Platforma .NET – Wykład 8
Dostęp do danych w .NET, LINQ, Entity Framework

Osoba prowadząca wykład, laboratorium i projekt:
dr hab. inż. Marek Sawerwain, prof. UZ

Instytut Sterowania i Systemów Informatycznych
Uniwersytet Zielonogórski

e-mail : M.Sawerwain@issi.uz.zgora.pl
tel. (praca) : 68 328 2321,
pok. 328a A-2,
ul. Prof. Z.Szafrana 2,
65-246 Zielona Góra

Ostatnia kompilacja pliku: Monday 5th June, 2023, t: 23:09

Notatki

Spis treści

Notatki

Plan wykładu – spotkania tydzień po tygodniu

(1) Informacje o wykładzie, pojęcie platformy, podstawowe informacje o platformie .NET
(2) Składowe platformy .NET: CLR, CTS, języki programowania, biblioteki klas, pojęcie podzespołu (ang. assembly)
(3) Programowanie w C# – środowisko VS, MonoDevelop, syntaktyka C#, wyjątki, współpraca z DLL
(4) Programowanie w C# – model obiektowy, typy uogólnione, lambda wyrażenia
(5) Programowanie w C# – aplikacje „okienkowe", programowanie wielowątkowe
(6) Programowanie w F# – podstawy, przetwarzanie danych tekstowych,
(*) "Klasówka I", czyli egzamin cześć pierwsza
(7) Dostęp do baz danych

Notatki

Plan wykładu – tydzień po tygodniu

(8) Język zapytań LINQ, Entity Framework
(9) Obsługa standardu XML
(10) Technologia ASP.NET 1/2
(11) Technologia ASP.NET 2/2
(12) Model widok i kontroler – Model View Controller
(13) Tworzenie usług sieciowych SOAP i WCF (komunikacja sieciowa)
(14) Wykład monograficzny .NET 1
(15) Wykład monograficzny .NET 2
(*) "Klasówka II", czyli egzamin cześć druga

Notatki

Plan wykładu – 1/2

1. Model bezpołączeniowy i transakcje,
   - model bezpołączeniowy,
   - transakcje w ADO .NET,
   - kontrolki ekranowe,
2. Technologia LINQ,
   - język zapytań
   - zapytania do typów zgodnych z kolekcjami,
   - zapytania do baz danych,
3. Zapytania równoległe PLINQ,
   - idea zapytań równoległych
   - wydajność
4. Wprowadzenie do Entity Framework
   - odrobina historii,
   - wersje pakietów EF,
   - najważniejsze założenia.

...

Notatki

Plan wykładu – 2/2

...

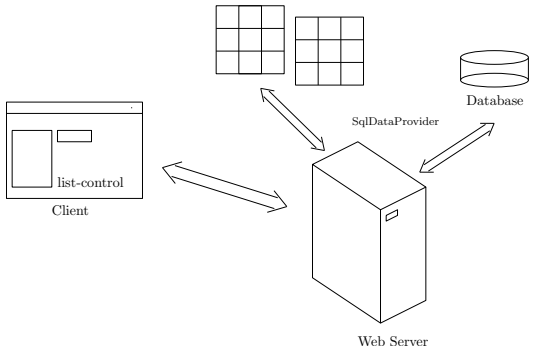
5. Architektura i model EF,
   - miejsce EF w aplikacji,
   - architektura EF,
   - rodzaje i stany encji,
   - trzy podejścia do współpracy z danymi.
6. Przykłady praktyczne
   - podejście „code-first".

Notatki

Użycie zbioru danych – DataSet

Zazwyczaj korzystanie z danych w trybie bezpołączeniowych przedstawia się następująco:

1. klient tworzy żądanie dostępu do danych,
2. utworzenie obiektów **SqlConnection** i **SqlDataAdapter**,
3. wypełnienie danymi obiektu **DataSet** przy pomocy **DataAdapter**,
4. klient otrzymuje dostęp do **DataSet**,
5. aktualnie danych w zbiorze danych (**DataSet**),
6. użycie obiektu **SqlDataAdapter** otwierającego połączenie SqlConnection, proces aktualizacji danych, zamknięcie połączenia.



Notatki

Kiedy warto stosować zbiory danych (DataSet)

Ogólne zasady stosowanie zbioru danych – **DataSet**:

1. jeśli dane są edytowane, czyli do bazy danych należy dodawać oraz usuwać rekordy,
2. jeśli wymagana jest dodatkowa organizacja danych jak np.: filtrowanie, sortowanie, czy wyszukiwanie,
3. jeśli rekordy pobrane z bazy danych będą przetwarzane w więcej niż jednej iteracji,
4. jeśli zbiór danych pomiędzy kolejnymi odwołaniami do tej samej strony musi zostać zachowany w obiekcie Session bądź Cache,
5. jeśli wyniki są przekazywane do obiektów warstwy biznesowej i usług Web Service.

Uwaga

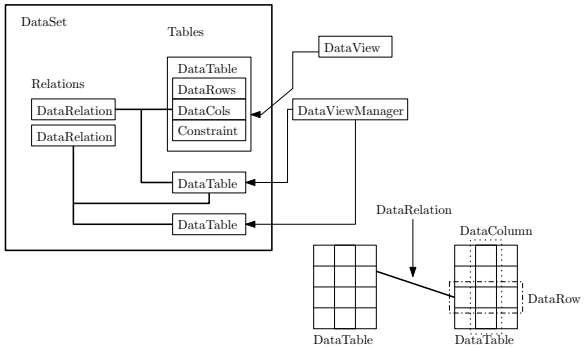
Odłączony obiekt **DataSet** może być „serializowany" do postaci XML i przesyłany z wykorzystaniem protokołu HTTP.

Notatki

Czym jest DataSet?

Klasa DataSet jest obiektem umieszczonym w pamięci operacyjnym reprezentującym zbiór danych w relacyjnej bazie danych, dostęp do danych jest realizowany poprzez interfejs zorientowany obiektowo:

- zbiory danych mogą zawierać wiele obiektów DataTables,
- relacje pomiędzy tabelami są reprezentowane przez klasę DataRelations,
- ograniczenia oraz klucze główne i obce są honorowane,
- dane dostępne w tabeli są reprezentowane przez klasy DataRow i DataColumn.



Notatki

Opis klasy DataSet

Klasa DataSet zawiera min.

- kolekcję DataTables
- kolekcję DataRelations

Natomiast DataTables zawiera min.

- kolekcję DataTableColumns (schema)
- kolekcję DataTableRows (data)
- tzw. DefaultView (DataTableView)

Klasa DataRelations

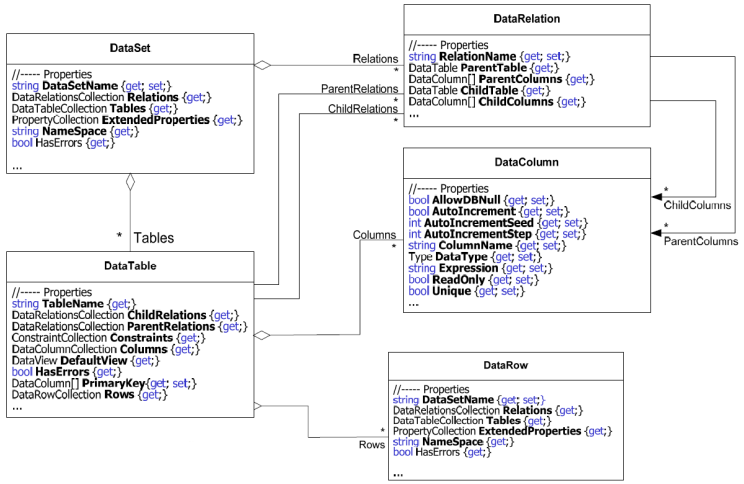
- łączy dwa obiekty DataTable
- określa ParentTable (ParentColumns) oraz ChildTable (ChildColumns)

Klasa oferuje dostęp do

- indywidualnych obiektów DataTable
  - przez indeks DataSet.Tables[0]
  - przez nazwę DataSet.Tables["tablename"]
- indywidualnych obiektów DataColumn
  - przez indeks DataTable.Columns[0]
  - przez nazwę DataTable.Columns["columnname"]
- indywidualnych obiektów DataRow
  - przez indeks DataTable.Rows[0]
- indywidualnych pól
  - przez indeks DataRow[0]
  - przez nazwę DataRow["columnname"]

Notatki

Schemat UML – DataSet



Notatki

DataAdapter – obiekt pośredniczący

Klasa DataAdapter jest obiektem pośredniczącym pomiędzy obiektem DataSet a źródłem danych (serwer) w operacjach otrzymywania i zapisywania danych. Klasa DataAdapter reprezentuje zbiór poleceń dla bazy danych oraz połączenie do bazy danych, jej zadaniem jest wypełnienie oraz aktualizacja danych w zbiorze danych DataSet. Każdy obiekt DataAdapter wymienia dane pomiędzy obiektem DataTable a rezultatem poleceń SQL, bądź też procedurą wbudowaną w określonej bazie danych. Własności klasy DataAdapter odnoszą się do następujących pojęć:

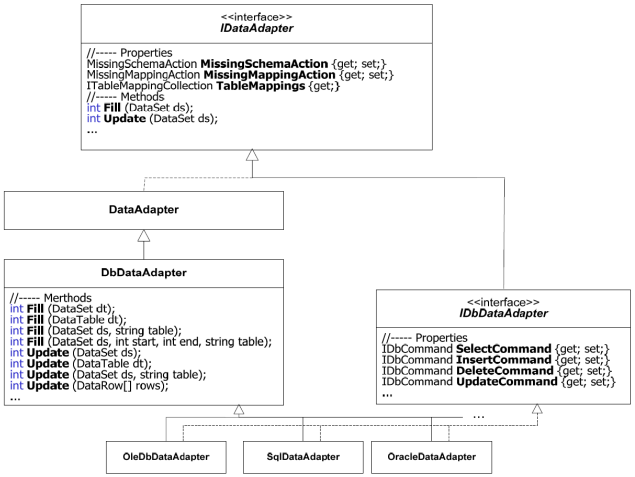
- SelectCommand - otrzymywanie rekordów/wierszt ze źródła danych,
- InsertCommand - zapisywanie wierszy ze zbioru do źródła danych,
- UpdateCommand - zapisywania zmian ze zbioru do źródła danych,
- DeleteCommand - usuwanie wierwszy ze źródła danych.

Metody stosowane przez DataAdapters

- Fill - dodanie bądź odświeżenie wierszy ze źródła danych w obiekcie DataSet (stosowanie polecenia SELECT),
- Update - przesłanie dokonanych zmian w DataSet do źródła danych (polecenia typu INSERT, UPDATE lub DataTable).

Notatki

DataAdapter schemat UML



Notatki

Tworzenie zbioru danych

Przykład obiektu adaptera pośredniczącego pomiędzy zapytaniem SQL a obiektem **DataSet**:

```
private static DataSet SelectRows(DataSet dataset,
        string connectionString, string queryString)
{
    using (SqlConnection connection = new SqlConnection(connectionString)) {
        SqlDataAdapter adapter = new SqlDataAdapter();
        adapter.SelectCommand = new SqlCommand(
            queryString, connection);
        adapter.Fill(dataset);
        return dataset;
    }
}
```


Notatki

Widok danych

Klasa DataView reprezentuje widok danych, widok ten może być modyfikowany poprzez dodatkowe operacje sortowanie, filtrowania, wyszukiwania, edycji oraz nawigacji określonej przez użytkownika. Jeden z konstruktorów:

```
public DataView(
    DataTable table,
    string RowFilter,
    string Sort,
    DataViewRowState RowState
)
```

Utworzenie widoku:

```
DataView view = new DataView(custDS.Tables["Customers"],
    "Country = 'France'",
    "ContactName",
DataViewRowState.CurrentRows);

view.AllowEdit = true;
view.AllowNew = true;
view.AllowDelete = true;
```


Notatki

Tworzenie własnej tabeli i widoku

Utworzenie tabeli i wypełnianie jej danymi:

```
DataTable table = new DataTable("table");
DataColumn colItem = new DataColumn("item",
    Type.GetType("System.String"));
table.Columns.Add(colItem);

DataRow NewRow;
for(int i = 0; i < 5; i++) {
    NewRow = table.NewRow();
    NewRow["item"] = "Element " + i;
    table.Rows.Add(NewRow);
}

table.Rows[0]["item"] = "kot";
table.Rows[1]["item"] = "pies";
table.AcceptChanges();
```


Notatki

Tworzenie własnej tabeli i widoku

Przygotowanie widoków:

```
DataView firstView = new DataView(table);
DataView secondView = new DataView(table);
PrintData( table );

firstView.RowStateFilter=DataViewRowState.ModifiedOriginal;
PrintData( firstView );

DataRowView rowView;
rowView=secondView.AddNew();
rowView["item"] = "fish";

secondView.RowStateFilter=DataViewRowState.ModifiedCurrent  | DataViewRowState.Added;
PrintData( secondView );
```


Notatki

Modyfikacja danych w wierszach

Modyfikacje danych w tabeli: (a) Metoda **BeginEdit** (wyłącza zgłaszanie zdarzeń oraz wyjątków), (b) metody **EndEdit** oraz **CancelEdit** (przywraca zgłaszanie zdarzeń oraz wyjątków),

```
drEmployee DataRow = dtEmployees.Rows(3)
drEmployee.BeginEdit()
drEmployee("FirstName") = "John"
drEmployee("LastName") = "Smith"
drEmployee.EndEdit()
```

Inne operacje:

- utworzenie nowego wiersza: `DataRow workRow = workTable.NewRow();`
- podanie danych: `workRow[0] = "1"; workRow["CustLName"] = "Smith";`
- dołączenie wiersza do DataTable: `workTable.Rows.Add(workRow);`
- jak wyżej ale w jednym kroku:
  `workTable.Rows.Add(new Object[] {1 "Smith"});`
- całkowite usunięcie wiersza: `dtEmployees.Rows.Remove(drEmployee);`
- zaznaczenie wiersza do usunięcia: `workRow.Delete();`

Notatki

Śledzenie zmian

Obiekt DataRow może przechowywać wiele wersji wartości dla każdej z kolumn:

- DataRowVersion.Current – aktualna zawartość,
- DataRowVersion.Original – oryginalna zawartość przed zmianami,
- DataRowVersion.Proposed – wartość proponowano w trakcie sesji z metodami BeginEdit / EndEdit,
- DataRowVersion.Default – wartość domyślna dla stanu wiersza (np.: dane oryginalne dla usuwanych wierszy)

Każdy wiersz posiada własność **RowState** przyjmującą następujące wielkości: Added, Deleted, Detached, Modified, Unchanged. Dwie istotne metody to min:

- AcceptChanges wprowadzenie zmian: RowState przyjmuje wartość Unchanged, zawartość Current jest kopiowana do Original,
- RejectChanges odrzucenie zmian: : RowState przyjmuje wartość Unchanged, zawartość Original jest kopiowana do Current.

Notatki

Modyfikacja danych w wierszach

Reakcja na wprowadzone zmiany do kolumny:

```
DataTable workTable = new DataTable();
workTable.Columns.Add("LastName", typeof(String));

workTable.ColumnChanged += new DataColumnChangeEventHandler(OnColumnChanged);
DataRow workRow = workTable.NewRow();
workRow[0] = "Smith";
workTable.Rows.Add(workRow);
```

Wywołanie zdarzenia:

```
workRow.BeginEdit();
workRow[0] = "";
workRow.EndEdit();
```

Postać funkcji obsługującej zdarzenie:

```
protected static void OnColumnChanged(Object sender, DataColumnChangeEventArgs args) {
    if (args.Column.ColumnName == "LastName")
        if (args.ProposedValue.ToString() == "") {
            Console.WriteLine("Last Name cannot be blank. Edit canceled.");
```


Notatki

Kontenery, źródła danych i kontrolki

Dostęp do pojemników danych:

- Data Connection: umożliwia połączenie z bazą danych
- Data Source: źródło danych umożliwia odczyt i aktualizację danych
- Data-bound Control: kontrolka wiążąca dane wyświetla informacje przekazywane prze źródło danych lub przekazuje dane do źródła

Kontrolki źródeł danych:

- SqlDataSource: umożliwia odczytywanie i zmianę informacji w bazie danych MS SQL
- XMLDataSource: odczyt i zmiana informacji w plikach XML
- ObjectDataSource: odczyt i zmiana danych zawartych w niestandardowych obiektach
- SiteMapDataSource: odczyt informacji z pliku mapy witryny
- LinqDataSource: odczyt i modyfikacja danych z użyciem języka LINQ

Kontrolki wyświetlające dane:

- GridView: Wyświetlanie rekordów w formie arkusza, umożliwia modyfikację i usuwanie danych
- DetailsView: wyświetla jeden rekord; wstawianie, modyfikacja, usuwanie
- FormView: wyświetla jeden rekord w sformatowanej postaci; wstawianie, modyfikacja, usuwanie

Notatki

Dwa modele transakcji

Dostępne są dwa modele transakcji:

- lokalne transakcje:
  - transakcje do pojedynczego połączenia
  - obsługiwane przez ADO.NET
- rozproszone transakcje:
  - transakcje dla wielu połączeń
  - stosowany jest Microsoft Distributed Transaction Component (MSDTC)
  - obsługa realizowana przez System.Transaction

Wykonywanie transakcji składa się następujacych kroków:

- wywołanie metody BeginTransaction z obiektu SqlConnection, oznacza rozpoczęcie transakcji, metoda BeginTransaction zwraca referencję do obiektu reprezentującego transakcję, zawierającego odniesienia do obiektów SqlCommand,
- przypisanie obiektu transakcji do obiektu SqlCommand, wykonanie polecenia SQL wymaga aktywnej transakcji, w przeciwnym przypadku zgłoszony zostanie wyjątek,
- wykonanie właściwych poleceń składających się na transakcję,
- wywołanie metody **Commit** z obiektu **SqlTransaction** kończy transakcję, wywołanie metody **Rollback** odwołuje transakcję. W przypadku utraty połączenia transakcja zawsze jest odwoływana.

Notatki

Przykład

Wykorzystanie lokalnej transakcji dla systemu MS SQL Server:

```
using (SqlConnection connection = new SqlConnection(connectionString)) {
    connection.Open();

    SqlTransaction sqlTran = connection.BeginTransaction();
    SqlCommand command = connection.CreateCommand();
    command.Transaction = sqlTran;
    try  {
        command.CommandText =
           "INSERT INTO Production.ScrapReason(Name) VALUES('Wrong size')";
        command.ExecuteNonQuery();
        command.CommandText =
           "INSERT INTO Production.ScrapReason(Name) VALUES('Wrong color')";
        command.ExecuteNonQuery();
        sqlTran.Commit();
    }
    catch (Exception ex) {
        Console.WriteLine(ex.Message);
        try {
            sqlTran.Rollback();
        }
        catch (Exception exRollback) {
            Console.WriteLine(exRollback.Message);
        }
    }
}
```


Notatki

Charakterystyka LINQ

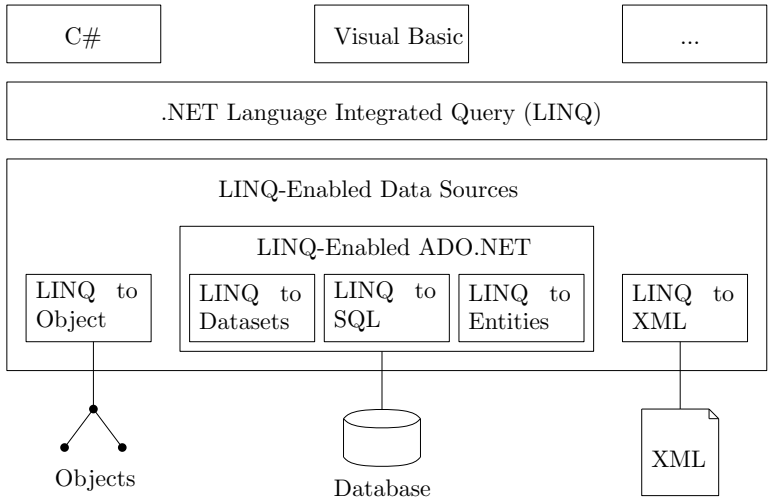
Technologia LINQ (Language-Integrated Query), to rozszerzenie języka C# (oraz Visual Basic .NET) obsługujące zapytania do danych reprezentowanych przez następujące źródła:

- obiekty, kolekcje dostępne w BCL,
- dane zapisane w XML,
- bazy danych systemu SQL Server,
- bazy danych ADO.NET,
- encje (ang. entities).

Zaletą LINQ, jest ujednolicenie dostępu do danych, a także zaoferowanie wspólnych metod ich modyfikacji, bowiem nie trzeba stosować oddzielnego języka zapytań. LINQ jest też technologią, która ukrywa detale związane ze źródłem danych. To samo zapytanie może zostać zadane do obiektu, a także do tabeli w bazie danych.

Notatki

Charakterystyka LINQ



Notatki

Postać zapytań

Syntaktyka zapytań LINQ jest podobna do pytań SQL, jednak treść zapytań jest tłumaczona do języka C#:

```
from id in source
  { from id in source |
 join id in source on expr equals expr [ into id ] |
    let id = expr |
    where condition |
    orderby ordering1, ordering2, ... }
select expr | group expr by key
    [ into id query ]
```


Notatki

Zapytania LINQ do obiektów

Najczęściej zapytania LINQ składają się z trzech etapów:

1. określenie dostępu do danych,
2. utworzenie postaci zapytania,
3. wykonanie zapytania.

```
int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

var numQuery = from num in numbers
               where (num % 2) == 0
               select num;

foreach (int num in numQuery) {
    Console.Write("{0,1} ", num);
}
```

Można wskazać trzy główne zalety LINQ względem typowych pętli foreach:

1. większa czytelność i precyzja, zwłaszcza w przypadku obecności wielu dodatkowych warunków,
2. niewielki nakład dodatkowego kodu w przypadku realizacji operacji filtrowania, porządkowania i grupowania,
3. są w bardzo dużym stopniu niezależne od źródeł danych.

Notatki

Zgodność obiektu z LINQ

Zapytania mogą być kierowane do obiektów zgodnych z interfejsem IEnumerable oraz `IEnumerable<T>`.

Uwaga

Dowolna zmienna, dowolny obiekt zgodny z `IEnumerable<T>` dla pewnego typu T jest traktowany jako sekwencja typów T.

Zestaw operatorów standardowych

Restriction	Where
Grouping	GroupBy
Quantifiers	Any, All
Aggregation	Count, Sum, Min, Max, Average
Projection	Select, SelectMany
Partitioning	Take, Skip, TakeWhile, SkipWhile
Conversion	ToArray, ToList, ToDictionary
Ordering	OrderBy, ThenBy
Sets	Distinct, Union, Intersect, Except
Casting	`OfType<T>`
Elements	First, FirstOrDefault, ElementAt

Notatki

Przykład translacji zapytania

Proste zapytanie do tablicy:

```
string[] names = { "Burke", "Connor",
                   "Frank", "Everett",
                   "Albert", "George",
                   "Harris", "David" };

var result = from n in names
             where n.Length == 5
             orderby n
             select n.ToUpper();

foreach (string item in result)
    Console.WriteLine(item);
```

To samo zapytanie zrealizowane w sposób bezpośredni:

```
Func<string, bool> filter = s => s.Length == 5;
Func<string, string> extract = s => s;
Func<string, string> project = s => s.ToUpper();
IEnumerable<string> expr = names
                .Where(filter).OrderBy(extract)
                .Select(project);
foreach (string item in expr)
    Console.WriteLine(item);
```


Notatki

Zapytanie do tablicy

Zapytanie do listy procesów, o procesy posiadające więcej niż sześć wątków:

```
var results =
     from p in Process.GetProcesses()
     select p;

var results =
     from p in Process.GetProcesses()
        where p.Threads.Count > 6
        orderby p.ProcessName descending
        select new {p.ProcessName, ThreadCount = p.Threads ;

foreach (var o in results) {
     Console.WriteLine(o.ToString());
}
```


Notatki

Zapytanie do listy

Obiekt reprezentujący pracowników i lista pracowników:

```
public class Employee {
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Decimal Salary { get; set; }
    public DateTime StartDate { get; set; }
}
 var employees = new List<Employee> {
    new Employee {
        FirstName = "Joe",  LastName = "Bob",
        Salary = 94000, StartDate = DateTime.Parse("1/4/1992") },
        ...
        };
```


Notatki

Zapytanie do listy

Zapytanie o najlepiej zarabiających pracowników:

```
var query = from employee in employees
    where employee.Salary > 100000
    orderby employee.LastName, employee.FirstName
        select new {
            LastName = employee.LastName,
            FirstName = employee.FirstName
        };

foreach (var item in query) {
    Console.WriteLine("{0}, {1}",
        item.LastName, item.FirstName);
}
```


Notatki

Zapytania do ciągu znaków

Liczba wystąpień wskazanego słowa w podanym tekście:

```
string text=@"jakis tekst ....";
string searchTerm = "słowo";
string[] source = text.Split(new char[] {
                        '.', '?', '!', ' ', ';', ':', ',' },
        StringSplitOptions.RemoveEmptyEntries);

var matchQuery = from word in source
     where word.ToLowerInvariant() == searchTerm.ToLowerInvariant()
     select word;
int wordCount = matchQuery.Count();
```


Notatki

LINQ oraz pliki i katalogi

Grupowanie plików ze względu na rozszerzenia:

```
string startFolder = @"\ścieżka\do\katalogu";

int trimLength = startFolder.Length;
System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(startFolder);

IEnumerable<System.IO.FileInfo> fileList = dir.GetFiles("*.*",
                            System.IO.SearchOption.AllDirectories);

var queryGroupByExt =
   from file in fileList
   group file by file.Extension.ToLower() into fileGroup
   orderby fileGroup.Key
   select fileGroup;
```


Notatki

Własne metody w zapytaniach LINQ

Obliczanie Mediany dla zbioru danych:

```
public static class LINQExtension {
    public static double Median(this IEnumerable<double> source) {
        if (source.Count() == 0) {
            throw new InvalidOperationException( " ... ");
        }
        var sortedList = from number in source
                         orderby number
                         select number;
        int itemIndex = (int)sortedList.Count() / 2;
        if (sortedList.Count() % 2 == 0) {
            return (sortedList.ElementAt(itemIndex)
                        + sortedList.ElementAt(itemIndex - 1)) / 2;
        }
        else {
            return sortedList.ElementAt(itemIndex);
        }
    }
}
```

Metoda zostanie dodana do LINQ za pomocą techniki metod rozszerzających.


```
double[] numbers1 = { 1.9, 2, 8, 4, 5.7, 6, 7.2, 0 };
var query1 = numbers1.Median();
```

Notatki

Zapytania do DataSet

Zbiory danych są obsługiwane bezpośrednio przez LINQ

- silnie i słabo typowalne zbioru danych (DataSet,
- dane zgrupowane i połączone (groupby, join),
- widoki utworzone z wielu tabel.

Przykład zapytania w odniesieniu do zbioru danych:

```
DataSet ds = new DataSet();
FillTheDataSet(ds); // wypełnienie danymi
DataTable dtPeople = ds.Tables["People"];
IEnumerable<DataRow> query =
    from people In dtPeople.AsEnumerable()
    select people;
foreach (DataRow p in query)
    Response.Write(p.Field<string>(\FirstName"));
```


Notatki

Zapytania do SQL

LINQ korzysta z odwzorowania (object-to-relational mapping (ORM)) obiektu do relacji, który jest obecny w bazach danych Microsoft SQL Server. Dostęp do danych jest silnie typowalny.
Dostęp zintegrowany na poziomie języka .NET

- odwzorowanie tabel i wierszy do klas i obiektów,
- wykorzystywany jest mechanizm transakcji ADO.NET oraz .NET Transactions.

Odwzorowanie relacji:

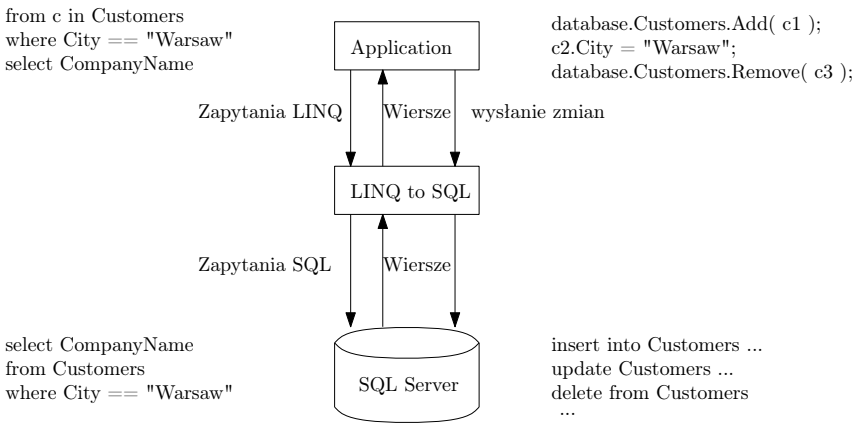
- zakodowane w atrybutach klas,
- generowane samodzielnie przez odpowiednie narzędzia lub napisane ręcznie,
- relację pomiędzy tabelami są opisywane przez własności.

Stałość danych

- samodzielne śledzenie zmian,
- aktualizacji poprzez SQL lub procedury wbudowane.

Notatki

LINQ oraz SQL



Notatki

Przykład zapytania SQL

Przed utworzeniem zapytania LINQ należy utworzyć obiekt DataContext. Jest on odpowiedzialny za tłumaczenie wyrażeń LINQ do SQL, oraz realizacją odwzorowań wyników do obiektów. Obiekt DataContext reprezentuje bazę danych, i zawiera tabele, widoki a także procedury wbudowane.

```
CustomersDC dbCust = new CustomersDC();
var query = from p in dbCust.Customers
    where p.City == "Warsaw"
    select p.Name, p.Country;

foreach (var c in query) {
  // c.Name
  // c.Country
}
```


Notatki

Klasa DataContext

Klasa DataContext to podstawowa klasa stosowana w przypadku zapytań LINQ do baz opartych o SQL:

- jest źródłem wszystkich „bytów" z bazy danych,
- śledzi wszystkie zmiany w danych otrzymywanych z bazy danych,
- utrzymuje spójność, co oznacza, że otrzymywane z bazy danych informacje np.: o tabeli zawsze są reprezentowane przez jeden obiekt.

Baza danych jest reprezentowana przez tzw. plik „LINQ to SQL Class" zawierający klasy odwzorowujące bazę danych. Wszystkie typy danych są wnioskowane na podstawie typów danych w tabelach, a nazwy metody są nazwami metod wbudowanych.

Notatki

Tworzenie obiektu DataContext

Klasa Customer reprezentująca wiersz w tabeli Customers:

```
[Table(Name = "Customers")]
public class Customer {
   [Column(IsPrimaryKey = true)]   public string CustomerID { get; set; }
   [Column]   public string CompanyName { get; set; }
   ...
   [Column] public string City { get; set; }
   [Column] public string Region { get; set; }
   ...
}
```

Obiekt DataContext:

```
DataContext context = new DataContext(
     "Initial Catalog=petdb;Integrated Security=sspi");
Table<Customer> custs = context.GetTable<Customer>();

var query = from c in custs
    where c.City = "London"
    select new { c.Name, c.Phone };
```


Notatki

Dodawanie danych – słowo INSERT

Operacja dodawania danych stosuje podejście obiektowe (metoda **InsertOnSubmit**), przetłumaczenie na SQL zostaje wykonane samodzielnie przez mechanizmy LINQ:

```
PersonDataClassesDataContext dbPeople = new PersonDataClassesDataContext();
People objPeople = new People();
objPeople.FirstName = "Gyan";
objPeople.LastName = "Singh";
objPeople.Age = 33;
dbPeople.Peoples.InsertOnSubmit(objPeople);
dbPeople.SubmitChanges();
```

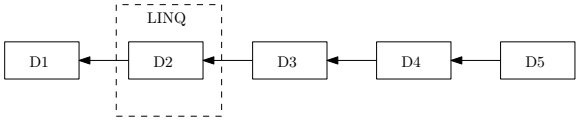
Usunięcie wierszy jest realizowana za pomocą metody **DeleteOnSubmit**:

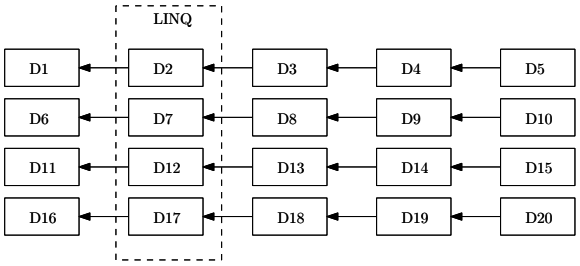
```
var query = from p in dbPeople.Peoples
    where p.PersonID == 1
    select p;
if (query.Count() > 0) {
    dbPeople.Peoples.DeleteOnSubmit(query.First());
    dbPeople.SubmitChanges();
}
```


Notatki

Zapytania równoległe LINQ

Zapytania równoległe LINQ pozwalają na wykorzystanie dodatkowych rdzeni obliczeniowych dostępnych we współczesnych systemach informatycznych.





Notatki

Test wydajności – Intel Core 2 Duo 8400 3.0 Ghz

Wydajność zapytania sekwencyjnego (56350 ms):

```
IEnumerable<int> numbers1 = Enumerable.Range(0, Int32.MaxValue);
Stopwatch sw = Stopwatch.StartNew();
int sum1 = (from n in numbers1
            where n % 2 == 0
            select n).Count();
Console.WriteLine("Rezultat: {0}", sum1);
Console.WriteLine("Czas: {0} ms", sw.ElapsedMilliseconds);
```

Wydajność zapytania równoległego (30182 ms):

```
IEnumerable<int> numbers2 = ParallelEnumerable.Range(0, Int32.MaxValue);
sw.Restart();
int sum2 = (from n in numbers2.AsParallel()
            where n % 2 == 0
            select n).Count();
Console.WriteLine("Parallel result: {0}", sum2);
Console.WriteLine("Parallel time: {0} ms", sw.ElapsedMilliseconds)
```


Notatki

Proste zapytania równoległe

Słowa zawierające literę „o", ułożone alfabetycznie:

```
IEnumerable<string> results = from p in words.AsParallel().AsOrdered()
    where p.Contains('o')
    select p;
foreach (string w in results)
    Console.WriteLine("    {0}", w);
```

Wpływanie na podział partycji danych do przetwarzania równoległego:

```
IEnumerable<string> results = presidents
    .AsParallel()
    .WithDegreeOfParallelism(2)
    .Where(p => p.Contains('o'))
    .Select(p => p);
```

Wartość dla metody **WithDegreeOfParallelism** ma naturalnie duży wpływ na wydajność.

Notatki

Metody ForAll i Range

Metoda ForAll stosuje podaną akcję **System.Action** do każdego elementu stanowiącego rezultat zapytania:

```
words.AsParallel()
    .Where(p => p.Contains('o'))
    .ForAll(p => Console.WriteLine("słowo: {0}", p));
```

Notacja LINQ:

```
ParallelQuery<string> pq = words.AsParallel();

IEnumerable<string> results = from p in pq
    where p.Contains('o')
    select p;
```

Operator zakresu:

```
ParallelQuery<int> pq = ParallelEnumerable.Range(0, 10);

IEnumerable<int> results = from i in pq
```


Notatki

Zgłaszanie wyjątków w zapytaniach PLINQ

Zgłoszenie wyjątku powoduje przerwanie wykonywanie zapytania:

```
IEnumerable<string> results = presidents
    .Select(p => {
        if (p == "Arthur")
            throw new Exception(String.Format("problem -> {0}", p));
return p;
});
try {
    foreach (string w in results)
        Console.WriteLine(" -> {0}", w);
} catch (Exception ex) {
Console.WriteLine(ex.Message);
}
```


Notatki

Zgłaszanie wyjątków w zapytaniach PLINQ

W przypadku równoległych zapytań wyjątek może pojawić się np.: w jednej bądź w wielu partycjach, możliwe jest zebranie wszystkich wyjątków dzięki klasie **AggregateException**:

```
IEnumerable<string> results = presidents
.AsParallel()
.Select(p => {
    if (p == "Arthur" || p == "Harding")
        throw new Exception(String.Format("problem -> {0}", p));
    return p;
});
try {
    foreach (string w in results) {
        Console.WriteLine(" -> {0}", w);
}
} catch (AggregateException agex) {
    agex.Handle(ex => {
        Console.WriteLine(ex.Message);
        return true;
    });
}
```


Notatki

Entity Framework

Notatki

Plan wykładu

1. Wprowadzenie
   - 1.1 odrobina historii,
   - 1.2 wersje pakietów EF,
   - 1.3 najważniejsze założenia.
2. Architektura i model EF,
   - 2.1 miejsce EF w aplikacji,
   - 2.2 architektura EF,
   - 2.3 rodzaje i stany encji,
   - 2.4 trzy podejścia do współpracy z danymi.
3. Przykłady praktyczne
   - 3.1 podejście „code-first”.

Notatki

Entity Framework

Entity Framework (EF) – technologia/pakiet/biblioteka/framework stosowany do dostępy do danych poprzez mapowanie danych do obiektów (ang. object-relational mapper – ORM). EF pozwala na współpracę ze źródłem danych poprzez obiekty z modelu obiektowego .NET.

Krótko o EF, w postaci poniższej tabeli:

EF6	EF Core
pierwsza wersja w 2008, .NET 3.5 SP1	pierwsze wydanie w 2016 .NET Core 1.0
wersja stabilna	nowe funkcje i rozwój
Windows	Linux, MacOS, Windows
wsp. z .NET 3.5+	wsp. z .NET 4.5+ oraz .NET Code
open source	open source

Notatki

Wydania wersji EF, do najnowszej wersji z rodziny 6.x:

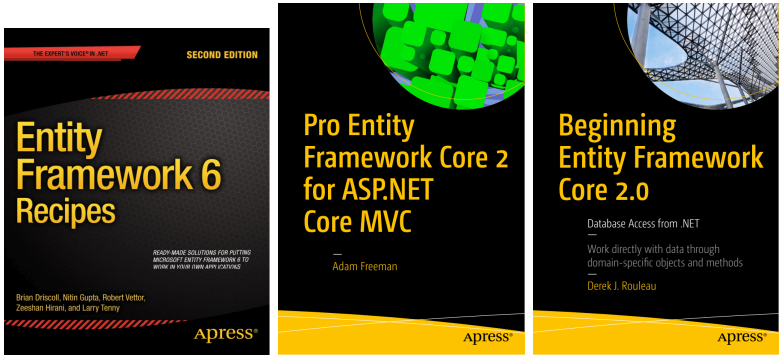
- EF 1.0 (or 3.5), 2008, .NET 3.5 SP1, VS 2008,
- EF 4.0, 2010, .NET 4.0, VS 2010,
- EF 4.3, 2011, .NET 4.0, VS 2012,
- EF 5, 2012, .NET 4.0, VS 2012,
- EF 6, 2013, .NET 4.0 & .NET 4.5, VS 2012,
- ...
- EF 6.4, 2019, ostatnie wydanie tej gałęzi EF.

Wydania wersji EF Core, do najnowszej wersji z rodziny 2.x:

- EF Core 1.0, June 2016, .NET Core 1.0,
- EF Core 1.1, November 2016, .NET Core 1.1,
- EF Core 2.0, August 2017, .NET Core 2.0, VS 2017,
- ...
- Entity Framework Core 7.0 (EF Core 7), 8 November 2022, .NET 6 LTS, .NET 7, VS 2022.

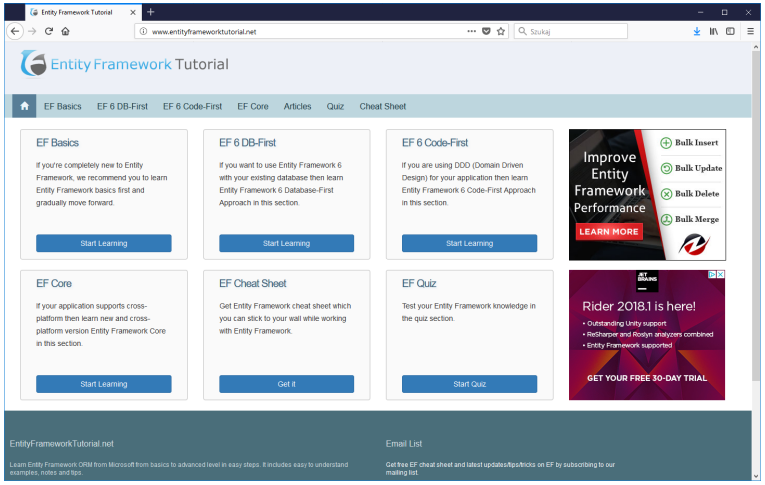
Notatki

Kilka pozycji książkowych:



Notatki

Przydatna strona, gdy rozpoczynamy poznawać EF:

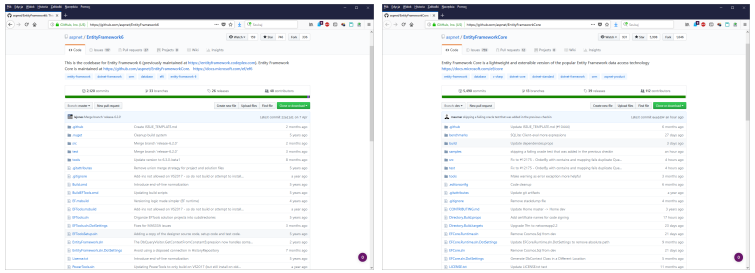


Kod źródłowy: `https://github.com/entityframeworktutorial`

Notatki

Notatki

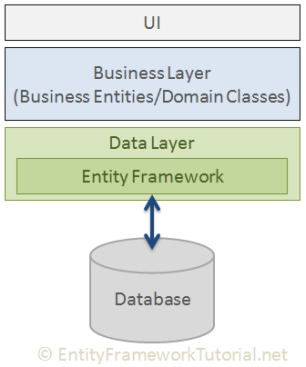
Warto także zajrzeć na strony z kodem źródłowym EF:



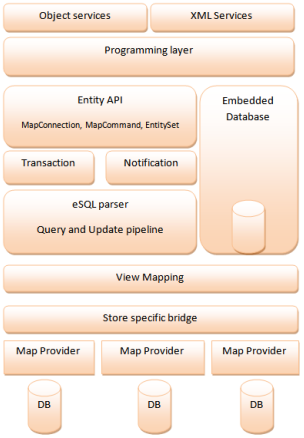
Adresy: `https://github.com/aspnet/EntityFramework6`, `https://github.com/aspnet/EntityFrameworkCore`.

Notatki

Miejsce EF w stosie aplikacyjnym

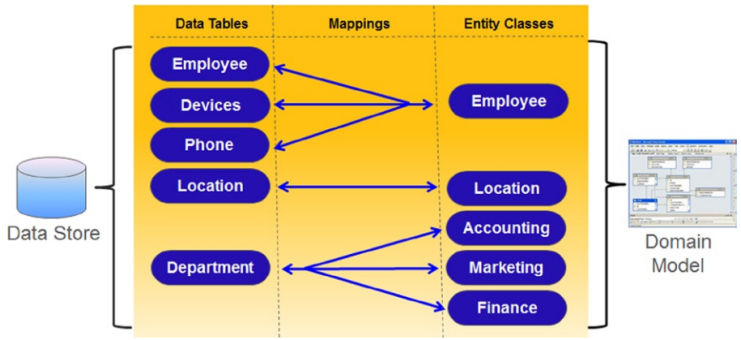


Rysunek pochodzi z: `http://www.entityframeworktutorial.net/`.



Rysunek pochodzi z:
`https://en.wikipedia.org/wiki/Entity_Framework`.

Notatki



Źródło: Entity Framework 6 Recipes, Hirani, Z., Tenny, L., Gupta, N., Driscoll, B., Vettor, R. , strona 3, Apress 2013.

Notatki

Główne cechy i zalety EF

Najważniejsze zalety EF:

- wieloplatformowość: EF Core funkcjonuje na trzech głównych systemach: Windows, Linux and Mac.
- modelowanie: EF tworzy tzw. EDM (Entity Data Model) oparty o obiekty/encje POCO (Plain Old CLR Object) wraz z odpowiednim zbiorem własności dla łatwiej obsługi różnych typów danych. Operacje na modelu odzwierciedlają operacje na bazie danych.
- zapytania: EF pozwala na stosowanie zapytań LINQ (C#/VB.NET) do odbierania danych z bazy danych. Baza danych powinna dostarczać odpowiedni obiekt do translacji zapytań LINQ do języka zapytań charakterystycznego dla danego systemu bazodanowego. EF umożliwia również wykonywanie zapytań SQL bezpośrednio do danej bazy danych.
- śledzenie zmian: EF śledzi też zmiany lokalnych instancji określonych encji i samodzielnie wprowadza odpowiedni zmiany do bazy danych.
- zapis/trwałość: EF wykonuje zapytania/polecenia INSERT, UPDATE, oraz DELETE na bazie danych w zależności od zmian wprowadzonych do encji, szczególnie po wywołniu metody SaveChanges(). EF dostarcza także metodę asynchroniczną SaveChangesAsync().

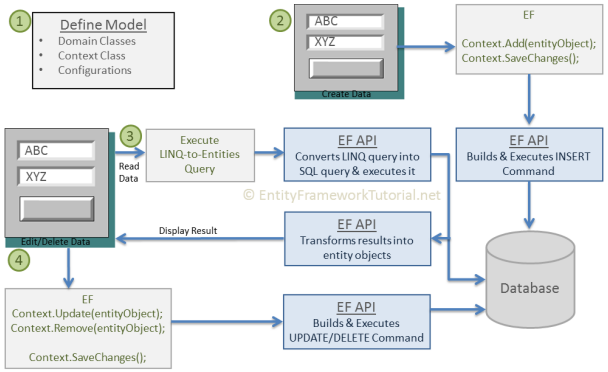
Notatki

Najważniejsze zalety EF ciąg dalszy:

- współbieżność (concurrency): EF stosuje tzw. „Optimistic Concurrency", aby zabezpieczyć dane przed nadpisaniem podczas pobierania danych z bazy danych,
- transakcje (transactions): Entity Framework wykonuje automatyczne zarządzanie transakcjami podczas zapytań oraz zapisu danych, możliwe jest również własne zarządzania transakcjami,
- buforowanie (caching): dostępny jest bufor pierwszego stopnia, oznacza to, iż powtarzanie zapytania zwraca dane z bufora zamiast ponownego przesyłania zapytania do serwera bazy danych,
- rozwiązania wbudowane (built-in conventions): dostępne są gotowe rozwiązania oraz zbiór reguł do samodzielnego tworzenia modelu EF bez jawnej jego specyfikacji,
- konfiguracje (configurations): EF umożliwia konfigurację modelu poprzez stosowanie adnotacji oraz atrybutów, dopuszcza się też stosowanie Fluent API do nadpisania domyślnych zasad tworzenia modelu,
- migracje (migrations): EF dostarcza także polecenia migracyjne wykonywane z poziomu konsoli menadżera pakietów NuGet lub linii poleceń (Command Line Interface) do tworzenia i zarządzania schematem bazy danych.

Notatki

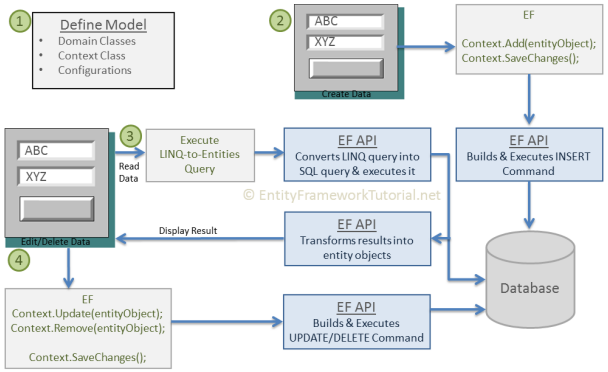
EF, schemat postępowania:



(1) Definicja modelu to pierwszy element, jaki należy wykonać. Przynależą do niego klasy dziedzin, klasa kontekstu wyprowadzona z DBContext oraz tzw. konfiguracje. Operacje typu CRUD będą realizowane poprzez zdefiniowany model.

Notatki

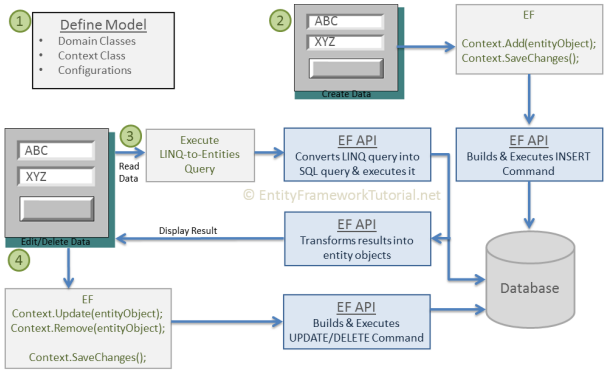
EF, schemat postępowania:



(2) Dodanie danych wymaga zastosowanie obiektu dziedzinowego do kontekstu oraz wywołania metody SaveChanges(). API EF samodzielnie wykona polecenie INSERT, aby zmodyfikować bazę danych.

Notatki

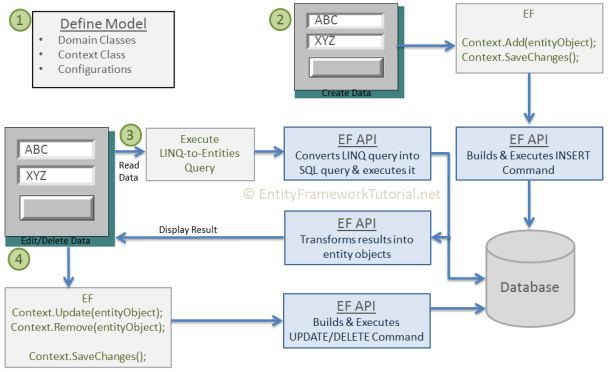
EF, schemat postępowania:



(3) Odczyt danych wykorzystuje zapytania LINQ. Tłumaczenie na SQL odbywa się automatycznie, a rezultat także samodzielnie przez EF jest zamieniany na obiekty i dostępny dla pozostałych elementów aplikacji.

Notatki

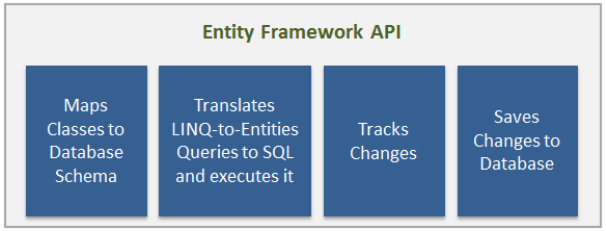
EF, schemat postępowania:



(4) Czynności związane z edycją, aktualizacją, usunięciem danych zawsze są tłumaczone na odpowiednie konstrukcje UPDATE lub DELETE po stronie serwera bazy danych.

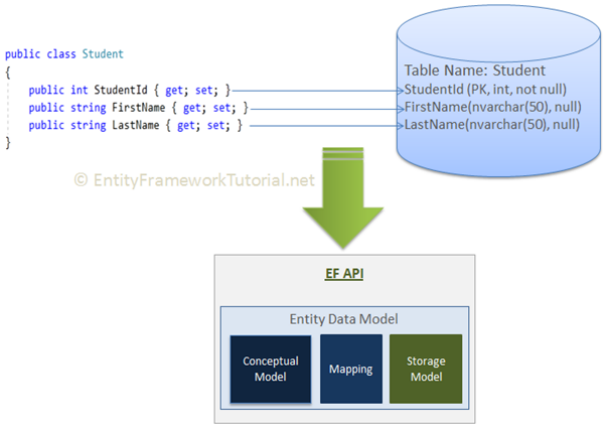
Notatki

Entity Framework API (wersje EF6 oraz EF Core) pozwalają na odwzorowanie klas dziedzin (encji) do schematu bazy danych. Samodzielnie realizowane są tłumaczenia zapytań LINQ do SQL, śledzene są zmiany w encjach w czasie życia aplikacji oraz zapisywane są zmiany do bazy danych.



Notatki

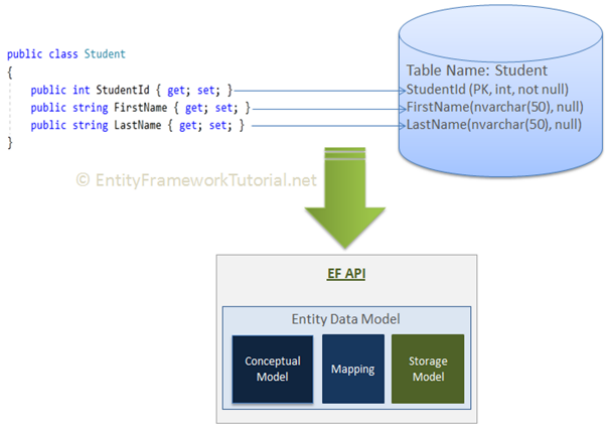
Entity Data Model (EDM), tj. model encji danych jest zawarty w pamięci operacyjnej i reprezentuje metadane: model koncepcyjny, model składu i relację pomiędzy tymi dwoma modelami.



Model koncepcyjny: jest tworzony na podstawie klas dziedzinowych, klasy kontekstu oraz domyślnych założeń podanych w klasach dziedzinowych oraz konfiguracjach.

Notatki

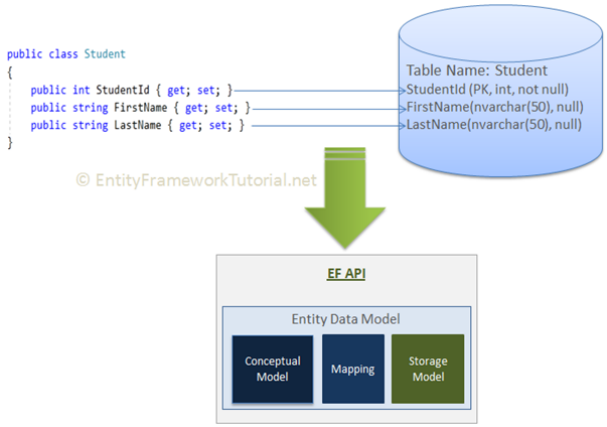
Entity Data Model (EDM), tj. model encji danych jest zawarty w pamięci operacyjnej i reprezentuje metadane: model koncepcyjny, model składu i relację pomiedzy tymi dwoma modelami.



Model składu (Storage Model): EF opracowuje model składu dla bazy danych jaka jest stosowana w projekcie. W podejściu „code-first" model ten jest wnioskowane z modelu koncepcyjnego. W podejściu „database-first", model zostanie opracowany z docelowej bazy danych.

Notatki

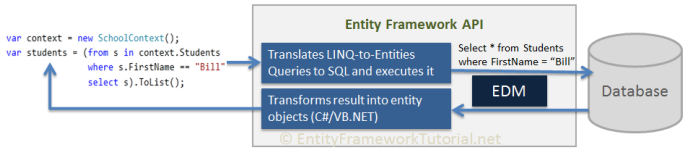
Entity Data Model (EDM), tj. model encji danych jest zawarty w pamięci operacyjnej i reprezentuje metadane: model koncepcyjny, model składu i relację pomiędzy tymi dwoma modelami.



Odwzorowanie (Mappings): EF zawiera odwzorowanie pomiędzy modelem koncepcyjnym a schematem bazy danych (modelem składowym/storage model).

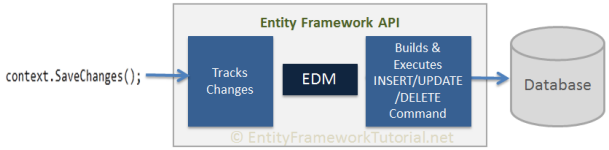
Notatki

API EF tłumaczy zapytanie „LINQ-to-Entities queries" do odpowiednich zapytań SQL uwzględniając system relacyjnej bazy danych jaki jest stosowany. Tłumaczenie odbywa się za pomocą EDM i odpowiedź jest naturalnie również samodzielnie zamieniana z powrotem do obiektów reprezentujących poszczególne encje.



Notatki

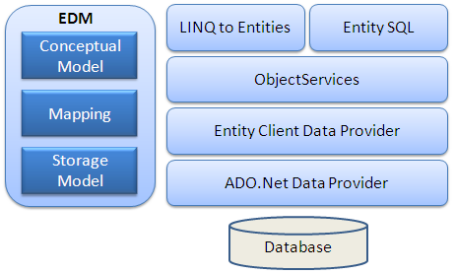
API EF jest odpowiedzialne za postać poleceń INSERT, UPDATE oraz DELETE. Polecenie te odpowiadają za ustalanie aktualnej wartości encji, podczas stosowania metody SaveChanges(). W ogólności podsystem „ChangeTrack" jest odpowiedzialny za śledzenie zmian stanów dla każdej encji, która jest używana w programie.



Notatki

Architektura EF

Ogólny schemat architektury EF:

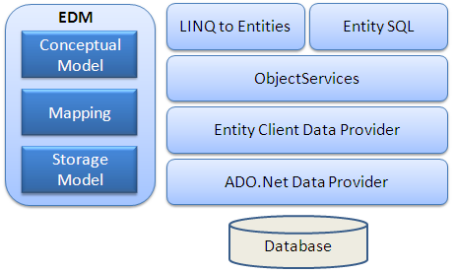


Pojęcia związane z architekturą EF:

- EDM (Entity Data Model): na EDM składają się trzy główne elementy: model koncepcyjny (conceptual model), odwzorowanie (mapping) oraz model składowy (storage model).
- Model koncepcyjny (conceptual model): zawiera klasy tworzące model oraz relacje. Model może cechować się niezależnością od postaci tabel w bazie danych.

Notatki

Ogólny schemat architektury EF:

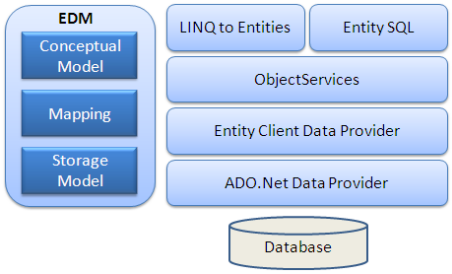


Pojęcia związane z architekturą EF:

- Model składowy (storage model): zawiera model opisujący postać bazy danych tj. tabele, widoki, procedury składowe, a także klucze oraz relacje.
- Odwzorowanie (mapping): zawiera informacje o tym jak model koncepcyjny jest odwzorowywany na model składowy.

Notatki

Ogólny schemat architektury EF:

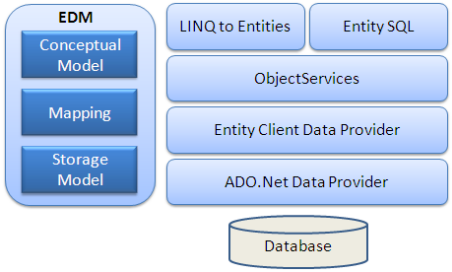


Pojęcia związane z architekturą EF:

- LINQ dla encji (LINQ to Entities): są to zapytania kierowane do modelu obiektowego, w rezultacie otrzymujemy encję określone przez model koncepcyjny.
- encje SQL (Entity SQL): to inny język zapytań (dostępny dla EF6) podobny w idei do LINQ dla encji, jednakże ogólnie jest oceniany jako trudniejszy w stosowaniu.

Notatki

Ogólny schemat architektury EF:

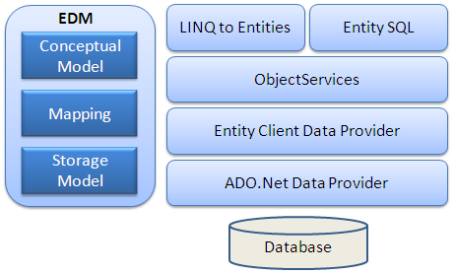


Pojęcia związane z architekturą EF:

- Usługa obiektu (object service): usługa obiektu to główna technika odpowiedzialna za dostęp do danych w bazie oraz ich odczyt lub aktualizację. W ogólności jest ona odpowiedzialne za dodatkową konwersję danych pomiędzy warstwą dostarczają dane, a obiektem encji.

Notatki

Ogólny schemat architektury EF:



Pojęcia związane z architekturą EF:

- dostarczanie danych dla encji (Entity Client Data Provider): głównym zadaniem jest konwersja LINQ do encji lub zapytania typu encje SQL do zapytań SQL w postaci związanej z konkretną bazą danych. W tym celu wykorzystywany jest obiekt danych ADO.Net, odpowiedzialny za wysyłanie oraz otrzymywanie danych z bazy danych.
- dostarczanie danych ADO.Net (ADO.Net Data Provider): warstwa odpowiedzialna za bezpośrednią komunikację z bazą danych za pomocą ADO.Net.

Notatki

Klasa kontekstu z bazą danych

Klasa kontekstu w EF jest klasą dziedziczącą z DbContext (w wersjach EF6 i Code). Klasa ta reprezentuję sesję z bazą danych. Przykładowa klasa

```
public class SchoolContext : DbContext {
public SchoolContext()
{
}

public DbSet<Student> Students { get; set; }
public DbSet<StudentAddress> StudentAddresses { get; set; }
public DbSet<Grade> Grades { get; set; }
}
```

Klasa korzysta z encji: student, adres oraz ocena.

Notatki

Klasy encji

Encja to klasa z dziedziny danych, stosowana w zbiorze danych np. DbSet<TEntity>. Przykładowe definicje:

```
public class Student
{
public int StudentID { get; set; }
public string StudentName { get; set; }
public DateTime? DateOfBirth { get; set; }
public byte[]  Photo { get; set; }
public decimal Height { get; set; }
public float Weight { get; set; }

public StudentAddress StudentAddress { get; set; }
public Grade Grade { get; set; }
}
```


Notatki

Klasy encji

Encja to klasa z dziedziny danych, stosowana w zbiorze danych np. DbSet<TEntity>. Przykładowe definicje:

```
public partial class StudentAddress
{
public int StudentID { get; set; }
public string Address1 { get; set; }
public string Address2 { get; set; }
public string City { get; set; }
public string State { get; set; }

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


Notatki

Klasy encji

Encja to klasa z dziedziny danych, stosowana w zbiorze danych np. `DbSet<TEntity>`. Przykładowe definicje:

```
public class Grade
{
public int GradeId { get; set; }
public string GradeName { get; set; }
public string Section { get; set; }

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

Klasy te są stosowane w klasie kontekstu.

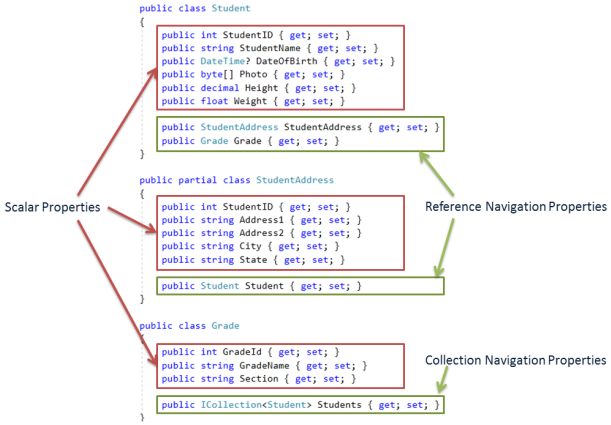
Notatki

Encje mogą zwierać dwa podstawowe typy własności: skalarne oraz nawigacyjne.

- własność skalarna (scalar property) – jest to podstawowy typ własności, wartość/własność skalarna jest odpowiedziana za reprezentację danych z bazy danych. Ogólnie, skalar reprezentuje odwzorowanie do pojedynczej kolumny w tabeli bazy danych.
- własność nawigacyjna (navigation property) – własność reprezentuje relację do innej encji. Wskazać można dwa podstawowe własności tego typu tj. nawigację referencyjną (navigation property) reprezentującą relację do innej encji oraz nawigację kolekcyjną (collection navigation):
  - własność referencyjna nawigacyjna (reference navigation property) – jeśli encja zawiera własność innej encji jest nazywana jako własność referencyjna nawigacyjna o wielokrotności jeden oznaczanej jako „(1)".
  - własność nawigacyjna kolekcyjna – jeśli encja zawiera typ kolekcji, jest nazywana własnością kolekcyjną nawigacyjną o wielokrotności wiele i oznaczoną się to jako „(*)".

Notatki

Schemat relacji między encjami oraz ich typy:



Notatki

Typy encji – 1/2

EF oferuje dwa typu/rodzaj encji „POCO" oraz „Dynamic Proxy".

- Encje POCO – Plain Old CLR Object – wykorzystane są w zapytaniach typu: select, insert, update, delete. Są one generowane przez Entity Data Model i wspierane przez EF 6 oraz EF Core,
- Dynamic Proxy (DP) to klasa opakowująca encje POCO, zapewnia tzw. leniwe ładowanie. Aby uzyskać DP klasa POCO musi spełniać następujące założenia:
  - klasa POCO musi być zadeklarowana jak klasa o publicznym dostępie,
  - klasa POCO nie może być opatrzona słowem sealed (lub NotInheritable w przypadku języka Visual Basic),
  - klasa POCO nie może być abstrakcyjna (MustInherit dla Visual Basic),
  - ...

Notatki

Typy encji – 2/2

EF oferuje dwa typu/rodzaj encji „POCO" oraz „Dynamic Proxy".

- Encje POCO – Plain Old CLR Object – wykorzystane są w zapytaniach typu: select, insert, update, delete. Są one generowane przez Entity Data Model i wspierane przez EF 6 oraz EF Core,
- Dynamic Proxy (DP) to klasa opakowująca encje POCO, zapewnia tzw. leniwe ładowanie. Aby uzyskać DP klasa POCO musi spełniać następujące założenia:
  - ...
  - każda własność nawigacyjna musi być deklarowana jako publiczna oraz wirtualna,
  - każda własność nawigacyjna kolekcyjna musi być typu `ICollection<T>`,
  - opcja ProxyCreationEnabled nie może być ustawiona na fałsz (domyślną wartością jest true) w klasie kontekstu.

Domyślne DP jest aktywne dla każdej encji, aby wyłączyć to domyślne zachowanie wystarczy skorzystać z klasy kontekstu:

```
context.Configuration.ProxyCreationEnabled = false;
```

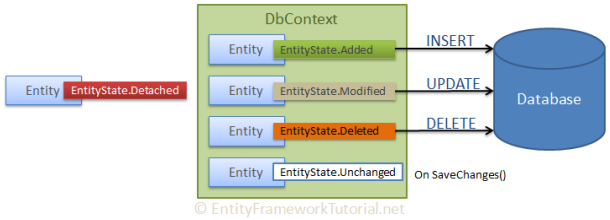

Notatki

Stany encji

Encje mają pole o typie System.Data.Entity.EntityState (Microsoft.EntityFrameworkCore.EntityState dla EF Core) z następującymi wartościami:

- Added, Modified,
- Deleted, Unchanged,
- Detached.

Obiekt kontekstu oprócz przechowywania referencji do obiektów encji odczytanych z bazy danych, przechowuje także informacje o zmianie danych w encji.



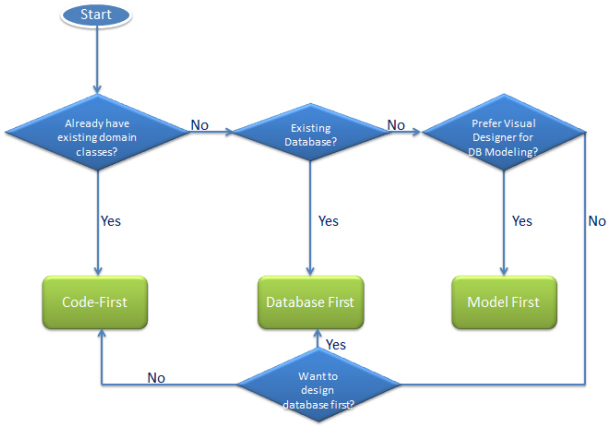
Jeśli, encja znajduje się w stanie Detached, zmiany nie są przenoszone do systemu bazy danych.

Notatki

Trzy podejścia EF

Tworzenie aplikacji EF można zrealizować za pomocą trzech głównych podejść:

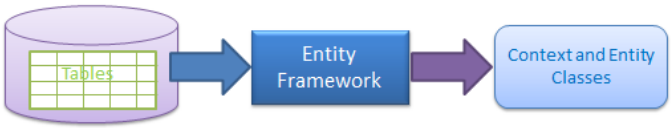
- „Database-First”,
- „Code-First”, „Model-First”.



Notatki

Podejście „DB-first”

W podejściu „Database-First”, tworzony jest kontekst oraz encje dla istniejącej bazy danych z zastosowaniem narzędzia EDM zintegrowanego ze środowiskiem Visual Studio. Możliwe jest również korzystanie z innych narzędzi dostępnych dla EF.



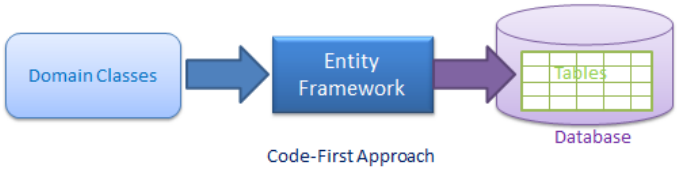
Database-First Approach

EF oferuje dość szerokie wsparcie dla podejścia DB-first, szczególnie w kontekście bazy danych MS SQL.

Notatki

Podejście „Code-first"

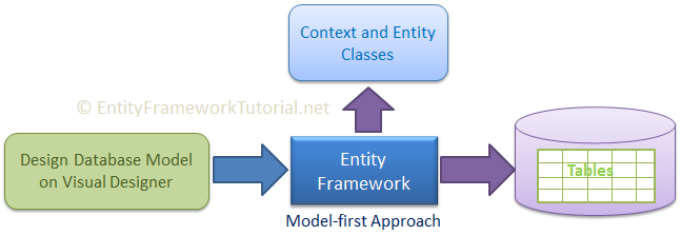
Podejście „code-first" powinno być stosowane w momencie kiedy nie posiadamy utworzonej bazy danych w aplikacji. W tym podejściu, rozpoczynamy proces pracy z danymi poprzez tworzenie encji (klas domen) i klasy kontekstu danych. Następnie na podstawie utworzonego kodu z pomocą narzędzi do migracji tworzona jest baza danych.



Notatki

Podejście „Model-first"

W tym podejściu, tworzenie encji, relacji, hierarchii dziedziczenia następuje w odpowiednim narzędziu projektowania wizualnego. Na podstawie schematu/diagramu, następuje generowanie pozostałych elementów takich jak encje, klasa kontekstu oraz skrypt związany z bazą danych.

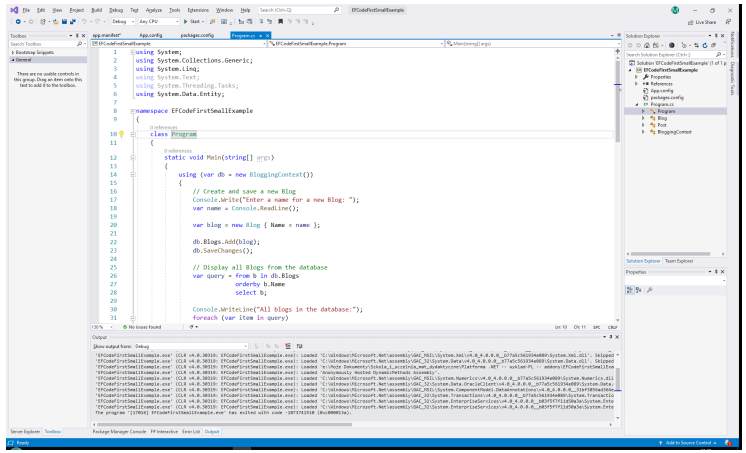


Notatki

Część praktyczna

Notatki

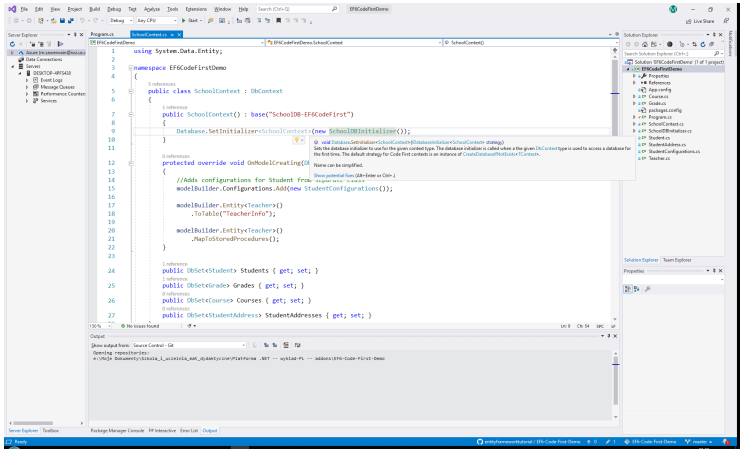
Operacje na prostym blogu:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;

namespace EFCodeFirstSmallExample
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new BloggingContext())
            {
                // Create and save a new Blog
                Console.Write("Enter a name for a new Blog: ");
                var name = Console.ReadLine();

                var blog = new Blog { Name = name };

                db.Blogs.Add(blog);
                db.SaveChanges();

                // Display all Blogs from the database
                var query = from b in db.Blogs
                            orderby b.Name
                            select b;

                Console.WriteLine("All blogs in the database:");
                foreach (var item in query)
```

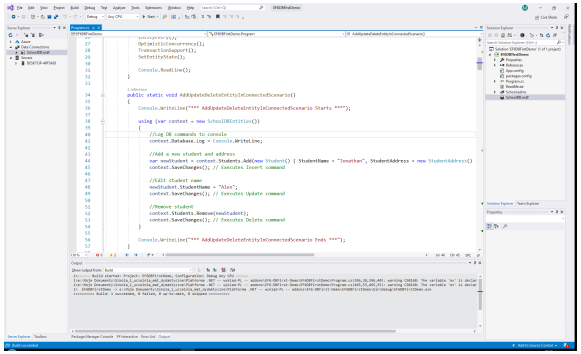

Notatki

Podejście „Code-First”:



Notatki

Podejście „DB-First”:



Notatki

W następnym tygodniu między innymi

1. czym jest XML,
2. obsługa XML w .NET,
3. przestrzeń nazw System.XML,
4. XML DOM API,
5. XPath oraz XSL,
6. LINQ oraz XML.

Proponowane tematy prac pisemnych:

1. porównanie modelu połączeniowego i bezpołączeniowego,
2. przedstawić dostępne techniki modyfikacji danych w trybie bezpołączeniowym,
3. zalety stosowania LINQ,
4. wydajność dla zapytań równoległych i zwykłych zapytań LINQ,
5. różnice i podobieństwa pomiędzy EF a NHibernate,
6. ogólne omówienie zagadnień dot. ORM,
7. wydajność podejścia EF do przetwarzania danych.

Dziękuje za uwagę!!!

Notatki

Notatki