Entity Framework

Data Manipulation

Lesson Objectives

- Querying in Entity Framework
- Raw SQL Queries
- Stored Procedure in EF
- Transaction in EF
- Logging Database Commands

Section 1

QUERYING IN ENTITY FRAMEWORK

LINQ-to-Entities Query

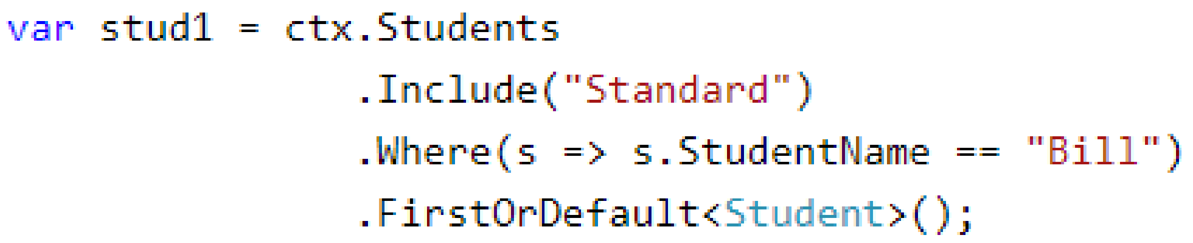
- Use LINQ for querying against DbSet. It will be converted to an SQL query.
- EF API executes this SQL query
  - ✓ to the underlying database,
  - ✓ gets the flat result set,
  - ✓ converts it into appropriate entity objects and
  - ✓ returns it as a query result

Eager Loading

- Is the process whereby a query for one type of entity also loads related entities as part of the query,
- Don't need to execute a separate query for related entities.
- Eager loading is achieved using the **Include()** method.

Eager Loading

- Example: Gets all the students from the database along with its standards using the Include() method.



```
var stud1 = ctx.Students
                    .Include("Standard")
                    .Where(s => s.StudentName == "Bill")
                    .FirstOrDefault<Student>();
```

Lazy Loading

- Is delaying the loading of related data, until specifically request for it.
- Navigation property should be defined as public, virtual.
- Context will **NOT** do lazy loading if the property is not defined as virtual.

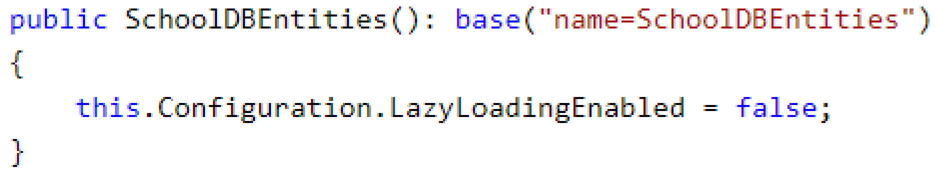
Lazy Loading

- The context first loads the Student entity data from the database,
- Then it will load the StudentAddress entity when we access the StudentAddress

```
//Loading students only
IList<Student> studList = ctx.Students.ToList<Student>();

Student std = studList[0];

//Loads Student address for particular Student only (seperate SQL query)
StudentAddress add = std.StudentAddress;
```

Disable Lazy Loading

- We can disable lazy loading for a particular entity or a context.
  - ✓ To turn off lazy loading for a particular property, do not make it virtual.
  - ✓ To turn off lazy loading for all entities in the context, set its configuration property to false.



```
public SchoolDBEntities(): base("name=SchoolDBEntities")
{
    this.Configuration.LazyLoadingEnabled = false;
}
```

Explicit Loading

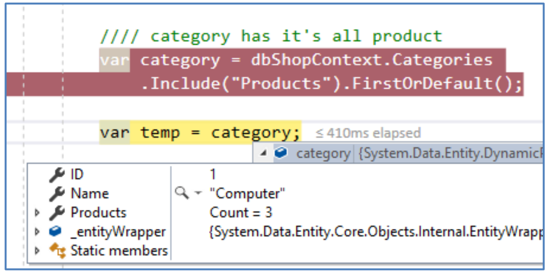
- **To** load the entities when lazy loading is disabled
- **By** calling the Load method for the related entities.
  - ✓ Reference: to load single navigation property
  - ✓ Collection: to load collections

```
var category = dbShopContext.Categories.Find(1);
dbShopContext.Entry(category).Collection(p => p.Products).Load();
```

Comparison

Eager Loading

Always load for first time

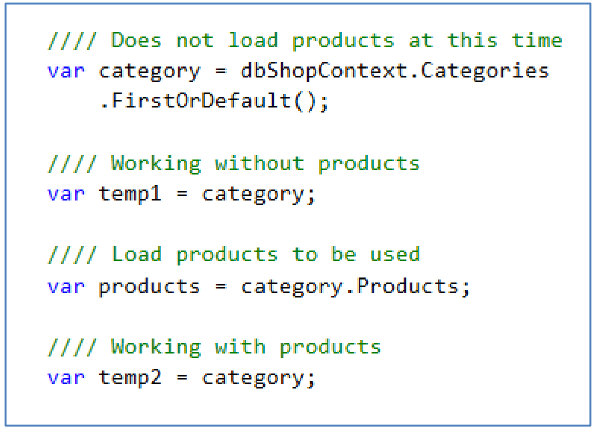


```
//// category has it's all product
var category = dbShopContext.Categories
    .Include("Products").FirstOrDefault();

var temp = category;  ≤ 410ms elapsed
```

```
category {System.Data.Entity.DynamicP
  ID                 1
  Name               "Computer"
  Products           Count = 3
  _entityWrapper     {System.Data.Entity.Core.Objects.Internal.EntityWrapp
  Static members
```

Lazy Loading

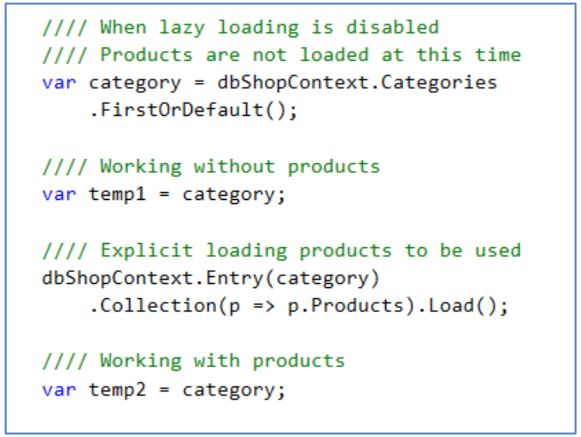
Load for first time requested



```
//// Does not load products at this time
var category = dbShopContext.Categories
    .FirstOrDefault();

//// Working without products
var temp1 = category;

//// Load products to be used
var products = category.Products;

//// Working with products
var temp2 = category;
```

Explicit Loading

Load when explicit call



```
//// When lazy loading is disabled
//// Products are not loaded at this time
var category = dbShopContext.Categories
    .FirstOrDefault();

//// Working without products
var temp1 = category;

//// Explicit loading products to be used
dbShopContext.Entry(category)
    .Collection(p => p.Products).Load();

//// Working with products
var temp2 = category;
```

When to use what

- Use Eager Loading when the relations are not too much. Thus, Eager Loading is a good practice to reduce further queries on the Server.
- Use Eager Loading when you are sure that you will be using related entities with the main entity everywhere.
- Use Lazy Loading when you are using one-to-many collections.

When to use what

- Use Lazy Loading when you are sure that you are not using related entities instantly.
- When you have turned off Lazy Loading, use Explicit loading when you are not sure whether or not you will be using an entity beforehand.

Section 2

RAW SQL QUERIES

Raw SQL Queries

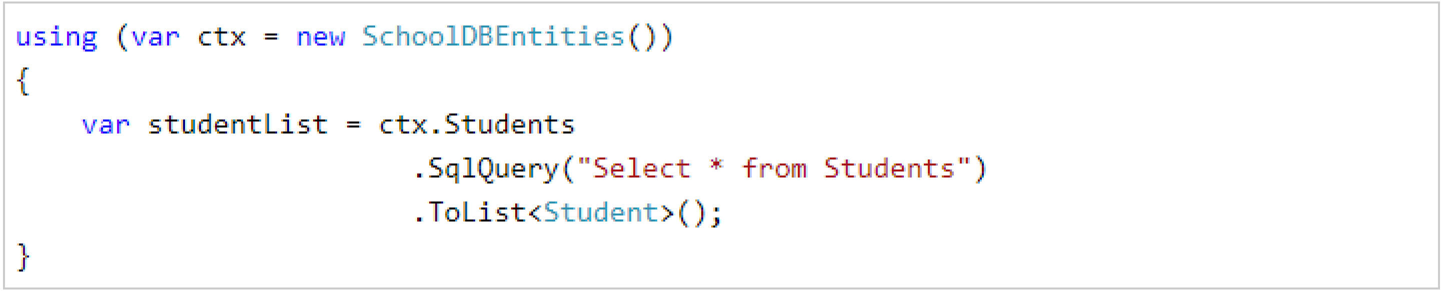
- Entity Framework allows to execute raw SQL queries for the underlying relational database.
  - ✓ DbSet.SqlQuery()
  - ✓ DbContext.Database.SqlQuery()
  - ✓ DbContext.Database.ExecuteSqlCommand()

Raw SQL Queries

- **Notes:**
  - ✓ Make sure you understand your SQL query
  - ✓ Avoid SQL mistake and SQL Injection
  - ✓ Try to convert to EF query if possible

DbSet.SqlQuery()

- Use the DbSet.SqlQuery() method to write raw SQL queries which return entity instances.
- The resulted entities will be tracked by the context, as if they were returned by the LINQ query.



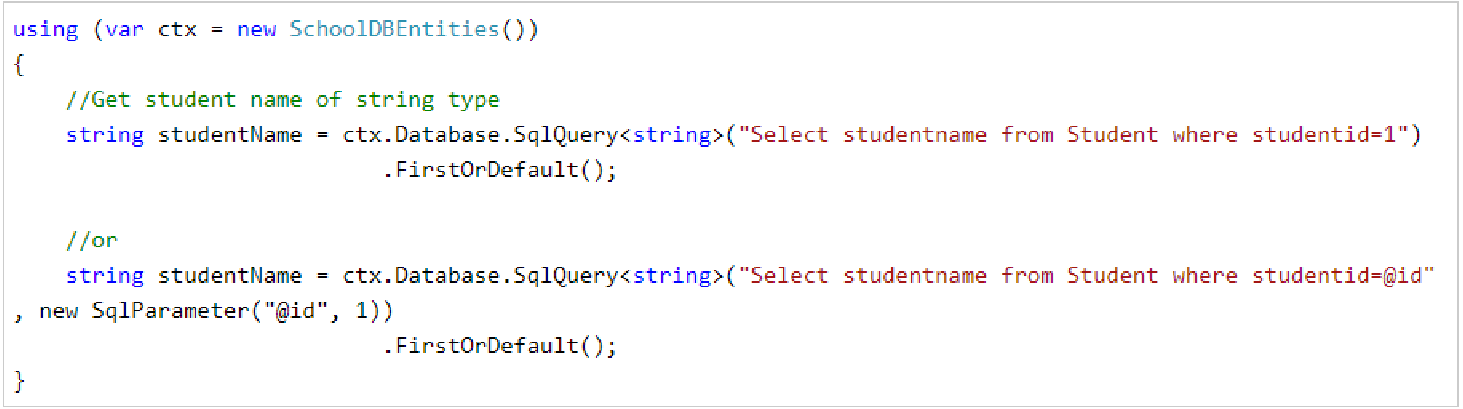
```
using (var ctx = new SchoolDBEntities())
{
    var studentList = ctx.Students
                        .SqlQuery("Select * from Students")
                        .ToList<Student>();
}
```

DbSet.SqlQuery()

- The column names in the SQL query must match with the properties of an entity type
- Specify the parameters using the object of SqlParameter
- The SQL query only for the table which is mapped with the specified entity

Database.SqlQuery()

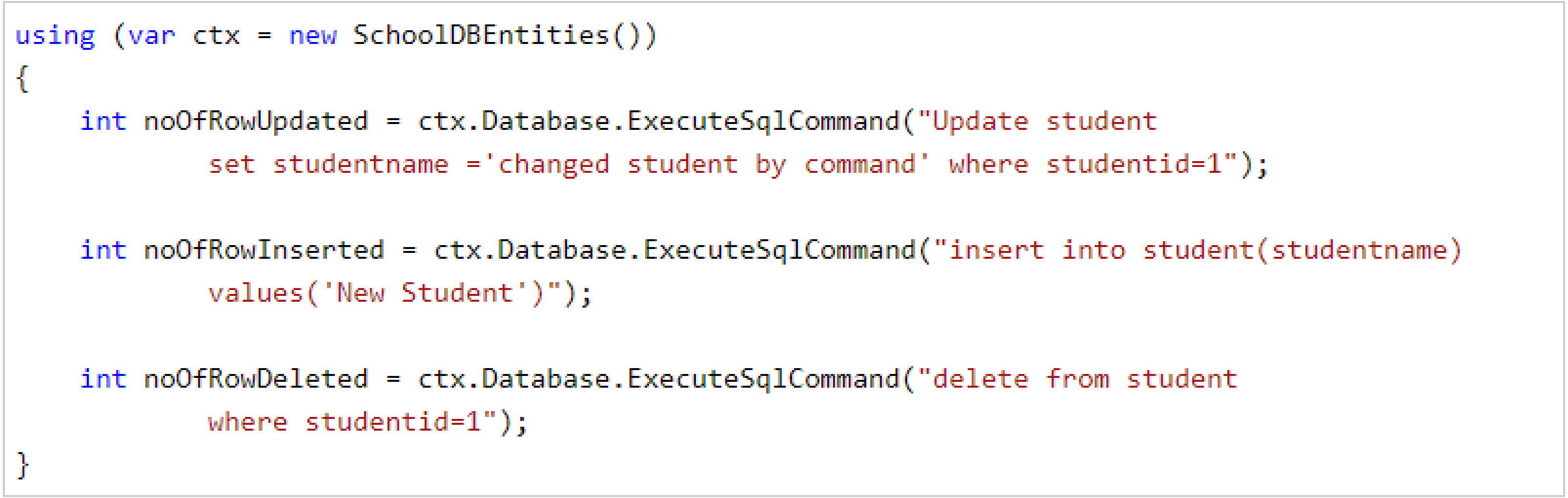
- Database.SqlQuery()
  - ✓ The Database class represents the underlying database and provides various methods to deal with the database.
  - ✓ The Database.SqlQuery() method returns a value of any type.



```
using (var ctx = new SchoolDBEntities())
{
    //Get student name of string type
    string studentName = ctx.Database.SqlQuery<string>("Select studentname from Student where studentid=1")
                            .FirstOrDefault();

    //or
    string studentName = ctx.Database.SqlQuery<string>("Select studentname from Student where studentid=@id"
, new SqlParameter("@id", 1))
                            .FirstOrDefault();
}
```

Database.ExecuteSqlCommand()

- The Database.ExecuteSqlCommnad() method is useful in executing database commands, such as the Insert, Update and Delete command.



```
using (var ctx = new SchoolDBEntities())
{
    int noOfRowUpdated = ctx.Database.ExecuteSqlCommand("Update student
            set studentname ='changed student by command' where studentid=1");

    int noOfRowInserted = ctx.Database.ExecuteSqlCommand("insert into student(studentname)
            values('New Student')");

    int noOfRowDeleted = ctx.Database.ExecuteSqlCommand("delete from student
            where studentid=1");
}
```

Stored Procedure in EF

- Stored Procedure is used limited in EF
- We use stored procedures for CUD (create, update, delete) operations for an entity when we call the SaveChanges() method in the database-first approach.

Section 3

TRANSACTION IN ENTITY FRAMEWORK

Transaction in Entity Framework

- In Entity Framework, the SaveChanges() method internally creates a transaction and wraps all INSERT, UPDATE and DELETE operations under it.
- Multiple SaveChanges() calls, create separate transactions, perform CRUD operations and then commit each transaction
- Any action in the transaction is false, the transaction should be roll-back

Multiple SaveChanges

- EF 6 and EF Core allow to create or use a single transaction with multiple **SaveChanges()** calls using the following methods:
  - ✓ DbContext.Database.BeginTransaction()
    - Creates a new transaction for the underlying database and allows us to commit or roll back changes made to the database using multiple SaveChanges method calls.
  - ✓ DbContext.Database.UseTransaction()
    - Allows us to pass an existing transaction object created out of the scope of a context object. This will allow EF to execute commands within an external transaction object. Alternatively, pass in null to clear the framework's knowledge of that transaction.

DbContext.Database.BeginTransaction

```
using (DbContextTransaction transaction = context.Database.BeginTransaction())
{
    try
    {
        var standard = context.Standards.Add(new Standard() { StandardName = "1st Grade" });

        context.Students.Add(new Student()
        {
            FirstName = "Rama2",
            StandardId = standard.StandardId
        });
        context.SaveChanges();

        context.Courses.Add(new Course() { CourseName = "Computer Science" });
        context.SaveChanges();

        transaction.Commit();
    }
    catch (Exception ex)
    {
        transaction.Rollback();
        Console.WriteLine("Error occurred.");
    }
}
```

DbContext.Database.UseTransaction

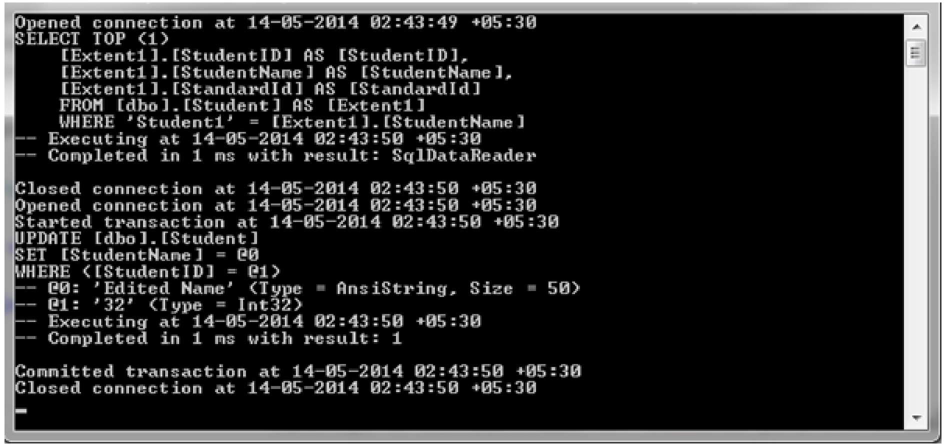
- The DbContext.Database.UseTransaction() method allows us to use an existing transaction created out of the scope of the context object.
- If we use the UseTransaction() method, then the context will not create an internal transaction object and will use the supplied transaction

Logging Database Commands

- EF 6 provides the DbContext.Database.Log property to log the SQL generated by DbContext.
- The Log property is of Action<string> type, so you can attach a delegate method with the string parameter and return void.

Logging Database Commands

```
using (var context = new SchoolDBEntities())
{
    context.Database.Log = Console.Write;
    var student = context.Students
                        .Where(s => s.StudentName == "Student1")
                        .FirstOrDefault<Student>();

    student.StudentName = "Edited Name";
    context.SaveChanges();
}
```



```
Opened connection at 14-05-2014 02:43:49 +05:30
SELECT TOP (1)
    [Extent1].[StudentID] AS [StudentID],
    [Extent1].[StudentName] AS [StudentName],
    [Extent1].[StandardId] AS [StandardId]
    FROM [dbo].[Student] AS [Extent1]
    WHERE 'Student1' = [Extent1].[StudentName]
-- Executing at 14-05-2014 02:43:50 +05:30
-- Completed in 1 ms with result: SqlDataReader

Closed connection at 14-05-2014 02:43:50 +05:30
Opened connection at 14-05-2014 02:43:50 +05:30
Started transaction at 14-05-2014 02:43:50 +05:30
UPDATE [dbo].[Student]
SET [StudentName] = @0
WHERE ([StudentID] = @1)
-- @0: 'Edited Name' (Type = AnsiString, Size = 50)
-- @1: '32' (Type = Int32)
-- Executing at 14-05-2014 02:43:50 +05:30
-- Completed in 1 ms with result: 1

Committed transaction at 14-05-2014 02:43:50 +05:30
Closed connection at 14-05-2014 02:43:50 +05:30
_
```

Summary

- DbSet is used to manipulate data
- Entity Framework uses LINQ for querying data
- 3 types: Eager Loading vs Lazy Loading vs Explicit Loading

Thank you