Microsof SQL

Kód

Získat informace

- ▶ Informace ze Serveru
- ▶ Informace z Tabulky
- ▶ Velikost Tabulek
- ▶ Informace o indexech na sloupcích

Hledat

- ▶ Sloupec a zjistit v jaké Tabulce se nachází
- ▶ Datový typ Sloupce z Tabulky
- ▶ Hodnoty ve všech textových a číselných sloupcích databáze
- ▶ Nejnovější a Nejstarší záznam
- ▶ Nejčastěji se vyskytující hodnoty
- ▶ Port na kterém je spuštěn Server

Výkon

- ▶ Efektivita dotazů

Konfigurace

- ▶ Vzdálený přístup

Co je MongoDB

= Dokumentová databáze

- NoSQL (typ databáze)

> **Note**
>
> Nevyužívá tabulkový formát, který je běžný u SQL databází.

- Data ukládá ve formátu zvaném `BSON`. (Binární verze formátu `JSON`)

> **Tip**
>
> Formát `BSON` podporuje více datových typů.
>
> Efektivnější při encoding a decoding než `JSON`.

Collections

- Data jsou organizována do `collections`

> **Note**
>
> Jsou ekvivalentem tabulek v SQL.

- Každá tato kolekce obsahuje `documents`, což jsou jednotlivé záznamy dat.

Note

Na rozdíl od řádků v tabulce SQL nemusí mít dokumenty v MongoDB stejnou strukturu.

To znamená, že různé dokumenty ve stejné kolekci mohou mít různé sady polí.

Příklad:

```
var document1 = { name: "Peter", age: 30, residence: "Prague" };
db.myPeople.insert(document1);

var document2 = { name: "Anna", age: 25, occupation: "Engineer" };
db.myPeople.insert(document2);
```

Klíčové pojmy

- Dokumenty
- Kolekce
- BSON

Kód

Vytvořit

- Databázi
- Kolekci
- Vložit dokument do kolekce
- Vložit více dokumentů do kolekce
- Vytvořit index
- Vytvoření více indexů

Hledat

- Výpis databází
- Výpis dokumentů
- Výpis kolekcí
- Hledání dokumentu
- Hledání dokumentu s určitými poli

▶ Hledání dokumentu s regulárním výrazem

Aktualizovat

▶ Aktualizace dokumentu
▶ Aktualizace více dokumentů
▶ Aktualizace dokumentu s upsert

Smazat

▶ Smazání databáze
▶ Smazání kolekce
▶ Smazání dokumentu
▶ Smazání všech dokumentů

Počet

▶ Počet dokumentů v kolekci
▶ Počet dokumentů odpovídajících určitému dotazu
▶ Počet unikátních hodnot v určitém poli
▶ Počet dokumentů odpovídajících regulárnímu výrazu

Setřídit

▶ Seřazení dokumentů podle pole
▶ Seřazení dokumentů podle více polí
▶ Seřazení a omezení počtu dokumentů
▶ Seřazení a přeskočení dokumentů

Rady a Tipy

Povolení autorizace

- MongoDB má vestavěný systém pro správu uživatelů a rolí.

  Pro povolení autorizace upravte konfigurační soubor MongoDB a nastavte `security.authorization` na `enabled`.

```
security:
  authorization: "enabled"
```

Využití indexů

- Indexy v MongoDB vytváříte pomocí metody `createIndex()`.

  Například pro vytvoření vzestupného indexu na pole `name` v kolekci `mycollection` použijete následující příkaz:

```
db.mycollection.createIndex({name: 1})
```

Optimalizace dotazů

- MongoDB poskytuje operátor `explain()`, který vám umožní zjistit, jak databáze vykonává váš dotaz.

  Tímto způsobem můžete identifikovat, které části dotazu je třeba optimalizovat.

```
db.mycollection.find({name: 'test'}).explain()
```

Správné modelování dat

- MongoDB je dokumentová databáze, která umožňuje velmi flexibilní modelování dat.

  Při návrhu vašeho datového modelu zvažte, jak budou data dotazována a jaké budou pracovní zátěže.

Škálování

- MongoDB podporuje horizontální škálování pomocí replikačních sad a sharding.

  Pro větší aplikace zvažte použití těchto funkcí pro zlepšení výkonu a dostupnosti.

Paměť

- MongoDB využívá paměť pro ukládání pracovní sady, což zlepšuje výkon dotazů.

  Ujistěte se, že váš server má dostatek RAM pro vaše pracovní zátěže.

Šetření prostředky

- Pokud máte dotazy, které se často opakují, zvažte ukládání výsledků těchto dotazů pro pozdější použití.

To může šetřit výpočetní prostředky a zlepšit výkon vaší aplikace.

PostgreSQL

Podporuje programovací jazyky:

- Python
- Java
- C/C++
- C#
- Node.js
- Go
- Ruby
- Perl
- Tcl

PostgreSQL podporuje v podstatě všechny funkce, které podporují jiné systémy pro správu databází.

▶ Uživatelské rozhraní

Instalace

▶ Výběr verze produktu
▶ Spustit instalaci
▶ Složka pro instalaci
▶ Výběr komponent
▶ Složka pro uložení dat databáze
▶ Nastavit heslo
▶ Port k naslouchání
▶ Geografické umístění serveru
▶ Kontrola před provedením

Příkazový řádek

▶ Otestovat zda PostgreSQL naslouchá

Entity Framework

Kdy použít

> **Note**
>
> Pro rychlý vývoj aplikací s menšími nároky na výkon a větší komplexitou modelů.

- Výkon: Nižší výkon kvůli režii ORM (Object Relation Mapping)
- Snadnost vývoje: Rychlý vývoj s minimálním SQL
- Komplexní modely: Automatická správa modelů a migrací
- Flexibilita dotazů: Omezenější – závisí na EF generátoru

Instalace

```
"C:\Program Files\dotnet\dotnet.exe" tool install --ignore-failed-sources --global dotnet-ef
```

> **Note**
>
> Balíček bude uložen ve složce: `C:\Users\<TvéUživatelskéJméno>\.dotnet\tools`
>
> Pro zálohu offline, zkopírujte obsah této složky na jiný počítač, kde nástroj `dotnet-ef` nebude dostupný online.

> **Warning**
>
> Pokud složku umístíte na jinou cestu, ujistěte se, že ji přidáte do proměnných do `PATH`, aby byl nástroj dostupný z příkazového řádku.

1. Spusťte build pro zobrazení chyb

```
dotnet build
```

2. Vytvořte první migraci

> **Important**
>
> Ujistěte se, že se nacházíte ve složce, kde se nachází váš `.csproj` soubor

```
dotnet ef migrations add InitialCreate
```

3. Aktualizujte databázi pomocí migrace

```
dotnet ef database update
```

Použití

1. Instalace NuGet balíčku:

```
dotnet add package Microsoft.EntityFrameworkCore
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

2. Konfigurace a použití:

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

// Model entity
public class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}

// DbContext pro správu databáze
public class AppDbContext : DbContext
{
    public DbSet<User> Users { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
```

```
    {
        optionsBuilder.UseSqlServer("Server=myServer;Database=myDatabase;User
Id=myUser;Password=myPassword;");
    }
}

// Služba pro práci s uživateli
public class UserService
{
    private readonly AppDbContext _dbContext;

    public UserService(AppDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public async Task ShowUsersAsync()
    {
        var users = await _dbContext.Users
            .Where(u => u.Age > 18)
            .ToListAsync();

        foreach (var user in users)
        {
            Console.WriteLine($"ID: {user.Id}, Name: {user.Name},
Age: {user.Age}");
        }
    }
}

// Hlavní program
class Program
{
    static async Task Main()
    {
        using var dbContext = new AppDbContext();
        var userService = new UserService(dbContext);

        await userService.ShowUsersAsync();
    }
}
```

Příkazy

Příkaz	Popis
`dotnet ef migrations add <Název>`	Vytvoří nový soubor pro migraci s názvem `<Název>`, který zachytí změny ve tvých modelech (entitách).
`dotnet ef migrations remove`	Smaže poslední migraci, kterou jsi přidal, ale nezmění databázi (pouze vrátí kód zpět).
`dotnet ef migrations list`	Zobrazí seznam všech migrací, které jsi vytvořil (ukazuje, jaké změny se postupně prováděly).
`dotnet ef database update`	Aplikuje všechny migrace (změny) na databázi, aby se databáze aktualizovala podle aktuálních modelů.
`dotnet ef database update <Název>`	Aplikuje migraci s názvem `<Název>` (pokud nechceš aplikovat všechny migrace).
`dotnet ef database drop`	Smaže celou databázi – dávej pozor, tímto příkazem přijdeš o všechna data.
`dotnet ef dbcontext list`	Ukáže všechny třídy DbContext ve tvém projektu (DbContext je hlavní třída pro práci s databází).
`dotnet ef dbcontext info`	Zobrazí informace o tvojí DbContext třídě (užitečné pro zjištění detailů o konfiguraci).
`dotnet ef dbcontext scaffold`	Vytvoří třídy (modely) podle existující databáze – tímto způsobem můžeš začít, pokud už máš databázi.
`dotnet ef migrations script`	Vygeneruje SQL skript, který obsahuje všechny změny v migracích – vhodné pro manuální nasazení.

Dapper

Kdy použít

> **Note**
>
> Pro projekty, kde je klíčový výkon nebo kontrola nad databází.

- Výkon: Maximální výkon, nízká režie
- Snadnost vývoje: Ruční psaní SQL, více práce
- Komplexní modely: Ruční správa modelů
- Flexibilita dotazů: Vysoká – plná kontrola nad SQL

Použití

1. Instalace NuGet balíčku:

```
dotnet add package Dapper
```

2. Konfigurace a použití:

```
using System;
using System.Data.SqlClient;
using System.Threading.Tasks;
using Dapper;

// Příklad implementace v aplikační vrstvě
public class UserRepository
{
    private readonly string _connectionString;

    public UserRepository(string connectionString)
    {
        _connectionString = connectionString;
    }

    // Metoda na získání uživatelů starších než zadaný věk
    public async Task<IEnumerable<User>> GetUsersOlderThanAsync(int age)
    {
        const string sql = "SELECT Id, Name, Age FROM Users WHERE Age > @Age";
```

```
        using (var connection = new SqlConnection(_connectionString))
        {
            return await connection.QueryAsync<User>(sql, new { Age = age });
        }
    }
}

// Model entity
public class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}

// Použití repository ve službě
public class UserService
{
    private readonly UserRepository _repository;

    public UserService(UserRepository repository)
    {
        _repository = repository;
    }

    public async Task ShowUsersAsync()
    {
        var users = await _repository.GetUsersOlderThanAsync(18);
        foreach (var user in users)
        {
            Console.WriteLine($"ID: {user.Id}, Name: {user.Name},
Age: {user.Age}");
        }
    }
}

// Hlavní program
class Program
{
    static async Task Main()
    {
        var connectionString = "Server=myServer;Database=myDatabase;User
Id=myUser;Password=myPassword;";
        var userRepository = new UserRepository(connectionString);
        var userService = new UserService(userRepository);
```

```
        await userService.ShowUsersAsync();
    }
}
```

Note

Dapper je v tomo příkladu použit v následující části kódu:

```
using (var connection = new SqlConnection(_connectionString))
{
    return await connection.QueryAsync<User>(sql, new { Age = age });
}
```