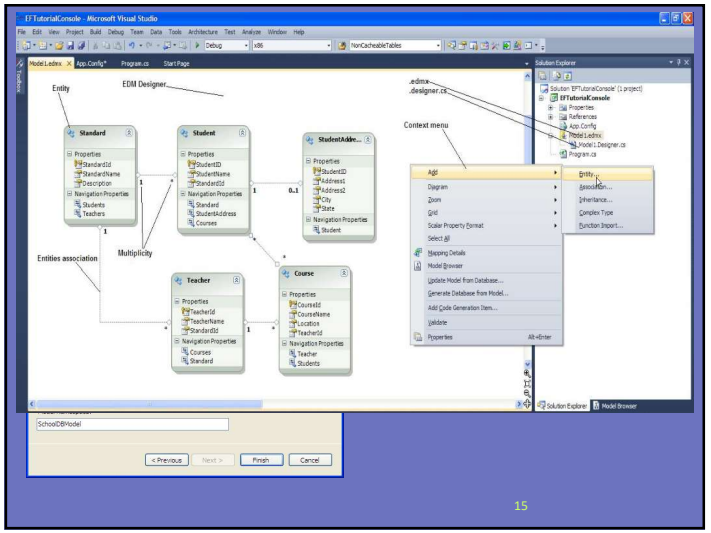



12

# Start New Project and EDM

Right click on 'EFTutorialProject' and go to Add -> New Item and select ADO.NET Entity Data Model from the popup and click "Add"

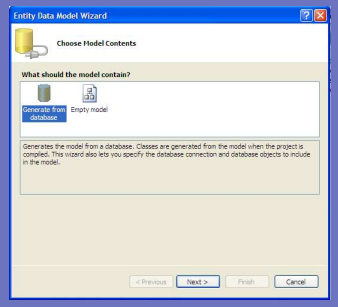


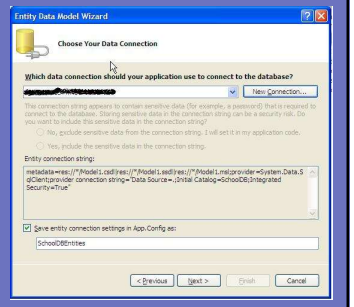

15

# EDM Configurations

It opens Entity Data Model Wizard. Select 'Generate from database' option and click 'Next'




Create new connection by clicking 'New Connection' button




14

# Querying with EDM

1. Linq to Entities
2. Entity SQL
3. Native SQL

16

## Querying with EDM LINQ to Entities

```
//Querying with LINQ to Entities
 using (var objCtx = new SchoolDBEntities())
 {
     var schoolCourse = from cs in objCtx.Courses
                        where cs.CourseName == "Course1"
                        select cs;
     Course mathCourse = schoolCourse.FirstOrDefault<Course>();
     IList<Course> courseList = schoolCourse.ToList<Course>();

     string courseName = mathCourse.CourseName;
 }
```

- Output:
  - mathCourse variable is an instance of Course object
  - courseList variable is an instance of list of Course object

17

## Querying with EDM Entity SQL

```
//Querying with Object Services and Entity SQL
using (var objCtx = new SchoolDBEntities())
{
    string sqlString = "SELECT VALUE cs FROM SchoolDBEntities.Courses
                        AS cs WHERE cs.CourseName == 'Maths'";
    ObjectQuery<Course> course = objCtx.CreateQuery<Course>(sqlString);
    Course coursename1 = course.FirstOrDefault<Course>();
}
```

- Output:
  - coursename1 variable is an instance of Course object

18

## Querying with EDM Native SQL

```
//Querying with native sql
    using (var objCtx = new SchoolDBEntities())
        {
            //Inserting Student using ExecuteStoreCommand
            int InsertedRows = objCtx.ExecuteStoreCommand("Insert into
Student(StudentName,StandardId) values('StudentName1',1)");

            //Fetching student using ExecuteStoreQuery
            var student = objCtx.ExecuteStoreQuery<Student>("Select *
from Student where StudentName = 'StudentName1'", null).ToList();

        }
```

- Output:
  - student variable is an instance of list of Student object

19

## Querying with EDM Notes

- Entity SQL and Native SQL are not recommend to use, because it will hard for maintaining or database changing in the future

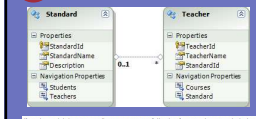
20

# Data Manipulation with Linq to Entity



## Add One-to-Many Relationship Entity Graph using DBContext

1 Entity Data Model




2

```
//Create new standard
var standard = new Standard();
standard.StandardName = "Standard1";

//create three new teachers
var teacher1 = new Teacher();
teacher1.TeacherName = "New Teacher1";

var teacher2 = new Teacher();
teacher2.TeacherName = "New Teacher2";

var teacher3 = new Teacher();
teacher3.TeacherName = "New Teacher3";

 /add teachers for new standard
tandard.Teachers.Add(teacher1);
tandard.Teachers.Add(teacher2);
tandard.Teachers.Add(teacher3);

sing (var dbCtx = new SchoolDBEntities())

   //add standard entity into standards entitySet
   dbCtx.Standards.Add(standard);
   //Save whole entity graph to the database
   dbCtx.SaveChanges();
```

3

If you trace database query then you can see following four queries executed when you call SaveChanges method:
First it inserts query for Standard table:

```
exec sp_executesql N'insert [dbo].[Standard]([StandardName], [Description])
values (@0, null)
select [StandardId]
from [dbo].[Standard]
where @@ROWCOUNT > 0 and [StandardId] = scope_identity()',N'@0 varchar(50)',@0='Standard1'
```

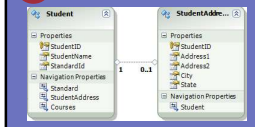
Three insert query for Teacher table:

```
exec sp_executesql N'insert [dbo].[Teacher]([TeacherName], [StandardId])
values (@0, @1)
select [TeacherId]
from [dbo].[Teacher]
where @@ROWCOUNT > 0 and [TeacherId] = scope_identity()',N'@0 varchar(50),
@1 int',@0='New Teacher1',@1=384

exec sp_executesql N'insert [dbo].[Teacher]([TeacherName], [StandardId])
values (@0, @1)
select [TeacherId]
from [dbo].[Teacher]
where @@ROWCOUNT > 0 and [TeacherId] = scope_identity()',N'@0 varchar(50),
@1 int',@0='New Teacher2',@1=384

exec sp_executesql N'insert [dbo].[Teacher]([TeacherName], [StandardId])
values (@0, @1)
select [TeacherId]
from [dbo].[Teacher]
where @@ROWCOUNT > 0 and [TeacherId] = scope_identity()',N'
@0 varchar(50),@1 int',@0='New Teacher3',@1=384
```



## Add One-to-One Relationship Entity Graph using DBContext

1 Entity Data Model




2 Code

```
//Create new student entity object
var student = new Student();

// Assign student name
student.StudentName = "New Student1";

// Create new StudentAddress entity and assign it to student entity
student.StudentAddress = new StudentAddress() { Address1 = "Student1's Address1",
        Address2 = "Student1's Address2", City = "Student1's City",
        State = "Student1's State" };

//create DBContext object
using (var dbCtx = new SchoolDBEntities())
{
    //Add student object into Student's EntitySet
    dbCtx.Students.Add(student);
    // call SaveChanges method to save student & StudentAddress into database
    dbCtx.SaveChanges();
}
```

3

If you trace database query then you can see following two queries executed when you call SaveChanges method:
First it inserts into Student table:

```
exec sp_executesql N'insert [dbo].[Student]([StudentName], [StandardId])
values (@0, @1)
select [StudentID]
from [dbo].[Student]
where @@ROWCOUNT > 0 and [StudentID] = scope_identity()',N'@0 varchar(50),@1 int',
@0='New Student1',@1=0
```

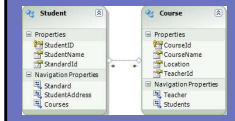
Second query inserts into StudentAddress table:

```
exec sp_executesql N'insert [dbo].[StudentAddress]([StudentID], [Address1],
[Address2], [City], [State])values (@0, @1, @2, @3, @4)'
,N'@0 int,@1 varchar(50),@2 varchar(50),@3 varchar(50),@4 varchar(50)',
@0=271,@1='Student1''s Address1',
@2='Student1''s Address2',@3='Student1''s City',
@4='Student1''s State'
```

## Add Many-to-Many Relationship Entity Graph using DBContext

1 Database Schema


Entity Data Model




2

```
//Create student entity
var student1 = new Student();
student1.StudentName = "New Student2";

//Create course entities
var course1 = new Course();
course1.CourseName = "New Course1";
course1.Location = "City1";

var course2 = new Course();
course2.CourseName = "New Course2";
course2.Location = "City2";

var course3 = new Course();
course3.CourseName = "New Course3";
course3.Location = "City1";

// add multiple courses for student entity
student1.Courses.Add(course1);
student1.Courses.Add(course2);
student1.Courses.Add(course3);

using (var dbCtx = new SchoolDBEntities())
{
    //add student into DBContext
    dbCtx.Students.Add(student1);
    //call SaveChanges
    dbCtx.SaveChanges();
}
```

3

```
exec sp_executesql N'insert [dbo].[StudentCourse]([StudentId], [CourseId])
values (@0, @1)
',N'@0 int,@1 int',@0=274,@1=249

exec sp_executesql N'insert [dbo].[StudentCourse]([StudentId], [CourseId])
values (@0, @1)
',N'@0 int,@1 int',@0=274,@1=250

exec sp_executesql N'insert [dbo].[StudentCourse]([StudentId], [CourseId])
values (@0, @1)
',N'@0 int,@1 int',@0=274,@1=251
```

## Update Entity using DBContext

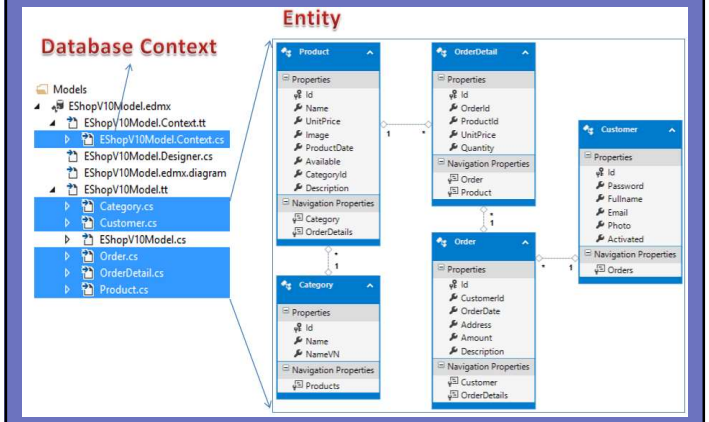
```
Student stud ;
// Get student from DB
using (var ctx = new SchoolDBEntities())
{
    stud = ctx.Students.Where(s => s.StudentName == "New Student1").FirstOrDefault<Student>();
}

// change student name in disconnected mode (out of DBContext scope)
if (stud != null)
{
    stud.StudentName = "Updated Student1";
}

//save modified entity using new DBContext
using (var dbCtx = new SchoolDBEntities())
{
    //Mark entity as modified
    dbCtx.Entry(stud).State = System.Data.EntityState.Modified;
    dbCtx.SaveChanges();
}
```

SaveChanges will send following update query to the database:

```
exec sp_executesql N'update [dbo].[Student]
set [StudentName] = @0, [StandardId] = @1
where ([StudentID] = @2)',N'@0 varchar(50),
@1 int,@2 int',@0='Updated Student1',@1=299,@2=267
```

25

## Example




27

## Delete Entity using DBContext

```
using (var dbCtx = new SchoolDBEntities())
    {

        //if already loaded in existing DBContext then use Set().Remove(entity) to delete it.
        var newtchr = dbCtx.Teachers.Where(t => t.TeacherName == "New teacher4")
                                .FirstOrDefault<Teacher>();
        dbCtx.Set(Teacher).Remove(newtchr);

        //Also, you can mark an entity as deleted
        //dbCtx.Entry(tchr).State = System.Data.EntityState.Deleted;

        //if not loaded in existing DBContext then use following.
        //dbCtx.Teachers.Remove(newtchr);

        dbCtx.SaveChanges();
    }
```

Above code results in following delete query which deletes the row from Teacher table.

```
delete [dbo].[Teacher]
where ([TeacherId] = @0)',N'@0 int',@0=1
```

26

## Example

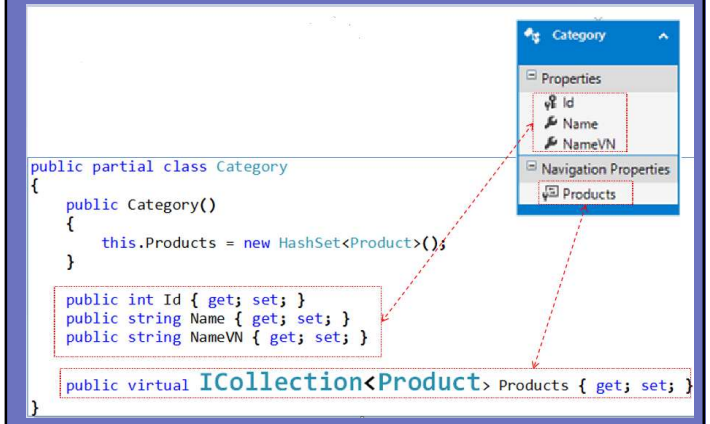
```
public partial class EShopV10 : DbContext
{
    public EShopV10()
        : base("name=EShopV10")
    {
    }

    public DbSet<Category> Categories { get; set; }
    public DbSet<Customer> Customers { get; set; }
    public DbSet<OrderDetail> OrderDetails { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<Product> Products { get; set; }
}
```

```
<connectionStrings>
    <add name="EShopV10" connectionString="…"
         providerName="System.Data.EntityClient" />
</connectionStrings>
```

Web.config

28

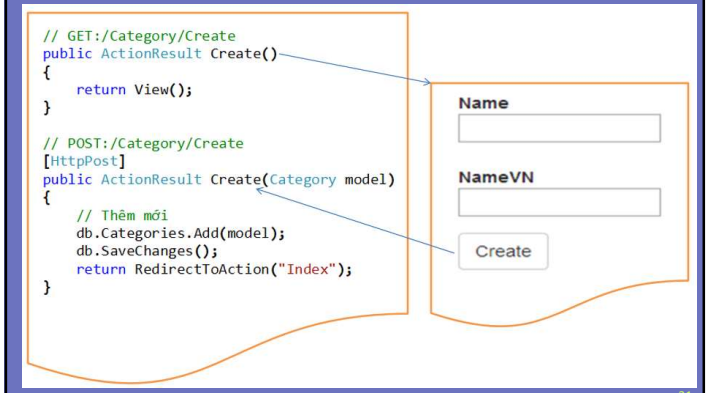
## Example




29

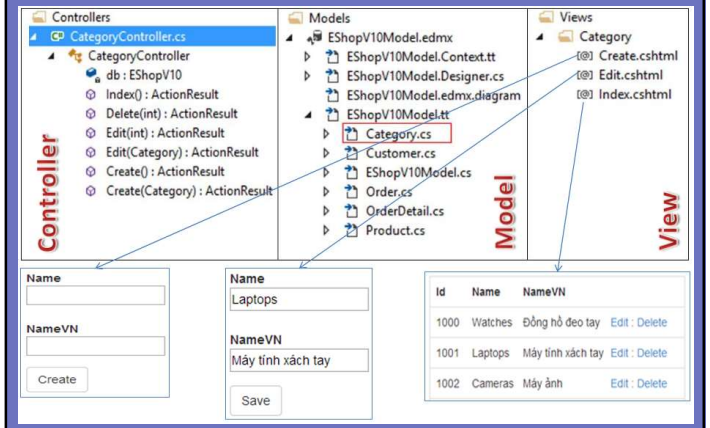
## Example

- Create:




31

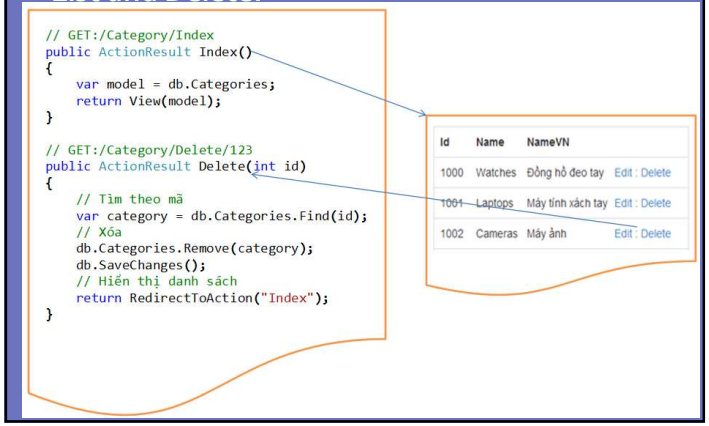
## Example




30

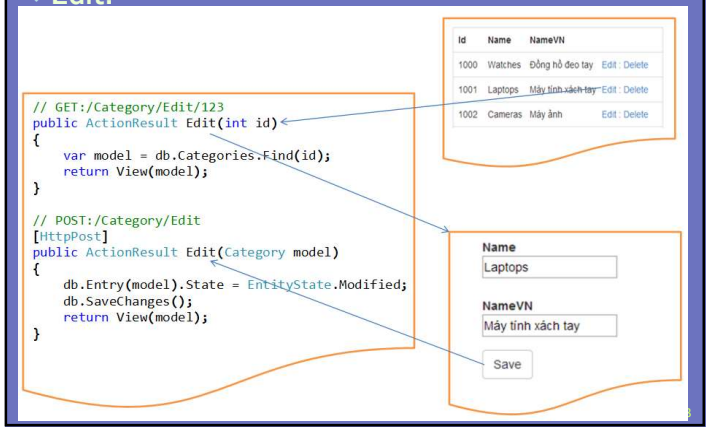
## Example

- List and Delete:



## Example

- Edit:




## Code First

# Code First

35

# End of Database First

34

## Code First

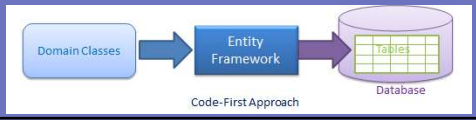
## Agenda

1. What is Code-First
2. Database Initialization
3. Simple Code-First Example
4. Configure Domain Classes in Code-First
5. DataAnnotation
6. Configure One-to-One
7. Configure One-to-Many
8. Configure Many-to-Many

36

## What is Code-First

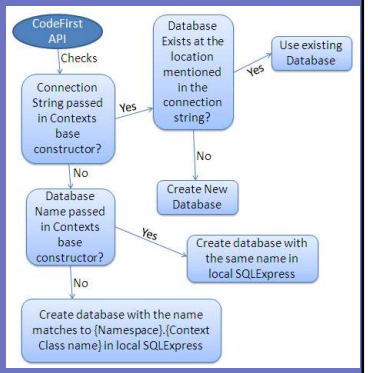
- Code First was introduced in Entity Framework 4.1 and mainly useful in Domain Driven Design
- Focus on the domain design and start creating classes as per your domain requirement rather than design your database First approach
- Code First APIs will create the database on the fly based on your entity classes and configuration




37

## Database Initialization

- Let's how database initialize in code-first application.
- Following figure shows database initialization workflow based on the parameter passed in base constructor of context class which is derived from DbContext




39

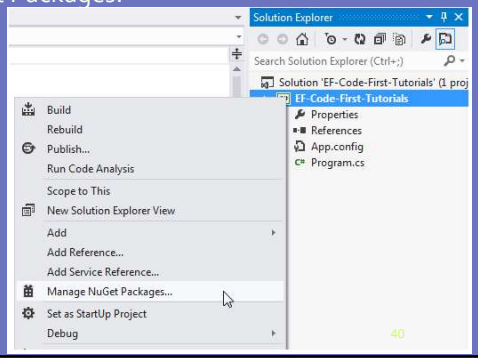
## What is Code-First

- So the basic workflow would be:
  - Write code-first application classes → Hit F5 to run the application → Code First API creates new database or map with existing database from application classes → Inserts default/test data into the database → Finally launch the application

38

## Install EF via Nuget

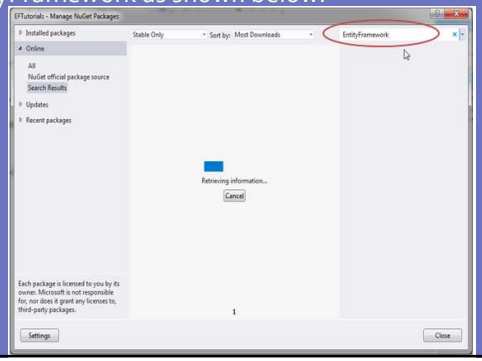
- First, create the console application. Right click on your project in the solution explorer and select Manage NuGet Packages.




40

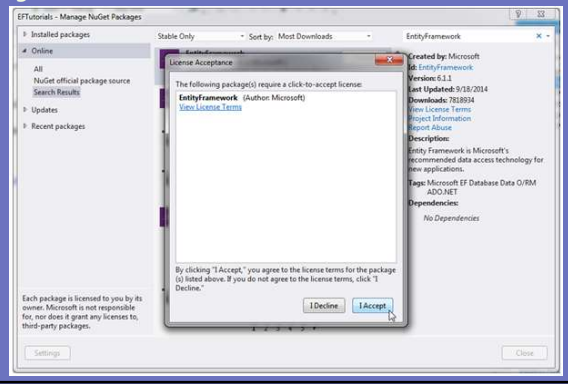
## Install EF via Nuget

- This will open **Manage NuGet Packages** dialogue box. Now, select Online in the left bar and search for EntityFramework as shown below.



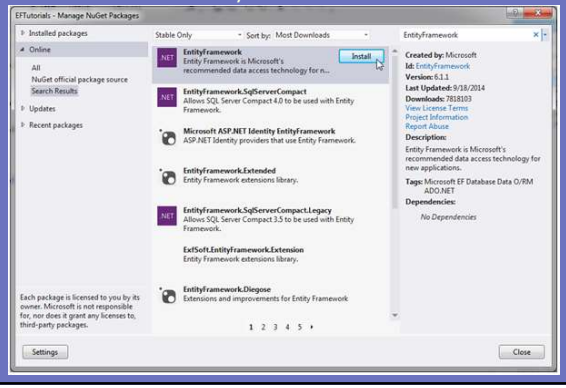

## Install EF via Nuget

- Click on the **I Accept** button in the License Acceptance dialogue box to start the installation.




## Install EF via Nuget

- This will search for all the packages related to Entity Framework. Select EntityFramework and click on Install.




## Simple Code-First Example

```
public class Student
{
    public Student()
    {

    }
    public int StudentID { get; set; }
    public string StudentName { get; set; }
}
```

1

```
namespace SchoolDataLayer
{
    public class Context: DbContext
        {
            public Context(): base()
            {

            }
            public DbSet<Student> Students { get; set; }
            public DbSet<Standard> Standards { get; set; }
        }
}
```

2

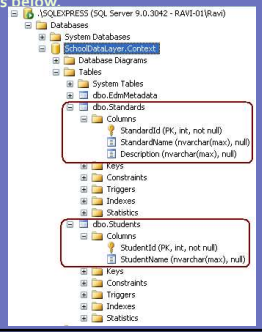
```
public class Standard
{
    public Standard()
    {

    }
    public int StandardId { get; set; }
    public string StandardName { get; set; }
    public string Description { get; set; }
}
```

```
using (var ctx = new Context())
{
    Student stud = new Student() { StudentName = "New Student" };
    ctx.Students.Add(stud);
    ctx.SaveChanges();
}
```

3

# Simple Code-First Example

- Code-first API will also create two tables in the database, Students and Standards table based on Student and Standard class. Code First APIs creates PrimaryKey in the table if class has either "Id" or ClassName + "Id" property. For example, Student class has "StudentId" property so it will create StudentId as PK column. It will also create other columns with the same name and datatype as property names and datatype as below.




45

# Configure Domain Classes in Code-First

Following is an example of DataAnnotation used in Student Class:

```
[Table("StudentInfo")]
public class Student
{
    public Student() { }

    [Key]
    public int SID { get; set; }

    [Column("Name", TypeName="ntext")]
    [MaxLength(20)]
    public string StudentName { get; set; }

    [NotMapped]
    public int? Age { get; set; }


    public int StdId { get; set; }

    [ForeignKey("StdId")]
    public virtual Standard Standard { get; set; }
}
```

46

# DataAnnotation in Code-First Validation Attributes

| Annotation Attribute | Description |
|---|---|
| Required | The Required annotation will force EF (and MVC) to ensure that property has data in it. |
| MinLength | MinLength annotation validates property whether it has minimum length of array or string. |
| MaxLength | MaxLength annotation maximum length of property which in-tern set maximum length of column in the database |
| StringLength | Specifies the minimum and maximum length of characters that are allowed in a data field. |

47

# DataAnnotation in Code-First Database Schema related Attributes

| Annotation Attribute | Description |
|---|---|
| Table | Specify name of the DB table which will be mapped with the class |
| Column | Specify column name and datatype which will be mapped with the property |
| Key | Mark property as EntityKey which will be mapped to PK of related table. |
| ComplexType | Mark the class as complex type in EF. |
| Timestamp | Mark the property as a non-nullable timestamp column in the database. |
| ForeignKey | Specify Foreign key property for Navigation property |
| NotMapped | Specify that property will not be mapped with database |
| ConcurrencyCheck | ConcurrencyCheck annotation allows you to flag one or more properties to be used for concurrency checking in the database when a user edits or deletes an entity. |
| DatabaseGenerated | DatabaseGenerated attribute specifies that property will be mapped to Computed column of the database table. So the property will be read-only property. It can also be used to map the property to identity column (auto incremental column). |
| InverseProperty | InverseProperty is useful when you have multiple relationship between two classes. |

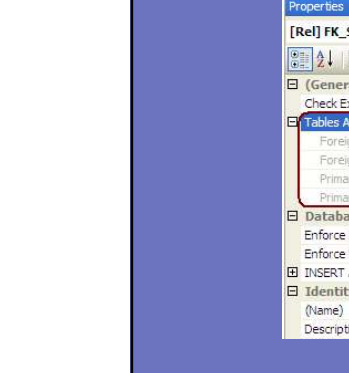
48

# Configure One-to-One Relationship

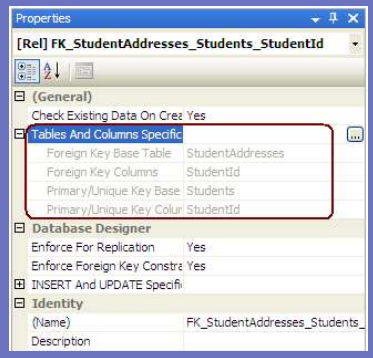
1

```
Configure one to zero or one relationship using DataAnnotation:
public class Student
{
    public Student() { }

    public int StudentId { get; set; }
    [Required]
    public string StudentName { get; set; }

    [Required]
    public virtual StudentAddress StudentAddress { get; set; }

}
public class StudentAddress
{
    [Key, ForeignKey("Student")]
    public int StudentId { get; set; }

    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public int Zipcode { get; set; }
    public string State { get; set; }
    public string Country { get; set; }

    public virtual Student Student { get; set; }
}
```

2




49



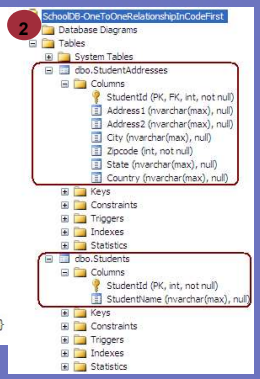

51

# Configure One-to-Many Relationship

1

```
public class Student
{
    public Student() { }

    public int StudentId { get; set; }
    [Required]
    public string StudentName { get; set; }

    public int StdandardId { get; set; }

    public virtual Standard Standard { get; set; }
}

public class Standard
{
    public Standard()
    {
        Students = new List<Student>();
    }
    public int StandardId { get; set; }
    public string StandardName { get; set; }
    public string Description { get; set; }

    public virtual ICollection<Student> Students { get; set; }
}
```

2



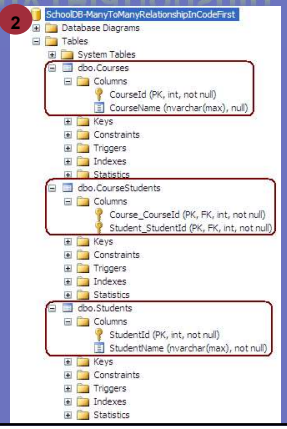

50

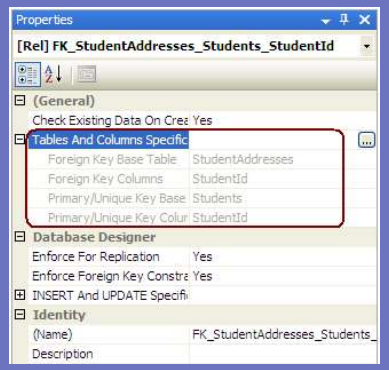
# Configure Many-to-Many relationship

1

```
public class Student
{
    public Student() { }

    public int StudentId { get; set; }
    [Required]
    public string StudentName { get; set; }

    public int StdandardId { get; set; }

    public virtual ICollection<Course> Courses { get; set; }
}
public class Course
{
    public Course()
    {
        this.Students = new HashSet<Student>();
    }

    public int CourseId { get; set; }
    public string CourseName { get; set; }

    public virtual ICollection<Student> Students { get; set; }
}
```

2








## Question & Answer



## References

- Basic Entity Framework
  - http://www.entityframeworktutorial.net/EntityFramework4.3/Introduction.aspx
- Code First Tutorial
  - http://www.entityframeworktutorial.net/code-first/entity-framework-code-first.aspx
- Ebooks
  - http://1drv.ms/1goxfRZ