Entity Framework (EF)

TABLE OF CONTENTS

WHAT IS A ENTITY FRAMEWORK?

is an Object-Relational Mapping (ORM) framework developed by Microsoft. It simplifies the interaction between .NET applications and databases by allowing developers to work with databases using .NET objects.

CONCEPTS OF ENTITY FRAMEWORK (EF)

Db Context

Db Context is a key class in Entity Framework. It represents a session with the database and is responsible for querying, saving, and managing entities.

SYNTAX FOR CREATING DBCONTEXT

csharp

Copy code

```csharp
using System.Data.Entity;

public class ApplicationDbContext : DbContext
{
    public DbSet<Product> Products { get; set; }
}
```

Entities

Entities are the .NET objects that represent the data in the database. Each entity typically corresponds to a row in a database table.

SYNTAX FOR CREATING ENTITY

csharp Copy code

```csharp
using System.ComponentModel.DataAnnotations.Schema;

[Table("YourTableName")]
public class YourEntity
{
    public int Id { get; set; }
    public string Name { get; set; }

    // Add other properties as needed
}
```

LINQ (Language Integrated Query)

Entity Framework allows developers to use LINQ queries to interact with the database. LINQ provides a strongly-typed query syntax that is integrated into C# and VB.NET.

LINQ SYNTAX

```
using System;
using System.Linq;
using System.Collections.Generic;

class Program
{
    static void Main(string[] args)
    {
        // Sample data
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

        // LINQ query syntax
        var query =
            from number in numbers
            where number % 2 == 0
            select number;

        // Execute the query and print the results
        foreach (var number in query)
        {
            Console.WriteLine(number);
        }
    }
}
```

Automatic Change Tracking

Entity Framework automatically tracks changes made to entities, and it can generate the appropriate SQL statements to update the database accordingly.

Automatic Change Tracking Syntax

```
using System.Linq;

class Program
{
    static void Main(string[] args)
    {
        // Assume we have a DbContext named ApplicationDbContext
        using (var dbContext = new ApplicationDbContext())
        {
            // Retrieve a product from the database
            var product = dbContext.Products.FirstOrDefault(p => p.Id == 1);

            // Check if the product exists
            if (product != null)
            {
                // Modify the product properties
                product.Name = "New Product Name";
                product.Price = 19.99m;

                // Changes are automatically tracked by Entity Framework
                // The state of the product is set to Modified

                // Save changes to the database
                dbContext.SaveChanges();
            }
            else
            {
                Console.WriteLine("Product not found!");
            }
```

Code-First and Database-First Approaches

With Code-First, you define your data model in code, and Entity Framework creates the database schema based on your model. With Database-First, you start with an existing database, and Entity Framework generates the corresponding .NET classes.

CODE FIRST APPROACH SYNTAX

```csharp
using System;
using System.Data.Entity;

// Define the entity class
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}

// Define the DbContext class
public class ApplicationDbContext : DbContext
{
    public DbSet<Product> Products { get; set; }
}
```

```csharp
// Usage example
class Program
{
    static void Main(string[] args)
    {
        using (var context = new ApplicationDbContext())
        {
            var product = new Product { Name = "Example Product", Price = 10.99m }
            context.Products.Add(product);
            context.SaveChanges();
        }
    }
}
```

DATABASE FIRST APPROACH SYNTAX

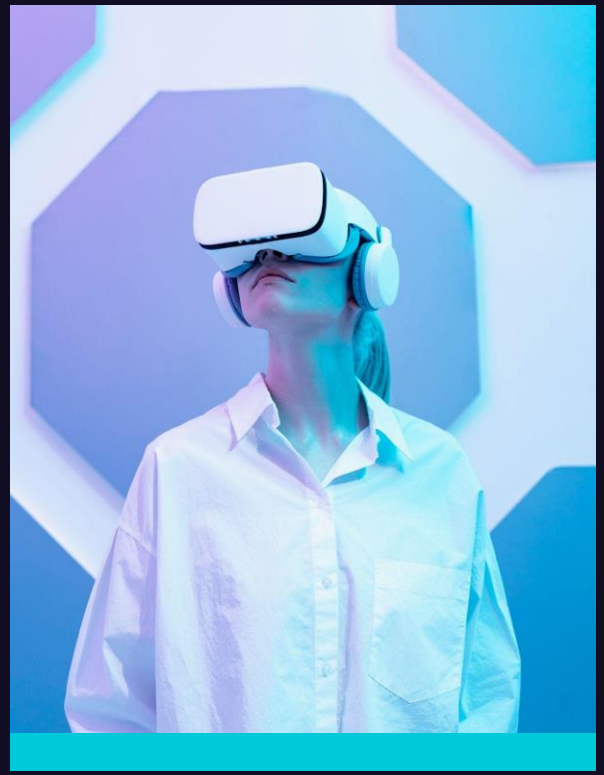
```
using System;
using System.Linq;
using YourGeneratedNamespace; // This namespace contains your generated entity clas

class Program
{
    static void Main(string[] args)
    {
        // Create an instance of your generated DbContext class
        using (var dbContext = new YourGeneratedDbContext())
        {
            // Query products from the database
            var products = dbContext.Products.ToList();

            // Display product information
            foreach (var product in products)
            {
                Console.WriteLine($"ID: {product.Id}, Name: {product.Name}, Price:
            }
        }
    }
}
```

Migrations

Entity Framework Migrations allow developers to evolve the database schema over time as the application evolves. Migrations provide a way to apply changes to the database in a structured and versioned manner.



MIGRATION SYNTAX

```
using System.Data.Entity;

// Define your entity class
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}

// Define your DbContext class
public class ApplicationDbContext : DbContext
{
    public DbSet<Product> Products { get; set; }
}

class Program
{
    static void Main(string[] args)
    {
```

```
class Program
{
    static void Main(string[] args)
    {
        // Create an instance of your DbContext
        using (var dbContext = new ApplicationDbContext())
        {
            // Enable migrations
            // Open Package Manager Console in Visual Studio and run: Enable-M

            // Add initial migration
            // Run: Add-Migration InitialCreate

            // Apply migrations
            // Run: Update-Database

            // Modify data model (e.g., add new entity or property)

            // Add new migration
            // Run: Add-Migration AddNewEntity

            // Apply new migration
            // Run: Update-Databas
        }
```