Edukativni primer generisanja i obrade podataka uz alate dostupne u .NET 5, u domenu Metrologije

Ivan Gutai, Prof. dr Platon Sovilj, Marina Subotin,
Đorđe Novaković, dr Nemanja Gazivoda, dr Bojan Vujičić

Septembar 2021.

Spisak korišćenih (Microsoft) tehnologija

- Microsoft SQL Server (MSSQL), Express verzija, T-SQL upiti
- Entity Framework Core (U okviru .NET i C#)
- LINQ (Language-Integrated Query), C# upiti
- C# u okviru .NET-a
- xUnit, alat za testiranje

Uvod

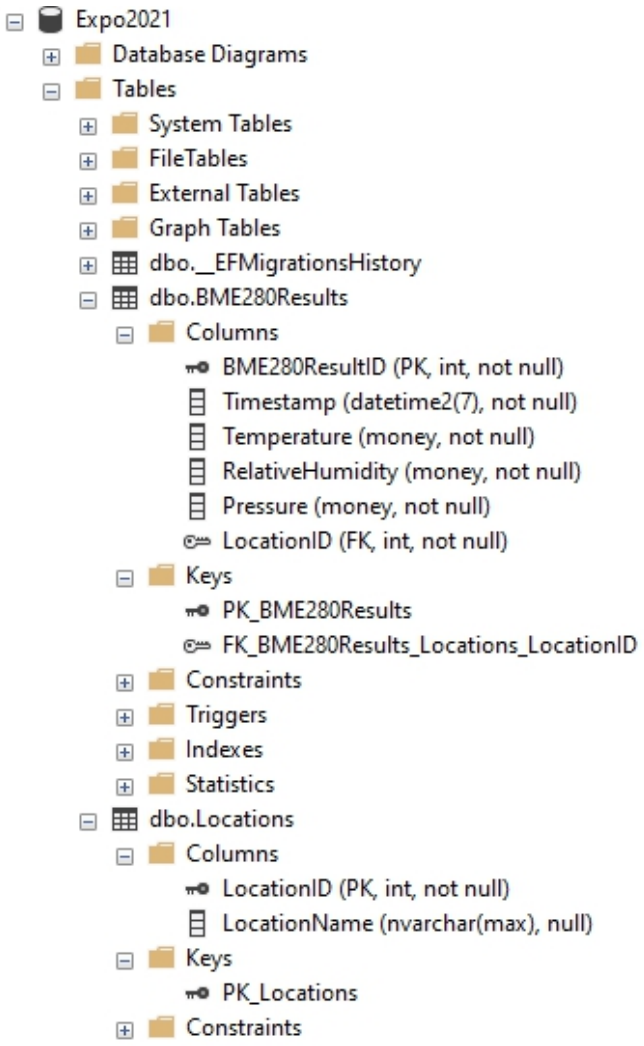
U ovom radu je akcenat stavljen na upotrebu Micorosoft-ovih tehnologija za projektovanje, razvoj i testiranje desktop aplikacije, kao i baze podataka.

Podaci se skladište na računaru i pristupamo im pisanjem SQL upita ili programski, upotrebom C# (LINQ upita). Svi rezultati mogu biti filtrirani, sortirani ili po potrebi izmenjeni.

Uz napomenu, da istovremeno pisanje LINQ upita i SQL upita za manipulaciju podacima iz baze, predstavlja na neki način verifikaciju LINQ upita.

Prikaz baze podataka

MSSQL



Baza podataka ima naziv "Expo2021". A u njoj se nalaze 3 tabele: BME280Results, Locations i __EFMigrationsHistory, koja služi za praćenje izmena, a "EF" je skraćeno od "Entity Framework". Tabela Locations ima kolone LocationID i LocationName i stavke 1 i 2 za Office i Balcony, respektivno. Mala i praktična optimizacija, da se vrednosti koje se ponavljaju, ne bi iznova upisivale u BME280Results tabelu. BME280ResultID, Timestamp, Temperature, RelativeHumidity, Pressure, LocationID predstavljaju nazive kolona u BME280Results tabeli. Ključevi u tabelama su sledeći: BME280Results tabela, PK_BME280Results i FK_BME280Results_Locations_LocationID, Locations tabela, PK_Locations.

Ukratko, tabela BME280Results ima sekundarni ključ u tabeli Locations, a vezani su sa LocationID-jem. Primarni ključevi u navedenim tabelama su njihove ID kolone. PK i FK su skraćenice od "primary" i "foreign" key, respektivno.

Prikaz rezultata iz više tabela, po različitim kriterijumima

MSSQL

Results | Messages

	BME280ResultID	Timestamp	Temperature	RelativeHumidity	Pressure	LocationName
1	1	2021-03-26 13:57:30.6166667	26,5833	40,5452	1012,2627	Balcony
2	2	2021-03-26 13:57:30.6166667	24,1285	52,1342	1017,6889	Balcony

	BME280ResultID	Timestamp	Temperature	RelativeHumidity	Pressure	LocationName
1	1000000	2021-03-26 14:10:41.1300000	26,0263	55,4484	1011,4289	Office
2	999999	2021-03-26 14:10:41.1300000	28,555	44,8757	1029,2481	Office

	LocationName	Items
1	Balcony	499152
2	Office	500848

	Temperature	NoOfItems
1	29,4203	26
2	29,0617	26

```
SELECT TOP (2) B.BME280ResultID, B.Timestamp,
B.Temperature, B.RelativeHumidity, B.Pressure,
L.LocationName
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
ORDER BY BME280ResultID ASC
```

```
SELECT TOP (2) B.BME280ResultID, B.Timestamp,
B.Temperature, B.RelativeHumidity, B.Pressure,
L.LocationName
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
ORDER BY BME280ResultID DESC
```

```
SELECT L.LocationName, COUNT(*) AS Items
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
GROUP BY L.LocationName;
```

```
SELECT TOP (2) B.Temperature, COUNT(*) AS NoOfItems
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
GROUP BY B.Temperature
ORDER BY NoOfItems DESC
```

- ✓ Prikaz prva dva zapisa u bazi
- ✓ Prikaz poslednja dva zapisa u bazi
- ✓ Brojni prikaz stavki, grupisan po lokaciji
- ✓ Prikaz modusa za izabranu veličinu

Opcija za kreiranje tabela, pisanjem T-SQL upita

MSSQL

```
USE [Expo2021]
GO

/****** Object:  Table [dbo].[BME280Results]    Script Date: 26.3.2021. 13:16:43 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[BME280Results](
                        [BME280ResultID] [int] IDENTITY(1,1) NOT NULL,
                        [Timestamp] [datetime2](7) NOT NULL,
                        [Temperature] [money] NOT NULL,
                        [RelativeHumidity] [money] NOT NULL,
                        [Pressure] [money] NOT NULL,
                        [LocationID] [int] NOT NULL,
 CONSTRAINT [PK_BME280Results] PRIMARY KEY CLUSTERED
(
                        [BME280ResultID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[BME280Results] ADD  DEFAULT (getdate()) FOR [Timestamp]
GO

ALTER TABLE [dbo].[BME280Results]  WITH CHECK ADD  CONSTRAINT [FK_BME280Results_Locations_LocationID] FOREIGN KEY([LocationID])
REFERENCES [dbo].[Locations] ([LocationID])
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[BME280Results] CHECK CONSTRAINT [FK_BME280Results_Locations_LocationID]
GO
```

Napomena: Nema potrebe na ovaj način kreirati tabele, pošto postoji alternativni (mnogo lakši) način, da se sve to uradi, preko C# koda.

Detalji korišćenih T-SQL upita

MSSQL

```sql
SELECT TOP (2) B.BME280ResultID, B.Timestamp,
B.RelativeHumidity, L.LocationName
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
ORDER BY BME280ResultID DESC

SELECT L.LocationName, COUNT(*) AS Items
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
GROUP BY L.LocationName;

SELECT TOP (2) B.RelativeHumidity, COUNT(*) AS NoOfItems
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
GROUP BY B.RelativeHumidity
ORDER BY NoOfItems DESC

-- TRUNCATE TABLE [Expo2021].[dbo].[BME280Results]

/*
    DELETE FROM [Expo2021].[dbo].[BME280Results]
    WHERE BME280ResultID > 1000000
*/
```

- ✓ Selektovanje i kombinovanje rezultata iz dve tabele, od kojih je jedna šifarnik.
- ✓ Selektovanje i grupisanje rezultata u odnosu na lokacije sa kojih su preuzeti podaci.
- ✓ Prikaz modusa iz određene kolone, npr. RelativeHumidity.
- ✓ Funkcija TRUNCATE, je funkcija koju treba koristiti isključivo prilikom testiranja i razvoja. Ova funkcija momentalno briše sve stavke i brojač ID-a vraća na inicijalnu vrednost. Npr. ako se obrišu sve stavke na ovaj način, prvi sledeći ID koji će biti dodeljen je podrazumevani, u ovom slučaju 1.
- ✓ Funkcija DELETE briše stavke po određenom kriterijumu, a njenom upotrebom se ne gubi zapis o ID-ju stavki. Npr. ako se obrišu sve stavke na ovaj način, prvi sledeći ID koji će biti dodeljen je 1000001 (u slučaju da je najveći ID, pre brisanja bio 1000000).

Kreiranje SQL tabela programski, code first pristup

Entity Framework Core

```
public class Location
{
    [Key]
    public int LocationID { get; set; }
    public string LocationName { get; set; }
    public virtual ICollection<BME280Result> BME280Results { get; set; }
    public Location()
    {
        this.BME280Results = new List<BME280Result>();
    }
    public class BME280Result
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int? BME280ResultID { get; set; }
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public DateTime Timestamp { get; set; }
        [Column(TypeName = "money")]
        public decimal Temperature { get; set; }
        [Column(TypeName = "money")]
        public decimal RelativeHumidity { get; set; }
        [Column(TypeName = "money")]
        public decimal Pressure { get; set; }
        public int LocationID { get; set; }
        public virtual Location Location { get; set; }
    }
}
```

- ✓ Prikazan je C# kod koji definiše imena tabela, nazive i tipove kolona koji su kreirani u bazi, a "[Column(TypeName = "money")]" predstavlja jedan od atributa anotacije u Entity Framework Core-u.
- ✓ Kreiranje baze iz C# koda se naziva code first pristup.
- ✓ Nakon pisanja C# koda, iz Package Manager Console je potrebno pokrenuti komande: "Add-Migration" sa nazivom npr. "Initial" i "Update-Database". Na navedeni način će biti kreirana prethodno prikazana struktura u SQL bazi podataka.
- ✓ Vrlo je praktično za dalji rad, modifikaciju postojećih tabela i dodavanje novih. Uvek postoji opcija vraćanja nekoliko koraka nazad, migraciju po migraciju.

Detalji korišćenih LINQ upita

LINQ (Language-Integrated Query)

```
public static decimal? Median(this IEnumerable<decimal?> sequence)
{
    int numberCount = sequence.Count();
    int halfIndex = sequence.Count() / 2;
    var sortedNumbers = sequence.OrderBy(n => n);
    decimal median;
    if ((numberCount % 2) == 0)
    {
        return median = Convert.ToDecimal(((sortedNumbers.ElementAt(halfIndex)
            + sortedNumbers.ElementAt((halfIndex - 1))) / 2));
    }
    else
    {
        return median = Convert.ToDecimal(sortedNumbers.ElementAt(halfIndex));
    }
}

public static decimal? Median<T>(this IEnumerable<T> sequence, Func<T, decimal?> selector)
{
    return sequence.Select(selector).Median();
}

public static decimal? Mode(this IEnumerable<decimal?> sequence)
{
    var mode = sequence.GroupBy(n => n)
        .OrderByDescending(g => g.Count()).Select(g => g.Key).FirstOrDefault();
    return mode;
}

public static decimal? Mode<T>(this IEnumerable<T> sequence, Func<T, decimal?> selector)
{
    return sequence.Select(selector).Mode();
}
```

- ✓ Osnovna ideja matematičke statistike jeste da se istraživanja sprovedu na uzorku i da se na osnovu njih donesu zaključci koji se proširuju na populaciju.
- ✓ Statistika koju poseduje LINQ je srednja vrednost (average), a statistike koje su dodate jesu medijana (median) i modus (mean).
- ✓ Na primeru modusa se može zaključiti, da se LINQ upit razlikuje od T-SQL upita isključivo u sintaksi. LINQ upit se piše u .NET podržanom jeziku, npr. C#, a SQL upit se piše jezikom T-SQL.

Generisanje zapisa u bazi podataka

C#

```
public void GenerateNItems(int NoOfItems)
{
    using (var db = new DatabaseContext())
    {
        for (int i = 1; i < NoOfItems + 1; i++)
        {
            var newEntry = new BME280Result();
            //newEntry.BME280ResultID = i;
            newEntry.Temperature = GetRandomNumberInRange(23, 32);
            newEntry.RelativeHumidity = GetRandomNumberInRange(40, 60);
            newEntry.Pressure = GetRandomNumberInRange(1010, 1030);
            newEntry.LocationID = (int)GetRandomNumberInRange(1, 3);
            db.BME280Results.Add(newEntry);

            if (i % 10000 == 0)
            {
                var count = db.SaveChanges();
                Console.WriteLine("{0} records saved to database", count);
            }
            else if (i == NoOfItems)
            {
                var count = db.SaveChanges();
                Console.WriteLine("{0} records saved to database", count);

            }
            //count = db.SaveChanges();
        }
        //Console.WriteLine("{0} records saved to database", count);
    }
}
```

```
public void LoadNItems(int NoOfItems)
{
    using (var db = new DatabaseContext())
    {
        Console.WriteLine("{0,16} | {1,-27} | {2,13:N2} | {3,18:N2}" +
            " | {4,10:N2} | {5,14:N2} |", "[BME280ResultID]",
            "[Timestamp]", "[Temperature]", "[RelativeHumidity]",
            "[Pressure]", "[LocationName]");

        var data = db.BME280Results.Join(
        inner: db.Locations,
        outerKeySelector: bm -> bm.LocationID,
        innerKeySelector: lo => lo.LocationID,
        resultSelector: (b, l) ->
        new { b.BME280ResultID, b.Timestamp, b.Temperature,
            b.RelativeHumidity, b.Pressure, l.LocationName }).Take(NoOfItems);

        foreach (var item in data)
        {
            Console.WriteLine("{0,16} | {1,27:dd.MM.yyyy. H:mm:ss zzz}" +
                " | {2,-13:N2} | {3,-18:N2} | {4,-10:N2} | {5,14} |"
                , item.BME280ResultID, item.Timestamp, item.Temperature,
                item.RelativeHumidity, item.Pressure, item.LocationName);
        }

    }
}
```

- ✓ *Funkcija za generisanje podataka*
- ✓ "using" se koristi zbog "unmanaged" resursa, tj. nakon završenog korišćenja SQL konekcije, konekcija se zatvara.

- ✓ *Funkcija za učitavanje podataka*
- ✓ Na isti način kao što bi se radio JOIN tabela sa T-SQL upitima u bazi, tako se u C# radi JOIN podataka sa LINQ sintaksom.

Eksportovanje podataka iz SQL baze u .json fajl

C#

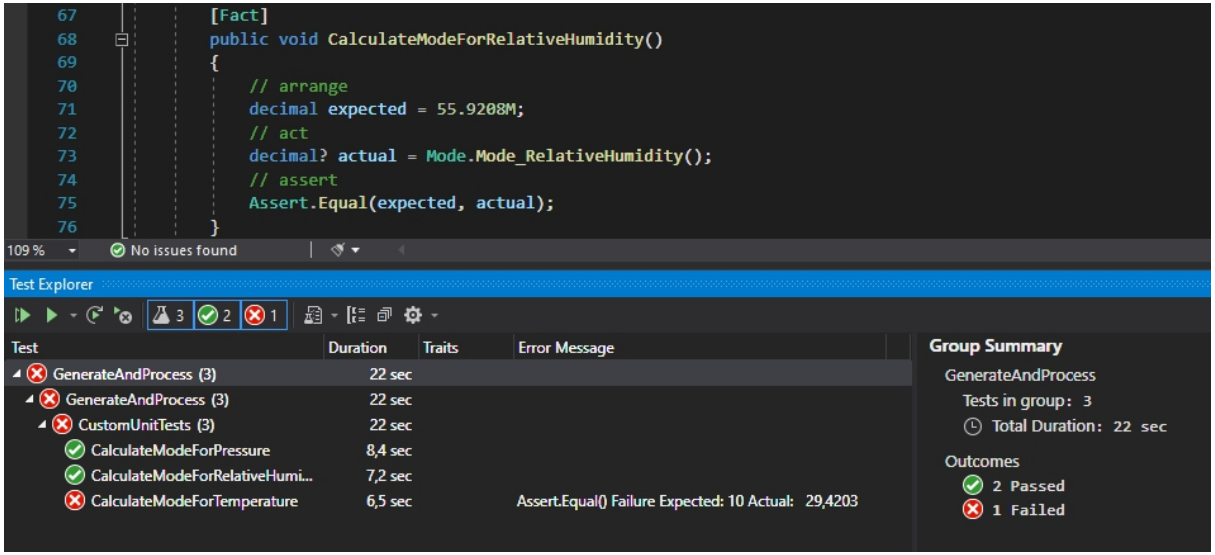
```
public IQueryable GenerateJSON(int NoOfItems)
{
    using (var db = new DatabaseContext())
    {
        var data = db.BME280Results.Join(
        inner: db.Locations,
        outerKeySelector: bm => bm.LocationID,
        innerKeySelector: lo => lo.LocationID,
        resultSelector: (b, l) =>
        new { b.BME280ResultID, b.Timestamp, b.Temperature, b.RelativeHumidity,
            b.Pressure, l.LocationName }).Take(NoOfItems).OrderByDescending(b => b.BME280ResultID);

        var results = data.ToList();

        JObject json =
            new JObject(
                new JProperty("id", "MillenialDIY2020LE"),
                new JProperty("BME280",
                    new JArray(
                            from p in results
                            orderby p.Timestamp
                            select new JObject(
                            new JProperty("BME280ResultID", p.BME280ResultID),
                            new JProperty("Timestamp", p.Timestamp),
                            new JProperty("Temperature", p.Temperature),
                            new JProperty("RelativeHumidity", p.RelativeHumidity),
                            new JProperty("Location", p.LocationName),
                            new JProperty("Pressure", p.Pressure)))));

        File.WriteAllText(@"c:\results.json", JsonConvert.SerializeObject(json));

        // serialize JSON directly to a file
        using (StreamWriter file = File.CreateText(@"c:\results.json"))
        {
            JsonSerializer serializer = new JsonSerializer();
            serializer.Serialize(file, json);
        }

        return data;
    }
}
```

- ✓ Eksportovanje podataka iz SQL baze u .json fajl se vrši funkcijom koja omogućava serijalizaciju podataka u zadatom obliku.
- ✓ Prikazani su rezultati sa dve stavke, pošto nije baš jednostavno ni na modernom PC-ju otvoriti .json fajl od 152 MB, sa svih 1000000 zapisa.


```
{
    "id": "MillenialDIY2020LE",
    "BME280": [
        {
            "BME280ResultID": 1000001,
            "Timestamp": "2021-03-30T10:41:23.0433333",
            "Temperature": 28.9234,
            "RelativeHumidity": 57.7886,
            "Location": "Office",
            "Pressure": 1020.5552
        },
        {
            "BME280ResultID": 1000002,
            "Timestamp": "2021-03-30T10:41:23.0833333",
            "Temperature": 27.1621,
            "RelativeHumidity": 47.7860,
            "Location": "Office",
            "Pressure": 1019.2326
        }
    ]
}
```

TDD (Test driven development) i AAA princip

xUnit



- ✓ TDD (Test driven development) i Unit testovi su postali deo žargona u IT industriji. Najprostiji opis je pisanje koda koji testira kod i daje rezultate o njegovoj ispravnosti.
- ✓ Zasniva se na "AAA principu", tj. "Arrange", "Act" i "Assert", kako god se test framework zvao, a u ovom slučaju je to xUnit.
- ✓ U delu "Arrange" pišemo pretpostavku, u "Act" testiramo funkciju, a u zavisnosti od rezultata, u "Assert"-u je određeno da li je test uspešan ili ne.
- ✓ Interesantno je da ukoliko test "padne", opisano je zbog čega se to desilo. TDD je praktičan na velikim projektima i olakšava regresione testove, zato što automatski proverava da li je nova funkcionalnost narušila neku staru.
- ✓ Testove treba pisati tako da testiraju stvarne funkcionalnosti, a ne da povećavamo broj testova koji uspešno prolaze.
- ✓ U xUnit-u test je označen sa "[Fact]".

Hvala na pažnji,

kompletan kod je dostupan na:

https://github.com/IvanGutai/publications