🗂️ Microsoft SQL – Praktické dotazy & tipy

> 🚀 Moderní přehled SQL dotazů, postupů a tipů pro práci s Microsoft SQL Serverem.

🔎 Získání informací o serveru a databázi

- ▶ 🖥️ Informace ze serveru
- ▶ 📋 Informace z tabulky
- ▶ 📏 Velikost tabulek
- ▶ 🗂️ Informace o indexech na sloupcích

🔍 Hledání v databázi

- ▶ 🔎 Najít tabulku podle sloupce
- ▶ 🔢 Datový typ sloupce z tabulky
- ▶ 🔍 Hledat hodnotu ve všech textových a číselných sloupcích
- ▶ ⏳ Nejnovější a nejstarší záznam
- ▶ 📊 Nejčastěji se vyskytující hodnoty
- ▶ 🔌 Port na kterém je spuštěn server

⚡ Výkon a statistiky

- ▶ 📈 Efektivita dotazů

⚙️ Konfigurace

- ▶ 🌐 Vzdálený přístup

🔗 Spojování tabulek

- ▶ ⬅️ LEFT JOIN
- ▶ ➡️ RIGHT JOIN
- ▶ 🔄 FULL OUTER JOIN
- ▶ 🔗 INNER JOIN
- ▶ 🔍 CROSS APPLY

🧵 Spojování hodnot ze sloupce do řetězce

▶ 🔗 STRING_AGG

🗂️ MongoDB – Praktický průvodce & tipy

> 🚀 Moderní přehled základních pojmů, příkazů a doporučení pro práci s MongoDB.

📖 Co je MongoDB?

- **Dokumentová databáze**
- **NoSQL** – nevyužívá tabulkový formát jako SQL databáze
- Data ukládá ve formátu **BSON** (binární JSON)

> ⓘ **Note**
>
> BSON podporuje více datových typů a je efektivnější při kódování/dekódování než JSON.

🧩 Klíčové pojmy

▶ 📄 Dokumenty
▶ 📚 Kolekce
▶ 🗃️ BSON

🛠️ Základní operace

Vytvoření

▶ 🗄️ Databáze
▶ 📁 Kolekce
▶ ➕ Vložit dokument
▶ ➕ Vložit více dokumentů
▶ 🔎 Vytvořit index
▶ 🔎 Vytvoření více indexů

Hledání

▶ 📋 Výpis databází
▶ 📄 Výpis dokumentů
▶ 📁 Výpis kolekcí

- 🔍 Hledání dokumentu
- 🔍 Hledání s určitými poli
- 🔍 Hledání s regulárním výrazem

Aktualizace

- ✏️ Aktualizace dokumentu
- ✏️ Aktualizace více dokumentů
- ✏️ Aktualizace s upsert

Smazání

- 🗑️ Smazání databáze
- 🗑️ Smazání kolekce
- 🗑️ Smazání dokumentu
- 🗑️ Smazání všech dokumentů

Počet

- 🔢 Počet dokumentů v kolekci
- 🔢 Počet dokumentů podle dotazu
- 🔢 Počet unikátních hodnot
- 🔢 Počet podle regulárního výrazu

Řazení

- ↕️ Řazení podle pole
- ↕️ Řazení podle více polí
- ↕️ Řazení a limit
- ↕️ Řazení a skip

🗂️ PostgreSQL – Praktický průvodce & tipy

> 🚀 Moderní přehled základních pojmů, instalace, příkazů a doporučení pro práci s PostgreSQL.

📖 Co je PostgreSQL?

- **Objektově-relační databázový systém**
- Podporuje širokou škálu programovacích jazyků:
  `C`, `C++`, `Java`, `Perl`, `Python`, `Ruby`, `Tcl`, `Scheme`, `PHP`, `Swift`, `Go`, `JavaScript`, `TypeScript`, `R`, `Rust`, `Kotlin`, `Lua`, `Erlang`, `Elixir`, `Haskell`, `Scala`, `D`, `Julia`, `PL/pgSQL`, `SQL`, `PL/Python`, `PL/Perl`, `PL/Tcl`, `PL/Java`, `PL/R`, `PL/V8`

> ⓘ **Note**
>
> `PL` znamená "Procedural Language" – umožňuje psaní funkcí a procedur přímo v databázi.

🖥️ Uživatelské rozhraní

▶ 🖥️ Otevření aplikace

🛠️ Instalace PostgreSQL

▶ 🔢 Výběr verze produktu
▶ ▶️ Spuštění instalace
▶ 📁 Složka pro instalaci
▶ 🧩 Výběr komponent
▶ 💾 Složka pro uložení dat
▶ 🔑 Nastavení hesla
▶ 🔌 Port k naslouchání
▶ 🌍 Geografické umístění serveru
▶ ✅ Kontrola před instalací

💻 Příkazový řádek

▶ 🔎 Otestovat naslouchání PostgreSQL

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 });
}
```