Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Porovnání nástrojů pro přístup k databázi na platformě .NET

Marcel Novák

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Marcel Novák

Informatika

Název práce

Porovnání nástrojů pro přístup k databázi na platformě .NET

Název anglicky

Database access tools comparison for .NET platform

Cíle práce

Hlavním cílem této bakalářské práce je analyzovat efektivnost různých nástrojů pro práci s databází na platformě .NET. Dílčími cíli jsou:
a) Charakteristika platformy .NET
b) Charakteristika jednotlivých nástrojů pro práci s databází
c) Vytvoření aplikace pro měření efektivnosti jednotlivých nástrojů
d) Na základě výsledků zhodnotit využití jednotlivých nástrojů ve vymezené oblasti

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. V praktické části bude navržena a implementována aplikace. Na základě vytvořené aplikace budou jednotlivé nástroje testovány v oblasti výkonu práce s databází. Dále bude sloužit k hodnocení efektivnosti kódu jednotlivých nástrojů ve vymezené oblasti. Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

45 stran

Klíčová slova

C#, Dapper, ADO.NET, EntityFramework, Framework, SQL, Databáze

---

Doporučené zdroje informací

FREEMAN, Adam. Pro ASP.NET MVC 5 platform. Berkeley, CA: Apress, 2014. Expert's voice in Web development. ISBN 1430265418

LACKO, Ľuboslav. Mistrovství v SQL Server 2012: [kompletní průvodce databázového experta]. Brno: Computer Press, 2013. ISBN 978-80-251-3773-4

LERMAN, Julia. Programming Entity framework. 2nd ed. Sebastopol: O´Reilly, c2010. ISBN 0596807260

RIORDAN, Rebecca M. Microsoft ADO.NET krok za krokem. Brno: Mobil Media, c2002. iDnes internet knihy. ISBN 8086593207

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Jan Masner, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 11. 9. 2018

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 10. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 11. 03. 2019

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Porovnání nástrojů pro přístup k databázi na platformě .NET" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 12.03.2019 _____

Marcel Novák

Poděkování

Rád bych touto cestou poděkoval Ing. Janu Masnerovi, Ph.D. za odborné vedení při tvorbě této bakalářské práce.

Porovnání nástrojů pro přístup k databázi na platformě .NET

Abstrakt

Tato bakalářská práce se zaměřuje na porovnání jednotlivých nástrojů pro přístup k databázi na platformě .NET. Práce se zabývá platformou .NET se zaměřením na nejnovější .NET Core, dále je v práci uveden způsob fungování databází, a především jsou představeny jednotlivé nástroje, konkrétně se jedná o Entity Framework, Entity Framework Core, Dapper a ADO.NET. Porovnávání nástrojů se provádí pro dvě modelové situace, které odpovídají praxi. Cílem je zvolit optimální nástroj pro každou situaci. Optimální nástroj se hledá pomocí vícekriteriální analýzy variant a vytvořeného programu sloužícího k měření výkonu jednotlivých nástrojů. Jednotlivé varianty, v tomto případě nástroje, mají zvolená kritéria, podle kterých se pro modelové situace vybírá ta optimální varianta. V závěru práce je shrnut výsledek a je doporučena optimální varianta vzhledem k modelovým situacím.

Klíčová slova: C#, Dapper, ADO.NET, Entity Framework, Framework, SQL, Databáze

Database access tools comparison for .NET platform

Abstract

This bachelor thesis is focused on comparison of single tools for access to database on platform .NET. This thesis deals with the .NET platform and is focused mainly on the latest .NET Core in the next part are explained relational databases and characterized single tools for access to database especially Entity Framework, Entity Framework Core, Dapper and ADO.NET. Comparison of tools is made for two model situations which correspond to practice. The goal is choose optimal tool for each of situation. Optimal tool is found by multi-criteria analysis and created program to measure the performance of single tools. Single variants, in this case tools, have criteria, which are used for choosing optimal variant for each of model situations. At the end of the thesis, the result is summarized, and the best option is recommended for each of the model situations.

Keywords: C#, Dapper, ADO.NET, Entity Framework, Framework, SQL, Database

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

1 Úvod

V současné době je kladen důraz na rychlost a efektivnost, a proto musí od prvotní myšlenky určitého projektu k jeho realizování uběhnout co nejkratší čas. Zároveň lidé nechtějí čekat, ať už při nákupu na internetových obchodech nebo jiných službách poskytovaných přes internet, minuty na načtení jednotlivých stránek. K jednotlivým službám jako jsou webové stránky, administrační systémy nebo jiné služby se využívají data v relačních databázích.

K relačním databázím se může přistupovat nepřeberným množstvím programovacích jazyků a jejich nástrojů. Každý z těchto nástrojů se může hodit na jinou situaci, jeden nástroj může být optimální z hlediska vývoje a druhý zas z hlediska rychlosti přístupu k databázi. Společnosti při vývoji nového programu anebo při úpravách stávajícího váhají, který nástroj je ten optimální pro jejich účely a zároveň pro uspokojení potřeb zákazníka.

Právě toto bylo motivací této práce, která se snaží pomoci těmto společnostem při rozhodování pomocí jasných dat. Existuje však spousta programovacích jazyků, které mohou přistupovat k databázi, a proto se tato práce zaměřuje pouze na platformu .NET od firmy Microsoft, konkrétně nejnovější .NET Core. Zároveň je více databází, ale pro tyto účely byla vybrána databáze od firmy Microsoft. Technologie od firmy Microsoft byly vybrány proto, že jsou v praxi ve webovém prostředí často využívány dohromady.

Cílem práce je porovnat jednotlivé nástroje na platformě .NET vůči sobě. Jako nástroje pro porovnání byly vybrány ADO.NET, Entity Framework Core a Dapper. Pro porovnáni byly vytvořeny modelové situace, které odpovídají praxi. Jedna modelová situace zastupuje velký elektronický obchod a druhá zastupuje menší firmu vyvíjející administrační systém pro zadavatele. Výsledky této práce mohou pomoci firmám při rozhodování ohledně výběru nástroje pro přístup do databáze v závislosti na jejich situaci.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této bakalářské práce je analyzovat efektivnost různých nástrojů pro práci s databází na platformě .NET a najít optimální nástroj pro dvě modelové situace. Dílčími cíli jsou:

a) Charakteristika platformy .NET
b) Charakteristika jednotlivých nástrojů pro práci s databází
c) Vytvoření aplikace pro měření efektivnosti jednotlivých nástrojů
d) Na základě výsledků zhodnotit využití jednotlivých nástrojů ve vymezené oblasti

2.2 Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. V praktické části byla navržena a implementována aplikace. Na základě vytvořené aplikace byly jednotlivé nástroje testovány v oblasti výkonu práce s databází. Aplikace byla zároveň využita k hodnocení efektivnosti kódu jednotlivých nástrojů. Pro porovnání efektivnosti nástrojů byly vytvořeny a použity dvě modelové situace. Modelová situace A představovala velký elektronický obchod a modelová situace B představovala malou firmu vyvíjející administrační systém. Pro obě modelové situace se hledal optimální nástroj. Pro nástroje byla stanovena kritéria ke správné komparaci. Kritérii byly:

- Výkon
- Počet řádků nutných na testované operace
- Počet funkcí
- Náročnost nástroje

Jednotlivé nástroje byly porovnávány pomocí vícekriteriální analýzy variant, konkrétně pomocí Saatyho metody a bodovací metody. Váhy kritérií byly stanoveny pro každou z modelových situací zvlášť. Na základě syntézy teoretických poznatků a výsledků praktické části byly formulovány závěry bakalářské práce.

3 Teoretická východiska

3.1 .NET

Jedná se o softwarovou technologii firmy Microsoft, která obsahuje všechny důležité vývojové možnosti. V současné době .NET zahrnuje základní .NET Framework a nový .NET Core. Pro vývoj dynamických webových stránek se používá ASP.NET a ASP.NET Core. Na platformě .NET je možné vyvíjet aplikace pro různorodá prostředí jako jsou:

- Web
- Mobilní zařízení
- Desktopová zařízení
- Počítačové a mobilní hry
- Machine Learning a AI (Artificial Intelligence)
- Internet věcí.

Pro psaní kódu pro jednotlivé platformy se nejčastěji využívají programovací jazyky C#, F# nebo Visual Basic. [1]

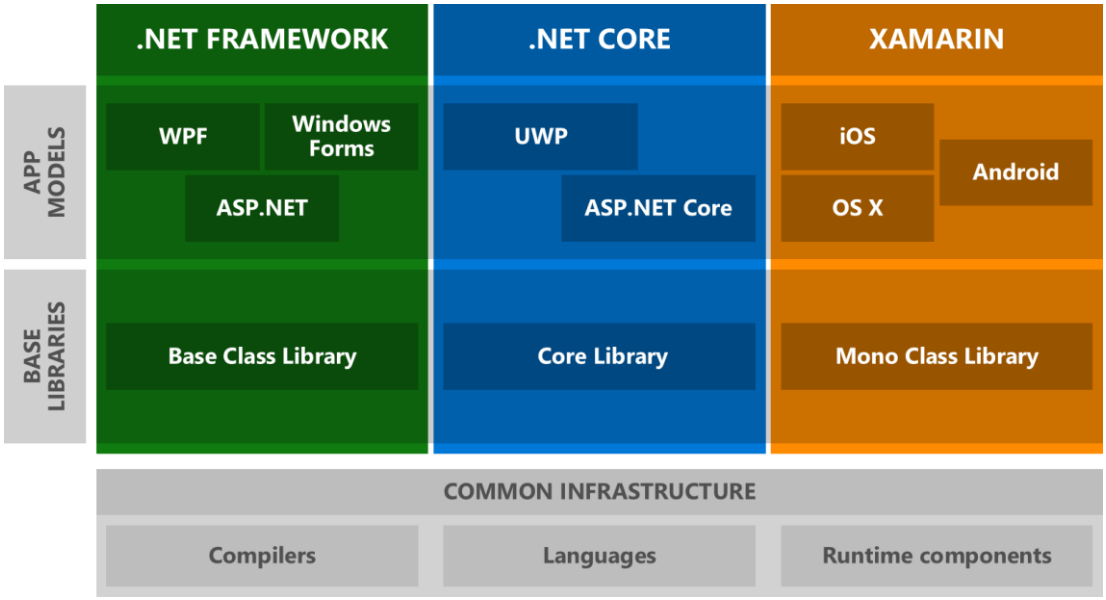
3.1.1 .NET Standard

.NET Standard je specifikace, která představuje sadu rozhraní API (Application Programming Interface), které musí implementovat všechny .NET platformy. To sjednocuje platformy a zabraňuje další fragmentaci.

Tato specifikace řeší problémy se sdílením a znovupoužitím kódu na všech platformách .NET přinesením všech rozhraní API na jedno místo. Konkrétně takovým způsobem, že dříve měla každá vývojová platforma vlastní knihovnu, jak je vidět na obrázku (Obrázek 1), ale teď díky .NET Standard jsou všechny knihovny sloučeny do jedné jak je vidět na obrázku (Obrázek 2), a proto můžeme jednotlivé funkce využít kdekoliv.

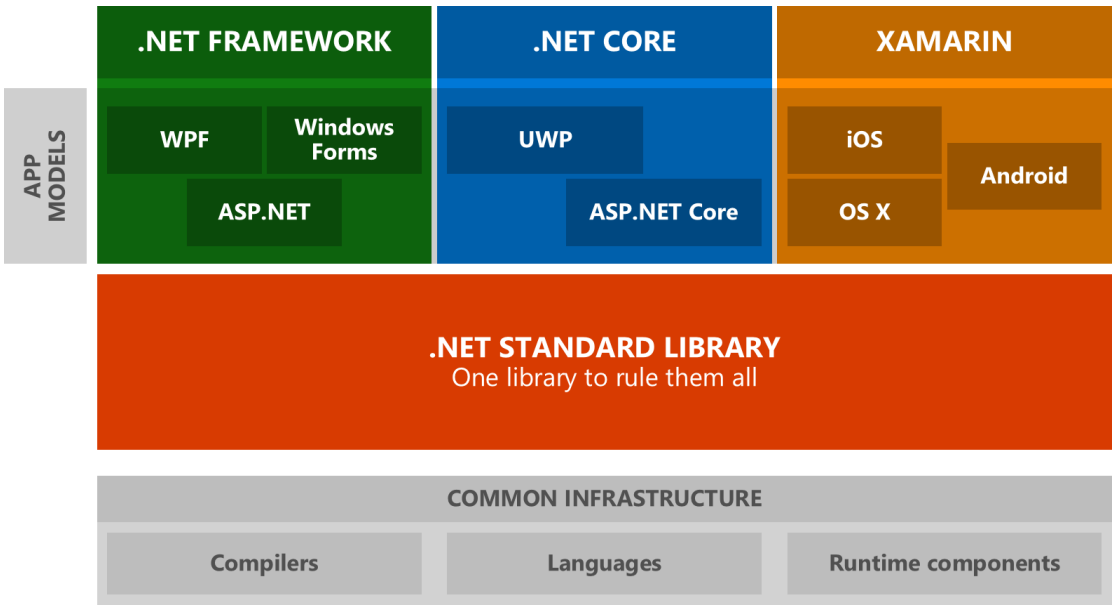
Jedná se o nahrazení jednotlivých knihoven pro jednotlivé platformy jako jsou .NET Framework, .NET Core a Xamarin. [2] [3]

Obrázek 1 – Stav před .NET Standard



Zdroj: [3]

Obrázek 2 - Současný stav s .NET Standard



Zdroj: [3]

3.1.2 .NET Framework

.NET Framework je spravované spouštěcí prostředí pro Windows, které poskytuje mnoho služeb spuštěným aplikacím. Skládá se ze dvou hlavních částí: common language runtime (CLR), který spouští kód napsaný vývojářem a .NET Framework Class Library,

která poskytuje knihovnu testovaných a znovupoužitelných kódů, které vývojáři mohou použít z vlastních aplikací. [4]

3.1.2.1 Common language runtime (CLR)

CLR je virtuální spouštěcí prostředí, které je zodpovědné za spouštění spravovaného kódu. Spravovaný kód je jakýkoliv kód napsaný ve vysokoúrovňovém jazyce jako je C#, Visual Basic apod. CLR poskytuje základní služby jako je automatická správa paměti, správa vláken a remoting a současně prosazuje přísnou bezpečnost typů a jiné formy kódu, které podporují zabezpečení.

Automatická správa paměti poskytuje nástroje jako je například Garbage Collection, který se stará o alokaci a uvolňování paměti pro aplikace. Pokud už není volné místo na alokování nové paměti spustí se Garbage Collection, který zkontroluje objekty v alokované paměti a pro ty, které se již v aplikaci dále nevyužívají provede nutné operace k získání paměti zpět.

CLR obsahuje Common type system (CTS), který definuje, jak jsou typy deklarovány, použity a zpracovávány v CLR a je důležitý pro integraci mezi různými jazyky. Integrace může fungovat způsobem, že třída zapsaná v jednom jazyce může v .NET Framework bez potíží dědit ze třídy, která byla zapsaná v jiném jazyce. [5]

3.1.2.2 .NET Framework Class Library

Jedná se o kolekci znovupoužitelných typů, které jsou těsně integrovány s CLR. Knihovna tříd je objektově orientována a poskytuje typy, ze kterých kód získává funkce.

Typy .NET Frameworku umožňují provádět řadu běžných programovacích úkolů, zahrnující správu řetězců, sběr dat, připojení k databázi a přístup k souborům. Krom těchto běžných úkolů obsahuje knihovna tříd typy, které podporují různé specializované vývojové scénáře. [6]

3.1.2.3 Historie

.NET Framework byl odpovědí na nástup platformy Java. Microsoft si uvědomil, že jeho zákazníci, jak začínali používat Javu se přesouvají do světa Linuxu, a proto musel

přijít s .NET Framework. Společně s ním byl postaven nový programovací jazyk C#, který poskytoval to nejlepší z C++, Visual Basic a Javy.

ASP.NET přinesl nový způsob vyvíjení webových aplikací, ve kterém Java v daný moment zaostávala. ASP.NET využíval jako hlavní pojivo mezi vším co může platforma nabídnout Extensible Markup Language (XML).

Microsoft postupem času vylepšoval svůj framework přidáváním nových funkcí až v roce 2016 představil odlehčený, multiplatformní .NET Core. Vývoj jednotlivých verzí společně s CLR je vidět v tabulce (Tabulka 1). [7]

Tabulka 1 - Vývoj jednotlivých verzí .NET Framework

Číslo verze	CLR verze	Datum vydání
1.0	1.0	13/02/2002
1.1	1.1	24/04/2003
2.0	2.0	07/11/2005
3.0	2.0	06/11/2006
3.5	2.0	19/11/2007
4.0	4	12/04/2010
4.5	4	15/08/2012
4.5.1	4	17/10/2013
4.5.2	4	05/05/2014
4.6	4	20/07/2015
4.6.2	4	02/08/2016
4.7	4	05/04/2017
4.7.2	4	30/04/2018

Zdroj: [12]

3.1.3 .NET Core

.NET Core byl vydán 27. června 2016. V současné době se používá .NET Core 2.1 a již je naplánována verze 3.0. Jedná se o Framework, který se snaží brát to nejlepší z .NET Frameworku a nabízí to v odlehčené verzi. Nabízí podporu pro tři programovací jazyky, kterými jsou: C#, Visual Basic a F#. Samotný .NET Core se skládá z jednotlivých

balíčku, které je možné si doinstalovat do aplikace a využívat jen ty funkce, které jsou potřeba, díky tomu je Core velmi modulární a záleží jen na vývojáři, jak si ho poskládá. Celková velikost .NET Core je díky způsobu balíčků a odlehčenému kódu oproti .NET Frameworku velmi malá. [10]

Core je veden jako open-source projekt, na kterém pracuje Microsoft společně s komunitou z celého světa. Jejich kód se nachází na stránkách GitHubu, do kterého může kdokoliv přispívat a vylepšovat, tak jednotlivé funkce. Core využívá licence MIT (Massachusetts Institute of Technology) a Apache 2. Ostatní balíčky, které je možné si doinstalovat k .NET Core a jsou vytvářeny Microsoftem jsou open-source projektem.

Při vytváření byl .NET Core celý přepsán a nejedná se pouze o lehce upravený .NET Framework. Díky přepsání nabízí čistší a modernější kód, ve kterém je kladen velký důraz na výkon a vylepšování jednotlivých funkcí. [8]

.NET Core nabízí na rozdíl oproti klasickému .NET Framework multiplatformnost. Výhody multiplaformnosti spočívají v:

- Práci na Windows, Linux a MacOS
- Práci s Windows i Linux kontejnery. Kdežto .NET Framework nabízí práci pouze s Windows kontejnery.

Těchto výhod se docílilo odstraněním závislosti na System.Web. Core jako další vylepšení nabízí menší velikost jednotlivých obrazů (image) při vytváření a následném nasazování kontejneru, a proto je doporučováno využívat při práci s kontejnery .NET Core oproti .NET Framework. [9]

Mezi další jeho výhody lze zahrnout jeho kompatibilitu s .NET Framework, Xamarin a Mono, díky .NET Standard. Core dokáže poskytovat instalaci různých verzí běhového modulu (runtime) Core na stejném počítači, díky tomu je možné mít více služeb na stejném serveru, z nichž každá jednotlivá služba bude mít svou vlastní verzi .NET Core.

.NET Core využívá především .NET Standard, avšak ne všechny knihovny leží právě v .NET Standard, to může zapříčinit problém, pokud je nutné pracovat s knihovnou, která leží mimo a je podporována pouze pro .NET Framework, v takovém případě není možné .NET Core využít. Podobný problém nastane, pokud vývojář zamýšlí použít technologii, která existuje pouze pro .NET Framework, jako může být ASP.NET Web Forms, opět je nutné použít .NET Framework.

.NET Core je složen z:

- .NET runtime (CoreCLR), který vychází z klasického CLR, ale je multiplatformní a klade větší důraz na výkon.
- Sad framework knihoven (CoreFX), které poskytují primitivní datové typy a základní nástroje.
- Sad nástrojů SDK a překladače jazyků.

Všechny tyto jednotlivé části jsou open-source projekty a vývojáři do nich mohou přispívat. [1]

3.1.3.1 ASP.NET Core

Jedná se o framework, pro vytváření moderních webových aplikací. ASP.NET Core je upravený ASP.NET 4.x s architektonickými změnami, které vedou k více modulárnímu frameworku. Jedním z hlavních důvodů přepsání byla zastaralost kódu pro ASP.NET MVC (Model-view-controller) 5 a ASP.NET Web API 2.2. Microsoft se rozhodl, že místo aby udržoval dvě podobné a zastaralé verze vytvoří ASP.NET Core, který sloučí MVC a Web API dohromady. Vývoj .NET Core je vidět v tabulce (Tabulka 2). [10]

ASP.NET Core nabízí v sobě zabudovaný návrhový vzor Dependency Injection (DI). Jedná se o mechanismus pro podporu volného propojení mezi objekty, namísto přímého propojení závislých objektů anebo předání specifické implementace do metod, tříd a metod k rozhraní. Tímto způsobem může být jakákoliv implementace rozhraní vložena do metod a tříd a tím rapidně zvýšit flexibilitu celé aplikace. [11]

ASP.NET Core je možné využívat jak z .NET Core, tak i z .NET Framework, díky tomu, že je složen z .NET Standard knihoven. [1]

Tabulka 2 - Vývoj jednotlivých verzí .NET Core

Číslo verze	Datum vydání
1.0	27/06/2016
1.1	16/11/2016
2.0	14/08/2017
2.1	30/05/2018

Zdroj: [17]

3.1.3.2 Vývojová prostředí (IDE)

Vývojová prostředí slouží vývojářům pro usnadnění práce s kódem. IDE pomáhají při vývoji programů pomocí pokročilých nástrojů jako je kompilátor, debugger, ale i zvýrazňováním syntaxí pro lepší čitelnost kódu. Jednotlivá vývojová prostředí často slouží pouze k práci s určitými programovacími jazyky.

Pro vývoj produktů Microsoft je doporučeno využívat Visual Studio, které se nachází pouze na platformě Windows a macOS ve zjednodušené verzi. Visual Studio je hlavní vývojové prostředí pro práci s .NET Frameworkem a dalšími vývojovými částmi, které jsou pouze pro Windows.

Pro práci s .NET Core je výběr vývojového prostředí daleko větší, jelikož je open-source, a proto je možné s ním pracovat i v systémech, na kterých neběží Visual Studio.

Pro .NET Core je možné využívat kromě Visual Studio také Visual Studio Code od Microsoftu, který je dostupný na všech operačních systémech. Další IDE, která lze využít pro práci s .NET Core jsou např.: Sublime Text, Vim nebo Rider od firmy JetBrains. [8]

3.2 Databáze

Databáze je velké úložiště dat, které může v jednu chvíli používat nespočetně uživatelů. Všechna data, která uživatelé chtějí získat jsou integrovaná způsobem, aby obsahovala co nejmenší množství duplikací.

Databáze také obsahuje popis svých dat, tato data se mohou také nazývat metadata. Díky tomu je databáze definována jako sebepopisující kolekce integrovaných záznamů. Popis dat se označuje systémový katalog. Systémový katalog obsahuje veškeré informace o názvech tabulek, názvech sloupců, kam patří i vlastnosti tabulek apod. Sebepopisující charakter databáze nabízí nezávislost dat. Nezávislost dat znamená situaci, kdy je do databáze přidána nová struktura nebo jsou již existující struktury aktualizovány, potom nejsou aplikace využívající databázi dotčeny a fungují stále stejným způsobem.

Databáze pracuje s daty. V databázi se uchovávají pouze surová data a až pozdější transformací je možné získat jasné vyjádření co daná data znamenají.

Pro komunikaci mezi uživatelem, databázovými aplikacemi a databází slouží systém řízení databáze (DBMS – database management system). [13]

3.2.1 Systém řízení databáze (DBMS)

DBMS má za úkol vytvářet, zpracovávat a spravovat databáze. K práci s databází poskytuje uživatelům možnost vkládat, aktualizovat, mazat a vyvolávat data z databáze.

V DBMS může uživatel vidět spoustu dat, které nejsou relevantní k jeho úkonům, a proto existuje mechanismus pohledů, který umožňuje každému uživateli pohled sestavený jemu na míru. Tento pohled je na databázi, kde pohled je podmnožinou databáze. Pohled je dotaz v dotazovacím jazyce SQL, který vytváří z tabulek jednu tabulku virtuální. Díky tomu může uživatel vidět data přesně tak, jak očekává.

Pro DBMS je specifikuje pět hlavních složek prostředí:

1. Hardware – jedná se o počítačový systém, na kterém běží databáze. Velikost může být od jednoho PC až po rozsáhlou počítačovou síť.
2. Software – zahrnuje DBMS, databázové aplikace, společně s OS i se síťovým softwarem, jestliže se DBMS využívá na síti.
3. Data – jedná se o spojovací složku mezi hardwarem, softwarem a uživateli.
4. Procedury – jedná se o instrukce a pravidla, která se využívají při řízení návrhu a používání databáze. Obsahuje i instrukce, jak se například přihlásit k databázi nebo dokáže zvládat selhání softwaru a hardwaru.
5. Osoby – do této sekce patří návrháři databáze, obchodní analytici, správci databáze atd. [14]

Architektura DBMS může být dvouvrstvá, ale i třívrstvá, která je v daný moment nejvyužívanější. Třívrstvá architektura vznikla v roce 1995 jako odpověď na čím dál složitější aplikace. Jednotlivými vrstvami jsou:

1. Uživatelské rozhraní – tato vrstva běží přímo u koncového uživatele (klient)
2. Aplikační server – tato vrstva obsahuje logiku provozu a zpracování dat. Aplikační server je vytvořen způsobem, kdy jeden server zvládá obsluhovat více klientů.
3. Databázový server – jedná se o DBMS, který má uložená veškerá data, která může požadovat střední vrstva. [18]

3.2.2 Relační model

Relační model je nejdůležitějším standardem databázového oboru. Vytvořil ho v roce 1970 E.F. Codd, který založil model na teorii relační algebry. V současné době se jeho model využívá při návrhu a implementaci většiny databází.

Všechna data v relačním modelu jsou logicky strukturována do relací (tabulek). Relaci je možné si představit jako dvourozměrnou tabulku, která je složena z řádků a sloupců. Jednotlivé objekty, které se sledují se nazývají entity. Entita je definována jako objekt, který je pro uživatele důležitý a je nutné ho reprezentovat v databázi. Každá relace by měla mít následující vlastnosti:

1. Každá tabulka má unikátní jméno pro příslušnou databázi.
2. Jednotlivé řádky relace obsahují data, která se vážou k entitě nebo k její části.
3. Atributy entity se definuje každý sloupec v tabulce.
4. Každá buňka v tabulce smí obsahovat pouze jednu hodnotu.
5. Všechny hodnoty v jednotlivém sloupci musí být stejného typu.
6. Každý sloupec musí být unikátně pojmenován.
7. Sloupce i řádky mohou být v libovolném pořadí, a přesto bude relace obsahovat stejný význam.
8. Každý záznam musí být unikátní. [13]

Pro zajištění unikátnosti záznamu je nutné určit sloupec nebo kombinaci sloupců, která zajistí unikátnost záznamu a k tomu slouží klíče relace. Pod klíčem je možné si představit jeden nebo více sloupců a klíč je jedinečný nebo nejedinečný. Jedinečný klíč je takový, který má na každém řádku unikátní hodnotu. Nejedinečný klíč slouží k identifikaci řádku, ale může se vyskytovat vícekrát v tabulce.

Pokud klíč obsahuje více než jeden atribut je takový klíč nazýván složený klíč. Stejným způsobem jako klíče s jedním atributem mohou být i složené klíče jedinečné nebo nejedinečné.

Další z klíčů se nazývá kandidátní klíč. Tento klíč jasně identifikuje záznam v tabulce a může být složen z jednoho sloupce nebo může být složeným klíčem. Z kandidátního klíče je zvolen primární klíč, který jedinečně určuje jednotlivé záznamy v tabulce. Pokud relace neobsahuje záznam, který splňuje podmínky primárního klíče

vytvoří se tzv. náhradní klíč, který slouží pouze pro identifikaci záznamu a přiděluje ho většinou DBMS. Primárním klíčem bývá obvykle číslo, které se s každým záznamem mění. Z kandidátních klíčů, které nebyly vybrány jako primární klíče se stávají alternativní klíče.

Pro vztah mezi dvěma tabulkami slouží cizí klíč. Cizí klíč je sloupec nebo i více sloupců v jedné tabulce, které se shodují s kandidátním klíčem jiné tabulky. Není nutné, aby cizí klíč měl stejné jméno jako klíč primární, důležité je, aby obsahoval pouze stejné hodnoty.

Pro správnost veškeré práce s databází existují integritní pravidla neboli omezení. Mezi integritní pravidla patří:

- Entitní integrita – se týká primárních klíčů. Znamená, že sloupec s primárním klíčem nesmí obsahovat prázdnou hodnotu.
- Referenční integrita – souvisí s cizími klíči. Jestliže je v tabulce cizí klíč, musí jeho hodnota odpovídat hodnotě některého ze záznamů v domovské tabulce, případně může mít cizí klíč prázdnou hodnotu. [14]

3.2.3 Entitně-relační modelování (ER)

Při návrhu databáze je jedním ze způsobů ER modelování. Postup při návrhu databáze tímto způsobem začíná zvolením důležitých dat (entit) a relací mezi daty, které je nutné v modelu reprezentovat. Poté přijdou na řadu informace o entitách a vztazích (atributy).

Relace se rozdělují podle stupně relace. Stupeň relace je počet entit zúčastněných v relaci. Nejčastější relace je relace stupně dva a nazývá se binární. Mezi binární relace patří také:

1. Vztah 1:1 – vztah, kdy jedna instance entity souvisí pouze z jednou další instancí entity
2. Vztah 1:N – vztah, kdy jedna instance entity souvisí s více instancemi jiné entity
3. Vztah N:M – vztah, kdy může několik instancí jedné entity souviset z mnoha instancemi jiné entity

Při návrhu databáze se také využívá normalizace. Ta má za cíl snížit redundanci dat v tabulkách. Pro zjištění, zda je databáze správně vymodelována, se kontrolují pravidla pro normální formu. Nejvyužívanějšími normálními formami jsou:

1. První normální forma (1NF) – tato forma je nejdůležitější pro vytvoření správných tabulek pro relační databázi. Ostatní formy jsou volitelné. V 1NF je nutné, aby v každé buňce, kde se střetává sloupec s řádkem byla pouze jedna hodnota.
2. Druhá normální forma (2NF) – tato forma se týká pouze tabulek, které obsahují složené primární klíče nebo jeden primární klíč. Za 2NF se má, pokud je již tabulka v 1NF a jejíž hodnoty každého sloupce, který nepatří mezi primární klíče, je možné determinovat všemi hodnotami sloupců, které primární klíč tvoří.
3. Třetí normální forma (3NF) – jedná se o situaci, kdy tabulka, která je v 1NF i 2NF a jejíž hodnoty ve sloupcích, které nejsou primárním klíčem, jsou determinovány pouze sloupci primárního klíče a nejsou determinovány žádnými jinými sloupci. **[**14]

3.2.4 SQL

Jedná se o strukturovaný dotazovací jazyk, který se využívá při práci s databází. SQL je neprocedurální jazyk, což znamená, že je nutné se ptát, co za informace požadujeme, a ne jak je získat.

Při práci s SQL jazykem jsou základním stavebním kamenem čtyři příkazy:

1. SELECT – tento příkaz se používá pro vyvolání a zobrazení dat z jedné nebo více tabulek. Je možné s ním i filtrovat data podle různých specifikací.
2. INSERT – přidává nové řádky do tabulky.
3. UPDATE – aktualizuje sloupce v tabulce.
4. DELETE – vymazává řádky z tabulky. [13]

3.3 Přístup do databáze

Přístup do databáze znamená propojení mezi zmíněnými databázemi a programovacím jazykem, v tomto případě konkrétně platformou .NET a programovacím jazykem C#.

V současné chvíli existuje více způsobů, jak získat data z databáze na platformě .NET. Některé z nich poskytuje sám Microsoft jako je např. ADO.NET nebo Entity Framework, další potom firmy, které si buď vytvořily přístup sami pro sebe a nabízejí ho dál jako open-source, nebo firmy za účelem zisku.

Výhodou toho, že existuje více způsobu, jak přistupovat k získávání dat z databáze je, že vývojář si může vybrat přesně ten způsob, který mu vyhovuje nejvíce a nemusí zůstávat pouze u jediného řešení.

Jednotlivé přístupy je možné rozdělit na:

- Objektově relační mapování (ORM), které reprezentuje Entity Framework a Dapper
- Čistý přístup do databáze, který reprezentuje ADO.NET

Existují čtyři základní operace, které se provádějí nad databází, zkratkou se nazývají CRUD operace a jedná se o:

- Create – vložení do databáze, ekvivalent v SQL je INSERT
- Read – čtení z databáze, ekvivalent v SQL je SELECT
- Update – aktualizace v databázi, ekvivalent v SQL je UPDATE
- Delete – vymazání z databáze, ekvivalent v SQL je DELETE

3.3.1 ORM

ORM má za úkol především pomoci vývojářům při vytváření aplikací, které potřebují komunikovat s databází. ORM přináší vývojářům mnoho výhod. Jednou z největších výhod je mapování relační databáze na logické objekty v programovacím jazyce, což zapříčiňuje především snížení počtu řádků kódu. Při velkém počtu tabulek, řádků a sloupců může být rozdíl oproti klasickému přístupu i několik set řádků méně, což v důsledku znamená větší efektivitu pro vývojáře.

ORM je možné rozdělit na dvě skupiny:

- klasický, plnohodnotný ORM
- Micro ORM.

Díky tomu, že existují tyto dvě skupiny si může vývojář vybrat přesně tu variantu, která vyhovuje jeho projektu a jemu samotnému. [22]

3.3.1.1 Plnohodnotné ORM

Jedná se o ORM, které poskytuje kromě automatického mapování i spoustu dalších funkcí, které mohou být pro vývojáře velmi užitečné. Tyto funkce bohužel stojí i režii navíc při práci s databází a nevýhodou může být pomalejší čtení z databáze.

Velkou výhodou plnohodnotného ORM je absence nutnosti psát vlastní SQL, protože ORM to vygeneruje automaticky. Související výhodou je snížení doby vývoje aplikací.

Naopak nevýhodou může být delší čas na začátku vývoje, kdy je nutné se seznámit s funkcemi ORM a vědět, jak ho správně nastavit.

3.3.1.2 Micro ORM

Jedná se o odlehčenou variantu plnohodnotného ORM, který poskytuje, stejně jako plnohodnotné ORM automatické mapování, ale postrádá některé funkce z plnohodnotného ORM.

Jedním z hlavních rozdílů oproti plnohodnotnému ORM je, že v Micro ORM je nutné psát vlastní SQL dotazy. Výhodou toho může být lepší rychlost při práci s databází, protože je možné upravit dotazy přesně pro požadavky aplikace. Nevýhodou je, ale nutná znalost SQL jazyka.

Chybějící funkcí v Micro ORM je především:

- Nepodporující vazby jako jsou 1:1, M: N, ve výsledku to znamená, že pokud je načítán objekt z databáze, nebudou všechny související objekty automaticky načteny nebo uloženy. Pokud je potřeba načíst související objekty, je nutné vytvořit dotaz speciálním způsobem, kde se způsob liší v závislosti na ORM.

Hlavní výhody jsou pouze dvě, patří mezi ně lepší výkon oproti plnohodnotnému ORM a také jednoduchost vytvoření, protože není potřeba zdlouhavé nastavování. [21]

3.3.1.3 Dapper

Za vytvořením Dapperu stojí tým StackExchange, ten zahrnuje například stránku Stack Overflow, který se v roce 2011 rozhodl, že potřebuje zajistit maximální výkon při

práci s databází a nechce jednotlivé objekty ručně mapovat. Jedná se o open source projekt, kde je možné prohlížet si kód a také do něj přispívat.

Dapper patří mezi Micro ORM a vyniká svou rychlostí. Dapper přináší největší výhodu v automatickém mapování a odkazuje se, že má téměř identickou rychlostí, jakou má klasický přístup s ADO.NET. [23]

Dapper v základu obsahuje pouze základní funkce, jako je automatické mapování. Pokud je potřeba funkcí, které nejsou v základním balíčku dostupné, je možné, že existuje rozšíření, které požadované funkce obsahuje. Výhodou toho je, že vývojář si může jednotlivé funkce, které potřebuje doinstalovat, je ale nutné si uvědomit, že s tím souvisí snížení výkonu a výhoda rychlosti Dapperu, v případě mnoha rozšíření, zmizí.

Psaní kódu, kvůli psaní celých SQL dotazů, pro jednotlivé operace je náročnější než například pro Entity Framework, ale zároveň, díky automatickému mapování je úspornější než ADO.NET a vývojář by měl získat podobnou rychlost jakou má ADO.NET.

Jednou z nevýhod oproti Entity Frameworku je, že pro načítání dalšího objektu je nutné vytvořit JOIN klausuli. Ta získá data z více tabulek a poté je nutné pomocí Lambda výrazu přiřadit jednotlivé objekty. Rozdělení na objekty se provádí podle určeného sloupce.

Díky tomu, že Dapper přináší především rozšíření IDbConnection je kompatibilní téměř se všemi existujícími databázemi, stačí pouze napsat SQL kód pro danou databázi. [24]

3.3.1.4 Entity Framework (EF)

Jedná se o přístup do databáze od firmy Microsoft. EF je plnohodnotné ORM s mnoha funkcemi a Microsoft v současnosti dává velký důraz na vývoj EF, v tuto chvíli se vyvíjí i Entity Framework Core, který je odlehčenou verzí EF. Entity Framework vychází z ADO.NET, nicméně EF si nedává za cíl nahradit ADO.NET, jedná se pouze o jiný přístup k datům.

Mezi velkou výhodu EF patří, že není nutné používat přímo SQL kód, pokud si to vývojář nepřeje. EF je především o modelování. Při práci s EF je nutné vytvořit entity, které jsou konceptuálním modelem fyzické databáze. Formálně se jedná o Entity Data Model (EDM). EDM je struktura pro definování dat, které se používají v aplikacích. EDM

definuje datové typy, specifické definice pro to, jaké typy vazeb jsou povoleny, schémata, která model podporuje a mapování mezi těmito schématy. Každá entita obvykle obsahuje jednu nebo více vlastností se specifickým datovým typem. Entita také obsahuje navigační vlastnosti, které odkazují na jiné entity. Každá entita obsahuje i set vlastností označující se Key neboli unikátní identifikátor entity. Obvykle se Key získávají z reprezentace v databázi. Ukázka fungování EF je vidět na obrázku (Obrázek 3). [25]

EF používá ve svém jádře infrastrukturu ADO.NET, konkrétně ke komunikaci s datovým úložištěm využívá datového poskytovatele ADO.NETu. Tento datový poskytovatel musí být ovšem aktualizován na podporu nové služby předtím, než může komunikovat s EF. S tím souvisí i podpora databázových systémů, datový poskytovatel Microsoft SQL Serveru je aktualizován a ostatní poskytovatelé nabízejí datové poskytovatele pro EF. Avšak pokud se nevyužívá SQL Server je nutné konzultovat podporu pro EF s výrobcem databázového systému.

Jednou z hlavních tříd v EF je DbContext, která poskytuje klíčové funkce pro třídy, které z ní dědí. Mezi tyto funkce patří schopnost uložit všechny změny, upravení řetězce pro připojení, vymazat objekty, volat uložené procedury a další základní funkce.

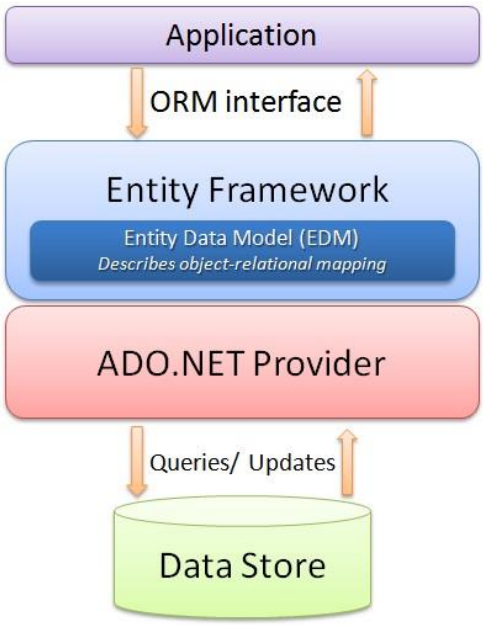
Další z výhod EF je způsob Code First, kde je možné vytvořit entity a pomocí funkcí ve Visual Studiu vytvořit buď novou databázi nebo doplnit tabulky do již existující databáze. Zároveň tento přístup neznamená, že nemůžeme pracovat už s hotovou databází a pracovat pouze s daty uvnitř. [27]

EF nabízí tři způsoby načítání dat:

1. Lazy loading je proces při kterém je entita nebo sada entit automaticky načtena z databáze pokaždé, kdy je navigační vlastnost požadována v kódu. Tento způsob práce může způsobovat veliké snížení výkonu, protože je automaticky aktivován.
2. Eager loading načítá objekty v jednom volání do databáze. Je to ideální v případech, kdy vývojář ví, že chce načíst všechny příbuzné objekty. Do databáze jde příkaz, který obsahuje klauzuli JOIN.
3. Explicit loading funguje stejným způsobem jako lazy loading jen s tím rozdílem, že v tomto případě si načítání příbuzných záznamů řídí sám programátor.

Jednou z nevýhod může být náročnější konfigurace připojení k databázi, jelikož je nutné mít správně namodelované entity a také třídu, která dědí z DbContext. [1] [19]

Obrázek 3 - Princip fungování Entity Framework



Zdroj: [15]

3.3.1.5 Entity Framework Core (EF Core)

Entity Framework Core je odlehčená, rozšiřitelná a multiplatformní verze Entity Frameworku. EF Core je vytvořen, stejně jako EF, týmem v Microsoftu. EF Core je kompletně přepsaný EF se všemi výhodami .NET Core. Kvůli tomu, že EF Core je úplně přepsaný, neobsahuje všechny funkce, jaké obsahuje EF. V současné verzi již ovšem obsahuje veškeré důležité prvky, které vývojář očekává.

Některé prvky si EF Core od EF nepřevezme, a to například EF visual designer. EF visual designer se používá při jednom z přístupu založení v EF, konkrétně se jedná o Model First. V současné chvíli se však vývojáři zaměřují především na Code First přístup, a proto tato ztráta v EF Core není tak významná.

EF Core vypadá, že funguje stejně jako EF, ale ve velmi podobných situací má menší vylepšení, kterými jsou:

- Vytvoření a konfigurace DbContext je založeno na dependency injection, a ne pouze na dědičnosti.

- Instalace je modulární, a proto je možné si vybrat, které funkce zrovna vývojář požaduje.
- Je zde významné zlepšení výkonu díky čistšímu a modernějšímu kódu a vylepšení dotazů jako jsou dávkové dotazy.

V současné chvíli je doporučováno využívat spíše EF Core, především díky zlepšení výkonu a výhod, které souvisejí s .NET Core. Nicméně je možné, aby v jedné aplikaci fungoval EF Core i EF, záleží vždy na vývojáři, co si vybere. [1][26]

3.3.2 ADO.NET

Jedná se o přístup do relační databáze nebo k XML, který vytvořil Microsoft. Microsoft ho doporučuje při využívání .NET Frameworku. Případně je možné zvažovat i Entity Framework od Microsoftu. Pod pojmem ADO.NET je možné si představit sadu knihoven, které nám poskytují funkce k přístupu do relační databáze nebo k XML. Mezi hlavní část patří především System.Data.dll. Princip fungování ADO.NET je vidět na obrázku (Obrázek 4).

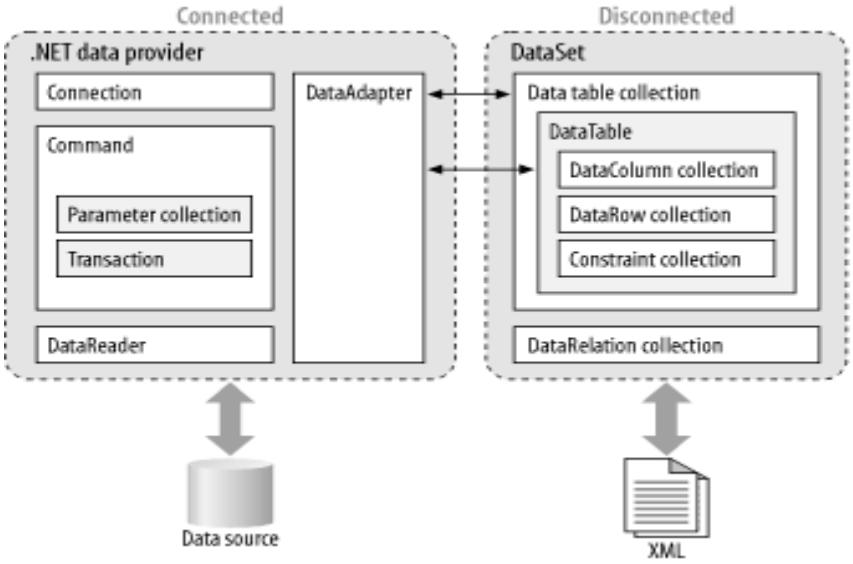
Funkce ADO.NETu využívají i jiní poskytovatelé připojení k databázím, jako například Entity Framework. [29]

ADO.NET se rozděluje na dvě hlavní části, které jsou více popsány dále:

- *Disconnected layer* – je možné vytvořit i bez připojení k uložišti dat
- *Connected layer* – slouží přímo k připojování a odpojování k uložišti dat

V ADO.NETu se používá buď čistě napsané SQL příkazy nebo uložené procedury, které se později provedou. Díky tomu má ADO.NET vysoký výkon. Nevýhodou je, že je nutné si jednotlivé objekty namapovat ručně. [1][28]

Obrázek 4 - Princip fungování ADO.NET



Zdroj: [16]

3.3.2.1 Disconnected layer

Disconnected třídy mohou být používány bez připojení k datovému úložišti. Třídy pro získávání dat lze nalézt v System.Data.dll ve jmenném prostoru (namespace) System.Data.

Hlavními třídami jsou DataTable a DataSet. Při práci s disconnected layer je vždy nutné používat DataTable. DataTable představuje tabulková data v mezipaměti, jako je tabulková mezipaměť řádku, sloupců a omezení. Při práci s DataTable je nutné prvně vytvořit instanci DataTable třídy a poté přidat DataColumn objekty, které definují typy dat a vložit DataRow objekty, která obsahují data. [28]

DataSet je paměťová, tabulková, relační reprezentace dat. DataSet funguje jako relační databáze v paměti, ale jedná se pouze o data v paměti a jinak neposkytuje žádné transakční vlastnosti, které jsou obvyklé pro klasické relační databáze. DataSet obsahuje kolekci DataTable a DataRelation objektů. DataRelation reprezentuje vztah mezi dvěma DataTable objekty. DataTable objekt může obsahovat primární a cizí klíč k prosazování integrity dat. [29]

3.3.2.2 Connected layer

ADO.NET poskytuje různé datové poskytovatele pro komunikaci s různými databázovými studii (DBMS) jako je např. MSSQL, MySql a další. Výhodou je, že je možné naprogramovat specifického datového poskytovatele pro přístup k unikátním funkcím určitého DBMS. Další výhodou je přímý přístup do databáze bez různých prostředníku a vyšší výkon poskytovatele.

V .NET jsou přímo tito poskytovatelé: OLE DB, ODBC a Microsoft SQL Server LocalDB. Většina DBMS má případně svého poskytovatele ke stažení. [1]

Pro správné připojení k datovému úložišti se využívá třída DbConnection. Je důležité v případě, že se dokončila práce s objektem vždy ukončit spojení s datovým úložištěm.

Pro zasílání SQL příkazů do datového úložiště se využívá objekt DbCommand. Tato třída může sloužit pro získávání, vkládání, aktualizování nebo mazání dat. Pro správné fungování DbCommand je nutné správné připojení do datového úložiště. [28]

Co se týče získávání dat z datového úložiště, slouží pro to objekt DbDataReader. Jedná se o velice efektivní způsob, jak získat veškerá data z datového úložiště. Další způsob, jak získat data z datového úložiště je pomocí DbDataAdapter, který získaná data může jednodušeji uložit do DataSet. Případné provedené změny v DataSetu vrací zpět na zdroj dat.

ADO.NET poskytuje také hromadné operace (BULK), které se využívají v případě, kdy je třeba poslat více dat najednou do databáze. Tato operace pošle všechny najednou místo toho, aby posílala jednotlivé příkazy po jednom. [29]

3.4 Shrnutí

Pro účely této práce byla využita platforma .NET, konkrétně nejnovější .NET Core, který vyniká svou modulárností a možností fungovat na všech operačních systémech. Na této platformě, která byla podrobněji představena v kapitole **č. 3.1** je vždy při vývoji nové aplikace nebo programu pro firmy jedním ze zásadních rozhodnutí výběr nástroje pro přístup k databázi, na jednotlivé nástroje se práce zaměřuje v kapitole **č. 3.3**. Tento výběr závisí na požadavcích firmy ohledně výkonu, funkcí a zkušenostech jednotlivých vývojářů, kteří budou nástroj využívat.

Firmy si mohou vybírat ze zavedených nástrojů jako je ADO.NET, který by měl vynikat svou rychlostí a zaostávat v přidaných funkcí pro snazší vývoj aplikací. Dalším z nástrojů firmy Microsoft je Entity Framework, v praktické části je konkrétně využit nejnovější Entity Framework Core, který nabízí spoustu funkcí, které mohou usnadnit vývoj aplikace, ale může zaostávat v rychlosti. Posledním z vybraných nástrojů je Dapper, který by měl mít podobnou rychlost jako ADO.NET a má přidanou funkci v podobě přímého mapování do objektů. Jednotlivé specifikace ohledně rychlosti budou ověřeny pomocí vytvořené aplikace v praktické části.

Problém při výběru vhodného nástroje se z tohoto důvodu bude řešit v praktické části, kde výsledky mohou pomoci firmám při výběru vhodného nástroje.

4 Vlastní práce

Na platformě .NET existuje nepřeberné množství nástrojů pro přístup k databázi, ale v praktické části bude kladen důraz na tři nejpopulárnější bezplatné nástroje, kterými jsou: ADO.NET, Entity Framework Core a Dapper.

Tyto nástroje byly vybrány pro svoji popularitu a možnost využívat je zdarma bez omezení. Jednotlivé nástroje budou porovnávány s ohledem na kritéria, kterými jsou:

- Výkon
- Počet řádků nutných na testované operace
- Počet funkcí
- Náročnost nástroje

Pro měření výkonu bude vytvořen program, který porovná tyto nástroje při přístupu k databázi pro všechny CRUD operace. Jako databáze bude využita MSSQL (Microsoft SQL Server) od firmy Microsoft.

Pro porovnání jednotlivých nástrojů budou vytvořeny dvě modelové situace odpovídající praxi. Jednotlivé modelové situace jsou podrobně popsány v kapitole č. **4.1**. Pro obě modelové situace se bude hledat optimální nástroj pomocí vícekriteriální analýzy variant, konkrétně Saatyho metodou a bodovací metodou. Obě dvě situace budou mít stanoveny váhy vůči výše zmíněným kritériím. Jednotlivá kritéria budou bodově ohodnocena na základě důkladné analýzy.

Výsledkem praktické části bude zhodnocení těchto nástrojů a následný výběr optimální varianty pro každou z modelových situací.

4.1 Modelové situace

4.1.1 Modelová situace A

Firma A patří mezi největší internetové prodejce na trhu a zvažuje z důvodu stáří jejich systému přepsání současného systému na modernější systém. Současná verze je psána pomocí technologie .NET Framework. Ve firmě jsou velké zkušenosti s tímto frameworkem, a proto se plánuje nasadit nový .NET Core, který vychází z .NET Frameworku.

Jako databáze se využívá MSSQL od společnosti Microsoft. K této databázi se přistupuje pomocí nástroje ADO.NET. Tento způsob byl původně vybrán především, kvůli vysoké vytíženosti systému. Zvažuje se, že by ADO.NET mohl nahradit nějaký ORM, kvůli časové náročnosti, která je v současnosti, s ADO.NET a jeho mapováním výsledků z databáze do modelů, velká.

Firma se rozhoduje mezi třemi nástroji, včetně současného ADO.NET. Její požadavky jsou především vysoký výkon, ale zároveň hledá nástroj, který by ji pomohl ušetřit čas při vývoji nových funkcí.

Velkou část logiky má **firma A** uloženou v procedurách, a proto je nutné, aby s nimi dokázal nový nástroj správně a bez problémů pracovat. Tyto procedury mohou mít i stovky řádků a může se zde spojovat pět a více tabulek dohromady.

4.1.2 Modelová situace B

Firma B se zabývá vytvářením administračních systému pro jiné firmy. Firma získala nabídku na vytvoření zcela nového administračního systému pro dodavatelskou firmu. V minulosti stavěla veškeré projekty na .NET Frameworku a jeden projekt na .NET Core. Pro tento nový projekt si proto vybrala novější .NET Core. Databáze, kterou využívají je MSSQL od společnosti Microsoft.

Požadavky firmy jsou především rychlé dodání administračního systému a vysoká spolehlivost, výkon není prioritou, jelikož systém budou ovládat pouze desítky lidí. **Firma B** zvažuje, jakým způsobem přistupovat do databáze tak, aby se jednalo o spolehlivý a zároveň jednoduchý nástroj.

4.2 Návrh a vývoj aplikace

Při vývoji programu byl využíván .NET Core 2.1. Vybranými nástroji byly:

1. ADO.NET – verze 4.4.3
2. Entity Framework Core – verze 2.1.4
3. Dapper – verze 1.50.5

Byla vytvořena konzolová aplikace, která obsahovala dva moduly. První z nich sloužil ke spuštění aplikace, k interakci s uživatelem a pro nastavení nutná k přístupu do databáze. Tento modul se jmenoval *BakalarskaPraceConsole*.

Druhý modul obsahoval logiku přístupu do databáze pro jednotlivé nástroje. Tento modul se jmenoval *AccessFacade*. Toto řešení bylo navrženo z důvodu maximální flexibility při práci s tímto projektem. Toto řešení umožňuje rychlejší změny uživatelského rozhraní, než kdyby se vytvořil nemodulární program.

4.2.1 BakalarskaPraceConsole modul

V *BakalarskaPraceConsole* se jako první vytvoří ConnectionPool do databáze, příklad je vidět na obrázku (Obrázek 5). Je to z důvodu, aby měly všechny nástroje stejné podmínky nehledě na pořadí přístupu do databáze. Pokud by se nevytvořil ConnectionPool před vstupem prvního nástroje do databáze znamenalo by to, že nástroj, který vstoupí do databáze jako první ho bude muset vytvořit a zapříčiní pomalejší výsledky prvního nástroje oproti nástrojům, které po něm následují. *ConnectionPool* stačí pouze otevřít a znovu zavřít, nicméně aplikace ho nechává ještě přibližně 10 minut otevřen.

Obrázek 5 - ConnectionPool

```
public void OpenConnectionPool()
{
    using (var connection = new SqlConnection(options.connectionString))
    {
        connection.Open();
        connection.Close();
    }
}
```

Zdroj: vlastní zpracování

V dalším kroku v *BakalarskaPraceConsole* si uživatel nejprve vybere CRUD operaci, kterou si přeje spustit. Je zde na výběr mezi asynchronním, synchronním způsobem a způsobem pomocí procedur.

Tímto krokem se program přesouvá přes rozhraní k *MainService,* ve kterém se provádí měření a volání *AccessFacade* modulu. Na začátku každé metody se inicializuje pole, které uchová celkový čas. Tento čas se měří pomocí třídy Stopwatch() z jmenného prostoru System.Diagnostics. Pro každý nástroj je vytvořen vlastní Stopwatch() a výsledky se na konci uloží přes druhý modul do databáze.

Každá metoda obsahuje smyčku s tisíci opakováními. Jedná se o nasimulování několika požadavků do databáze za sebou. Tisíc opakování bylo vybráno tak, aby odpovídalo vysokému zatížení. Toto opakování může také ovlivňovat výsledky, protože nástroje mohou mít první vstup do databáze pomalejší, ale dalšími vstupy mohou zrychlovat více než ostatní nástroje. Před začátkem každé smyčky začíná Stopwatch() měřit a po provedení celé smyčky se měření ukončí. Ukázka provedení je vidět na obrázku (Obrázek 6).

Obrázek 6 - Smyčka

```
dapper.Start();
for (int i = 0; i < 1000; i++)
{
    string dapperSync = dapperService.SelectDapperSync();
}
dapper.Stop();
times[0] = dapper.Elapsed.ToString();
```

Zdroj: vlastní zpracování

Metody, které provádějí operace INSERT a DELETE volají před vstupem do smyčky pomocnou metodu k vložení, respektive k vymazání, položek z tabulky. Je to z důvodu, aby měly všechny nástroje stejné podmínky. Pro výše dvě zmíněné operace a UPDATE se také generují náhodné hodnoty pomocí knihovny *RandomNameGeneratorLibrary*.

4.2.2 AccessFacade modul

V jednotlivých smyčkách se aplikace přesune do druhého modulu *AccessFacade*, ve kterém se využívá *Repository pattern*, který spočívá v tom, že každý nástroj a způsob přístupu do databáze má vlastní repositář. Každý repositář má své vlastní rozhraní. Tato rozhraní dědí z rozhraní *IBusinessObject*, který obsahuje čtyři základní CRUD operace. Rozhraní, která se využívají pro asynchronní přístup dědí z rozhraní *IBusinessObjectAsync*.

Repositáře obsahují veškerou logiku pro přístup k databázi. ADO.NET i Dapper využívají klíčového slova *using* a tedy IDisposable, který zajišťuje správné ukončení spojení s databází.

Modul obsahuje čtyři základní services, tři pro jednotlivé nástroje a jednu service pro ukládání výsledků do databáze, otevření ConnectionPool a zároveň pomocné práce s databází u operací INSERT a DELETE.

Dále obsahuje *AccessFacadeOptions* třídu, která obsahuje ConnectionString, který se předává pomocí *DependencyInjection* do jednotlivých repositářů, případně do *Context* pro Entity Framework Core. *ConnectionString* se získá z další třídy v modulu, která se nazývá *ServiceExtension* a registrují se v ní také jednotlivé závislosti mezi repositáři, jejich rozhraními a jednotlivými services.

Modul také obsahuje složku Dal, ve které jsou již zmíněné repositáře s jejich rozhraními. Dále je zde umístěna třída *EfCoreDbContext*, ve které se provádí nastavení pro správné fungování Entity Framework Core, tato třída dědí z DbContext. Poslední složkou je Entities, která obsahuje modely jednotlivých tabulek z databáze včetně jejich navigačních vlastností pro následné správné mapování mezi těmito objekty.

4.2.3 Vývojové prostředí

Pro zajištění správných výsledků se testovalo vždy na stejném počítači ve stejný čas. K testování se využíval počítač, který měl procesor Intel Core i7-4600M s CPU 2.90GHz. RAM byla 8 GB, systém byl 64bitový Windows 10 Pro. Jako pevný disk sloužil SSD disk o kapacitě 256 GB. Při testování byly vždy všechny aplikace až na testovací ukončeny a počítač nebyl připojen k internetové síti.

Pro vývoj programu byl využíván program Visual Studio 2017 ve verzi 15.9.4 společně s SQL Server Management Studio verze 17.9.1 a SQL Serverem 2017 všechny nástroje od firmy Microsoft.

4.2.4 Návrh databázové struktury

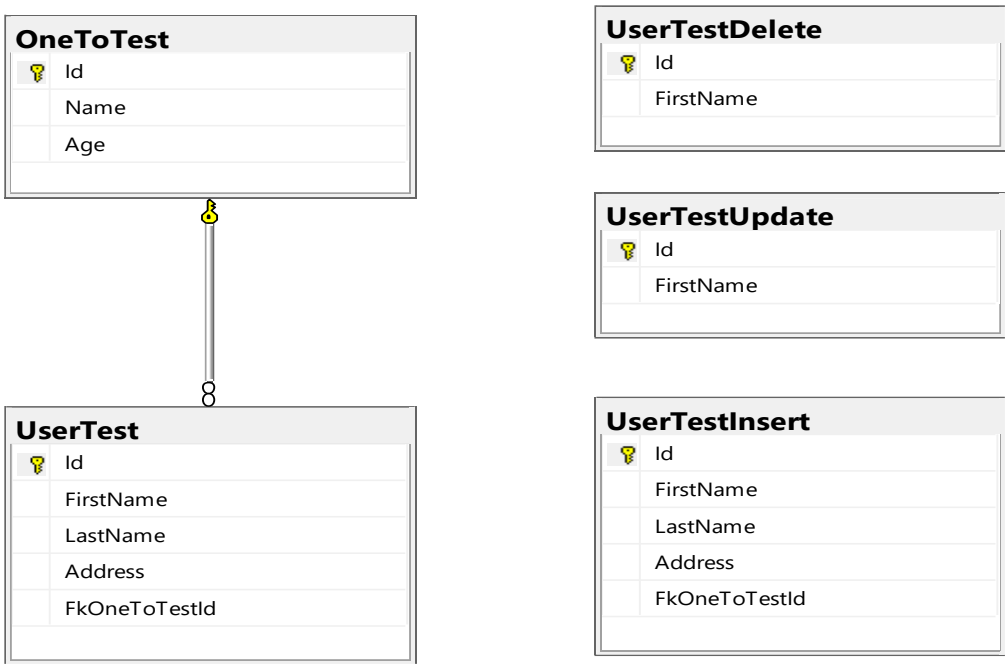
K vytvoření databáze byl použit server SQL Server 2017, který je volně ke stažení ze stránek společnosti Microsoft. Pro práci se serverem se využíval SQL Server Management Studio (SSMS).

SSMS sloužilo k vytvoření nové databáze pro otestování výkonu jednotlivých nástrojů, která bude sloužit k nasimulování základních operací, které se dají provést v databázi.

Databáze je zároveň dostatečně jednoduchá, aby se mohl ukázat výkon jednotlivých nástrojů. Nicméně vyskytuje se zde i vztah 1:N, který ukáže, jaké problémy mohou nastat, pokud chceme provádět složitější operace.

V SSMS byly také vytvořeny čtyři procedury, které následně voláme z aplikace. Diagram vytvořené databáze je vidět na obrázku (Obrázek 7).

Obrázek 7 - Diagram struktury databáze



Zdroj: vlastní zpracování

4.2.5 Úskalí nástrojů

Jedním z hlavních problémů, který se vyskytl při vývoji aplikace je operace, při které se spojuje více tabulek dohromady, konkrétně se jedná o vztah 1:N. U všech použitých nástrojů to obnášelo složitější práci s nimi, jelikož na tyto operace nejsou příliš dobře vyvinuty.

Velké rozdíly oproti jednoduchému SELECT byly v případě nástroje Dapper, kde bylo nutné upravit syntaxi volání a také využití *Dictionary*. Dapper přímo v dokumentaci neuvádí příklady, jak pracovat s takovým vztahem, proto je to bráno jako nestandartní přístup. *Dictionary* bylo nutné využít i u ADO.NET, nicméně zbytek volání do databáze zůstal stejný. Nejjednodušší práce na pohled byla u Entity Framework Core, kde je velmi čistý kód přístupu do databáze. Je to dáno jeho *Contextem*, protože v něm se tento vztah

nastavuje a dále je nutné mít nastaveny správně navigační vlastnosti u objektů. Celkově je však kód pro tento vztah nejpřehlednější u Entity Framework Core.

Další z problémů, který byl již popsán v bodě č. **4.2.1** je ConnectionPool, kde je důležité ho inicializovat pro správné měření pro každý nástroj bez ohledu na pořadí volání.

Další problém se týkal Entity Framework Core. Jednalo se o případ, kdy by se volaly všechny CRUD operace postupně v jednom spuštění aplikace. V takovém případě po první operaci SELECT provedenou Entity Framework Core, která se provedla v obvyklém čase, následovaly několikanásobně delší časové úseky, než bylo obvyklé u zbylých CRUD operací. Tento problém je znám a nahlášen na opravu u týmu stojícího za vývojem Entity Framework Core. V práci se problém vyřešil tím, že vytvořená aplikace vždy testuje jen jednu operaci pro každý nástroj.

4.2.6 Testování

Aplikace byla spuštěna každý den v různé časy. Samotné testování probíhalo bez připojení k internetu a bez toho, aby běžela jakákoliv jiná aplikace. To zajistilo optimální podmínky pro měření.

Před každým testováním se nově spustila aplikace a pomocí jednoduchého uživatelského rozhraní byla vybrána operace, která se bude vykonávat. Operace měřila celkový čas pro všechny tři nástroje. Měření obsahovala veškerou práci s daným nástrojem včetně inicializace objektu pro Entity Framework Core, protože při práci s ním to je nutností. Každá operace na konci měření vypsala výsledky na obrazovku počítače, zároveň je uložila do databáze a byla vypnuta. Tímto způsobem se pokračovalo pro každou operaci z toho důvodu, aby byla zajištěna co největší konzistence měření pro každou operaci zvlášť.

4.2.7 Výsledky měření

Výsledky měření se v databázi uložily v časovém formátu společně s milisekundami. Následně se získala průměrná doba jednoho měření vždy pro každou operaci a každý způsob zvlášť. Průměrné hodnoty za každou operaci a způsob se sečetly dohromady a byl získán celkový průměrný čas pro každou operaci. Poté se ohodnotily výsledky operací pomocí pořadí od jedné do tří, kde první místo znamená nejlepší nástroj

a třetí místo nástroj nejhorší. K jednotnému výsledku, který nástroj je nejrychlejší za všechny operace se použila četnost jednotlivých pořadí pro každý nástroj.

Z výsledků je patrné, že co se týče rychlosti vyhrál ADO.NET, ale jen o malý rozdíl oproti Dapper a za ním je s obrovskou ztrátou Entity Framework Core. ADO.NET patřil mezi nejrychlejší nástroje ve všech měřeních kromě operace SELECT, ve které zvítězil Dapper. Výsledky měření pro jednotlivé operace jsou vidět v tabulce (Tabulka 3), četnost pořadí a celkové pořadí nástrojů je vidět v tabulce (Tabulka 4). Podrobnější výsledky lze nalézt v přílohách.

Tabulka 3 - Výsledky měření

	SELECT	INSERT	UPDATE	DELETE	**CELKEM**
DAPPER	0:23:29,267	0:00:09,449	0:00:11,444	0:00:07,449	**0:23:57,609**
EFCORE	0:48:41,525	0:02:07,659	0:00:50,512	0:00:34,608	**0:52:14,304**
ADO.NET	0:26:43,510	0:00:07,815	0:00:08,165	0:00:07,099	**0:27:06,589**

Zdroj: vlastní zpracování

Tabulka 4 – Celkové pořadí nástrojů

	Pořadí				**Konečné pořadí**
	SELECT	INSERT	UPDATE	DELETE	
Dapper	1.	2.	2.	2.	**2.**
ADO.NET	2.	1.	1.	1.	**1.**
EFCORE	3.	3.	3.	3.	**3.**

Zdroj: vlastní zpracování

4.3 Výpočet pomocí vícekriteriální analýzy variant

Pro nastínění reálného výběru mezi těmito třemi nástroji jsou vytvořeny dvě modelové situace. Tyto situace byly vybrány tak, aby co nejvíce odpovídaly využívání těchto nástrojů v praxi.

Každá situace bude porovnávána pomocí vícekriteriální analýzy variant. Jednotlivé porovnávání se provede metodou párového porovnání, konkrétně Saatyho metodou a bodovací metodou. Dvě metody byly zvoleny z důvodu co nejpřesnějších výsledků.

Jednotlivé situace budou mít stanovené váhy pro jednotlivá kritéria tak, aby co nejvíce odrážela stanovené požadavky firem. Byla stanovena celkem čtyři kritéria.

4.3.1 Saatyho metoda

Saatyho metoda bere v úvahu všechny prvky, vazby mezi nimi a intenzitu s jakou na sebe působí. Pro stanovení jednotlivých vah a vyjádření velikosti preferencí mezi jednotlivými kritérii se využívá Saatyho škála.

Tabulka 5 - Saatyho škála

Číselné preference	Slovní preference
1	Kritéria (varianty) jsou **stejně** významná
3	První kritérium (varianta) je **slabě** významnější než druhé
5	První kritérium (varianta) je **silně** významnější než druhé
7	První kritérium (varianta) je **velmi silně** významnější než druhé
9	První kritérium (varianta) je **absolutně** významnější než druhé

Zdroj: [20]

Pro přesné porovnání lze využít také sudé hodnoty, konkrétně 2, 4, 6 a 8, tyto hodnoty se nazývají mezihodnotami.

Při stanovování preferencí mezi jednotlivými variantami (kritérii) se využívá párové porovnání, to spočívá v určení vyšší preference varianty (kritéria) a stanoví se mu hodnota ze Saatyho škály, variantě (kritériu), která má menší preferenci se přiřadí hodnota v podobě 1/hodnota vyšší preference.

4.3.2 Bodovací metoda

Důležitost jednotlivých kritérií se stanovuje bodovým ohodnocením, čím je kritérium důležitější, tím má více bodů. Bodovací stupnice byla stanovena na rozpětí 1 až 10 bodů. Pro výpočet je nutné stanovit normované váhy, které se vypočítají jako podíl mezi body kritérii a sumou všech bodů kritérií.

4.3.3 Stanovení kritérií

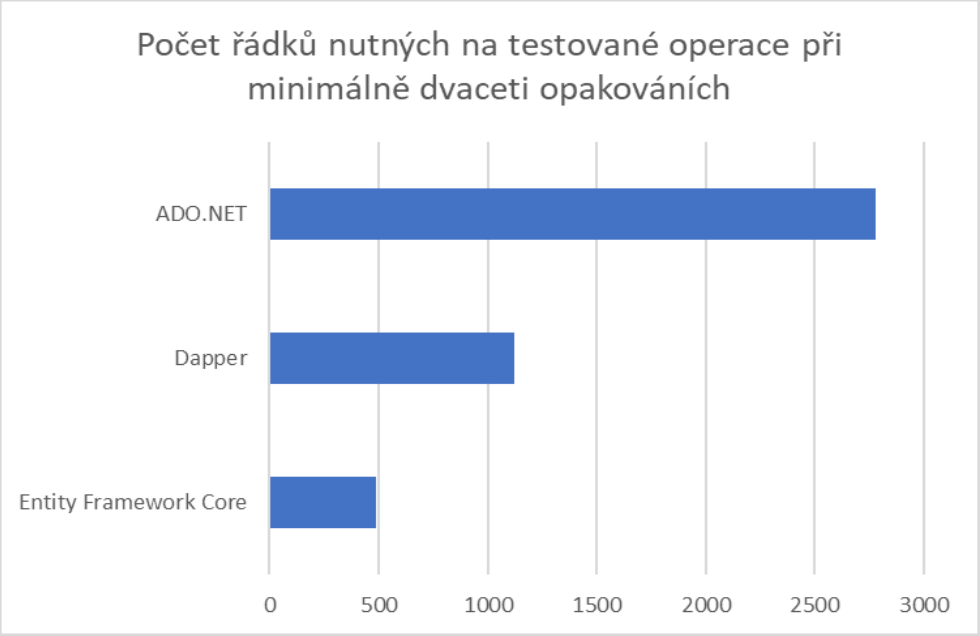
Kritéria byla stanovena jako maximalizační neboli čím vyšší hodnota tím lépe. Bodové ohodnocení kritérií v tabulce (Tabulka 6) bylo založeno na důkladné analýze nástrojů. Nástroje byly hodnoceny na stupnici od jedné do deseti.

Výkon byl následovně převeden z celkového výsledku na body, aby bylo možno mezi sebou kritéria porovnat. Pomocí těchto kritérií se následně stanovily váhy pro jednotlivé modelové situace.

Jednotlivá kritéria jsou:

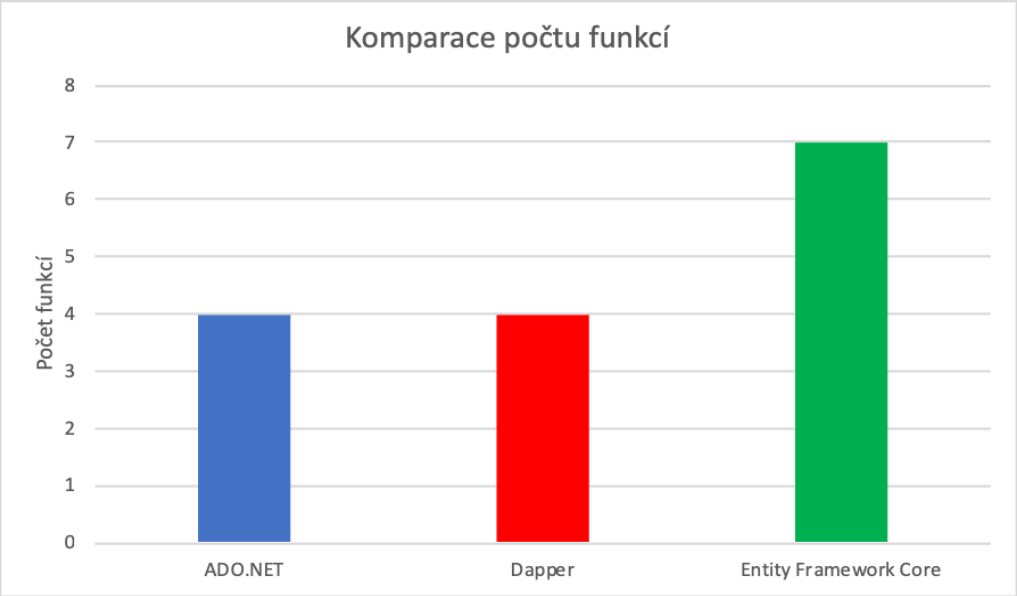
1. **Výkon** – měřen pro všechny CRUD operace a pro synchronní, asynchronní a způsob s procedurami.
2. **Počet řádků nutných na testované operace** – jedná se o součet řádků pro všechny testované operace dohromady, v případě, že by se operace opakovaly dvacetkrát. Výsledky jsou vidět v grafu (Graf 1), kde čím méně tím lépe, podrobnější výsledky je možné nalézt v příloze.
3. **Funkce** – jedná se o počet vybraných funkcí nástrojů, výsledný počet funkcí je vidět na grafu (Graf 2), podrobnější výsledky je možné nalézt v příloze.
4. **Náročnost nástroje** – reflektuje, jak náročná může být pro vývojáře práce s nástrojem při jednotlivých operací, také kolik času bude muset vývojář strávit při učení se s nástrojem, včetně počtu materiálů pro nástroj a jak je náročné řešení známých chyb pro tento nástroj.

Graf 1 - Počet řádků nutných na testované operace



Zdroj: vlastní zpracování

Graf 2 - Komparace počtu funkcí nástrojů



Zdroj: vlastní zpracování

Tabulka 6 - Bodové ohodnocení kritérií

Kritéria	ADO.NET	Entity Framework Core	Dapper
Výkon	10	1	9
Počet řádků	1	10	4
Funkce	4	7	4
Náročnost nástroje	7	5	7

Zdroj: vlastní zpracování

4.3.4 Váhy pro modelovou situaci A

Váhy kritérií v Saatyho metodě se vypočítají jako podíl mezi geometrickým průměrem jednotlivého kritéria a sumou geometrického průměru všech kritérií.

K této situaci byly stanoveny váhy kritérií pro Saatyho metodu následovně:

Tabulka 7 - Stanovení vah pro Saatyho metodu

Kritéria	Výkon	Počet řádků	Funkce	Náročnost nástroje	Geometrický průměr	Váhy
Výkon	1,00	5,00	7,00	7,00	3,956320998	0,658219201
Počet řádků	0,20	1,00	3,00	3,00	1,158292185	0,19270685
Funkce	0,14	0,33	1,00	0,33	0,354948106	0,059053261
Náročnost nástroje	0,14	0,20	3,00	1,00	0,541082269	0,090020688
CELKEM	-	-	-	-	6,010643558	1

Zdroj: vlastní zpracování

V tabulce (Tabulka 7) je vidět velký důraz na výkon, který má více než poloviční váhu a zároveň velmi malý důraz na funkce a náročnost nástroje, jelikož firma věří, že má dostatečně kvalitní vývojáře, aby si dokázali poradit, v případech, kdy práce s nástrojem není nejjednodušší.

Pro bodovací metodu jsou stanovené váhy následovně:

Tabulka 8 - Stanovení vah pro bodovací metodu

Kritéria	Body	Váhy
Výkon	10	0,5
Počet řádků	5	0,25
Funkce	2	0,1
Náročnost nástroje	3	0,15
Celkem	20	1

Zdroj: vlastní zpracování

4.3.5 Výpočet modelové situace A

Pro každé kritérium se sestaví Saatyho matice, ve které se porovnávají jednotlivé varianty. Výsledkem jednotlivých kritérií je vážený geometrický průměr.

V tabulkách (Tabulka 9 až Tabulka 12) je vidět postup výpočtu pro jednotlivá kritéria.

Tabulka 9 - Saatyho matice pro výkon

Kritérium výkonu	ADO.NET	EF CORE	DAPPER	Geometrický průměr	Vážený geometrický průměr
ADO.NET	1,00	9,00	2,00	2,620741394	0,589126932
EF CORE	0,11	1,00	0,13	0,240374928	0,054034841
DAPPER	0,50	8,00	1,00	1,587401052	0,356838227
CELKEM	-	-	-	4,448517375	1

Zdroj: vlastní zpracování

Tabulka 10 - Saatyho matice pro počet řádků

Kritérium Počet řádků	ADO.NET	EF CORE	DAPPER	Geometrický průměr	Vážený geometrický průměr
ADO.NET	1,00	0,11	0,33	0,333333333	0,065793742
EF CORE	9,00	1,00	7,00	3,979057208	0,785391188
DAPPER	3,00	0,14	1,00	0,753947441	0,14881507
CELKEM	-	-	-	5,066337982	1

Zdroj: vlastní zpracování

Tabulka 11 - Saatyho matice pro funkce

Kritérium Funkce	ADO.NET	EF CORE	DAPPER	Geometrický průměr	Vážený geometrický průměr
ADO.NET	1,00	0,33	1,00	0,693361274	0,2
EF CORE	3,00	1,00	3,00	2,080083823	0,6
DAPPER	1,00	0,33	1,00	0,693361274	0,2
CELKEM	-	-	-	3,466806372	1

Zdroj: vlastní zpracování

Tabulka 12 - Saatyho matice pro náročnost nástroje

Kritérium Náročnost nástroje	ADO.NET	EF CORE	DAPPER	Geometrický průměr	Vážený geometrický průměr
ADO.NET	1,00	3,00	1,00	1,44224957	0,428571429
EF CORE	0,33	1,00	0,33	0,480749857	0,142857143
DAPPER	1,00	3,00	1,00	1,44224957	0,428571429
CELKEM	-	-	-	3,365248997	1

Zdroj: vlastní zpracování

Optimální varianta se zjistí jako suma součinu vah a jednotlivého váženého geometrického průměru kritéria. Na základě těchto výsledků se vytvoří konečná tabulka, která zjistí optimální variantu jako sumu součinu vah a jednotlivých vážených geometrických průměrů. Z celkových výsledků se vytvoří pořadí, ve kterém největší číslo znamená nejlepší výsledek a nejmenší číslo nejhorší výsledek.

Tabulka 13 - Výsledná Saatyho matice pro situaci A

Celkem	Výkon	Počet řádků	Funkce	Náročnost nástroje	Součet hodnocení	Pořadí
ADO.NET	0,589126932	0,065793742	0,200000000	0,428571429	**0,45084451**	**1.**
EF CORE	0,054034841	0,785391188	0,600000000	0,142857143	**0,235209087**	**3.**
DAPPER	0,356838227	0,148815070	0,200000000	0,428571429	**0,313946403**	**2.**
Váhy kritérií	0,658219201	0,19270685	0,059053261	0,090020688	-	-

Zdroj: vlastní zpracování

Optimální varianta pro bodovací metodu se získá jako suma součinu vah a jednotlivého bodového ohodnocení kritérií z tabulky (Tabulka 5). Varianta s nejvyšším číslem je optimální variantou pro tuto situaci. Výsledky bodovací metody vypadají následovně:

Tabulka 14 - Výsledná tabulka pro bodovací metodu pro situaci A

Varianty	Body	Pořadí
ADO.NET	6,7	1.
EF Core	4,45	3.
Dapper	6,45	2.

Zdroj: vlastní zpracování

4.3.6 Váhy pro modelovou situaci B

Pro tuto situaci byly stanoveny váhy kritérií následovně:

Tabulka 15 - Stanovení vah pro Saatyho metodu

VÁHY	Výkon	Počet řádků	Funkce	Náročnost nástroje	Geometrický průměr	Váhy
Výkon	1,00	0,14	0,33	0,14	0,287190895	0,049656133
Počet řádků	7,00	1,00	5,00	3,00	3,201085873	0,553476969
Funkce	3,00	0,20	1,00	0,33	0,668740305	0,115627125
Náročnost nástroje	7,00	0,33	3,00	1,00	1,626576562	0,281239773
Celkem	-	-	-	-	5,783593634	1

Zdroj: vlastní zpracování

Zde je vidět velký důraz především na počet řádků nutných na testované operace a náročnost nástroje, z toho důvodů, aby vývoj administračního systému zabral co nejkratší možný čas.

Pro bodovací metodu jsou stanoveny váhy následovně:

Tabulka 16 - Stanovení vah pro bodovací metodu

Kritéria	Body	Váhy
Výkon	2	0,086956522
Počet řádků	10	0,434782609
Funkce	4	0,173913043
Náročnost nástroje	7	0,304347826
Celkem	23	1

Zdroj: vlastní zpracování

4.3.7 Výpočet modelové situace B

Postup je identický jako u výpočtu modelové situace A, a proto výpočty pro jednotlivá kritéria je možné nalézt v příloze. Zde je uváděna pouze konečná tabulka s výsledky:

Tabulka 17 - Výsledná Saatyho matice pro situaci B

Celkem	Výkon	Počet řádků	Funkce	Náročnost nástroje	Součet hodnocení	Pořadí
ADO.NET	0,589126932	0,065793742	0,200000000	0,428571429	0,20932584	**3.**
EF CORE	0,054034841	0,785391188	0,600000000	0,142857143	0,54693248	**1.**
DAPPER	0,356838227	0,148815070	0,200000000	0,428571429	0,24374168	**2.**
Váhy kritérií	0,049656133	0,553476969	0,115627125	0,281239773	-	-

Zdroj: vlastní zpracování

Výsledky pro bodovací metodu jsou:

Tabulka 18 - Výsledná tabulka pro bodovací metodu pro situaci B

Varianty	Body	Pořadí
ADO.NET	4,130435	3.
EF Core	7,173913	1.
Dapper	5,26087	2.

Zdroj: vlastní zpracování

5 Výsledky a diskuze

5.1 Modelová situace A

Z výsledků je patrné, že nejvýhodnější pro Firmu A bude zůstat u již vybraného ADO.NET, je to zapříčiněno především velkým důrazem firmy na výkon, kde se mu mohl rovnat pouze Dapper, nicméně ADO.NET je celkově rychlejší. Entity Framework Core vyniká především v doplňkových částech a počtu řádků pro testované operace, ale jeho výkon je oproti výše zmíněným velmi rozdílný, a proto skončil až na třetím místě.

V případě, že by se firma rozhodla akceptovat o trochu nižší výkon Dapperu a hodnotila by ho stejnou hodnotou jako ADO.NET, byl by Dapper nejlepším nástrojem pro firmu.

5.2 Modelová situace B

Z výsledků je patrné, že nejvýhodnější pro Firmu B je využít Entity Framework Core. Je to dáno velkým důrazem firmy na co nejpohodlnější vývoj administračního systému. Co se týče počtu řádků na testované operace, nemohl se mu žádný nástroj rovnat, nicméně u kritéria náročnost nástroje ho však oba zbylé nástroje předehnaly, protože prvotní nastavení Entity Framework Core je obtížnější než u zbylých dvou. Celkově byl Entity Framework Core nejlepší a za ním zůstal Dapper a na třetím místě ADO.NET.

5.3 Zhodnocení obou situací

U obou situací je vidět, že při výběru nástroje pro přístup do databáze také záleží na požadavcích firmy, zároveň všechny tři nástroje mají svá specifika. Pro nejpohodlnější vývoj je ideální použít Entity Framework Core zvláště v případech, kdy už s ním mají vývojáři zkušenosti, protože poté je vývoj aplikací velmi rychlý.

Pokud vývojář potřebuje co nejrychleji přistoupit k databázi pomoci jednoduché operace, je vhodný Dapper, který je schopen namapovat výsledky z databáze na objekty a práce s ním je velmi jednoduchá a přímočará.

Jestliže je kladen velký důraz na celkový výkon, je nejlepší ADO.NET, který přistupuje k databázi nejrychleji, ale je nutné poté výsledky z databáze mapovat ručně a je

s ním zdlouhavější práce. V tomto případě lze zvažovat i Dapper, jelikož rychlostně je velmi blízko ADO.NET a v některých případech ho překonává. Zároveň však zvládá automatické mapování. Nevýhodou Dapperu může být menší komunita kolem něj, menší dokumentace a je možné, že se narazí na problémy, s kterými si Dapper nemusí vědět rady.

6 Závěr

Hlavním cílem práce bylo analyzování efektivnosti různých nástrojů pro práci s databází na platformě .NET. Pro správné analyzování byla použita platforma .NET (kapitola č. **3.1)**, která byla zaměřena především na nejnovější .NET Core a jeho rozdíly oproti původnímu .NET Framework.

Dále byl představen přístup k databázi pomocí ORM v kapitole č. **3.3.1** a jednotlivých nástrojů, se kterými se dále pracovalo. Ke komparaci byly vybrány nástroje ADO.NET, Dapper a Entity Framework Core.

Ke správnému porovnání jednotlivých nástrojů byla vytvořena aplikace, která si kladla za cíl měření výkonu výše zmíněných nástrojů. Tato aplikace sloužila jako hlavní stavební kámen pro další hodnocení nástrojů. V aplikaci byla získána potřebná data ke správnému hodnocení nástrojů, a především výsledky měření výkonu jednotlivých nástrojů pro operace CRUD v synchronním, asynchronním způsobu přístupu do databáze a způsobu s procedurami. Testování všech způsobů bylo vybráno z důvodu co největší reflexe reálného využití nástrojů v praxi.

K porovnání jednotlivých nástrojů byly vytvořeny dvě modelové situace, které odpovídají praxi. Na těchto situacích bylo cílem ukázat, jak mohou být jednotlivé nástroje různě využitelné v závislosti na požadavcích buď firmy nebo vývojářů.

Pro správné porovnání byla využita vícekriteriální analýza variant, kde byly pomocí Saatyho metody a bodovací metody porovnávána jednotlivá kritéria vůči požadavkům firmy.

Dvě metody byly zvoleny z důvodu ověření správnosti výběru optimální varianty, jelikož je možné, že jedna metoda by mohla zobrazit výsledky rozdílně. Výsledky u obou metod jsou shodné a na jejich základě došlo k vybrání a doporučení optimální varianty pro každou z firem.

Pro firmu, která si klade za cíl vysoký výkon aplikace je optimální variantou ADO.NET a s minimálním odstupem Dapper. Pokud však firma klade důraz na rychlost vývoje aplikace je pro ni optimální variantou Entity Framework Core.

Výsledky této práce mohou posloužit firmám při výběru vhodného nástroje pro přístup do databáze na platformě .NET, jelikož ukazují využitelnost jednotlivých nástrojů v různých situacích, které mohou v praxi nastat.

7 Seznam použitých zdrojů

1. TROELSEN, Andrew a Philip JAPIKSE. *Pro C# 7: With .NET and .NET Core*. Eight Edition. New York: Apress, 2017. ISBN 978-1-4842-3017-6.
2. .NET Standard. *Microsoft Docs* [online]. 2018, 18.05.2018 [cit. 2018-07-06]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/net-standard
3. LANDWERTH, Immo. Introducing .NET Standard. *.NET Blog* [online]. 2016, 26.09.2016 [cit. 2018-07-06]. Dostupné z: https://blogs.msdn.microsoft.com/dotnet/2016/09/26/introducing-net-standard/
4. DUFFY, Joe. *Professional .NET Framework 2.0*. Indianapolis: Wiley Publishing, 2006. ISBN 978-0764571350.
5. POKORNÝ, Jan. *Úvod do .NET Framework*. Brno: Mobil Media, 2002. ISBN 80-86593-16-9.
6. Common Language Runtime (CLR) overview. *Microsoft Docs* [online]. 2018, 16.04.2018 [cit. 2018-07-06]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/standard/clr
7. Get started with the .NET Framework. *Microsoft Docs* [online]. 2018, 10. 04. 2018 [cit. 2018-07-06]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/framework/get-started/
8. NET Core Guide. *Microsoft Docs* [online]. 2016, 20.06.2016 [cit. 2018-08-05]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/core/
9. Choosing between .NET Core and .NET Framework for server apps. *Microsoft Docs* [online]. 2018, 19.06.2018 [cit. 2018-08-05]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/standard/choosing-core-framework-server
10. Introduction to ASP.NET Core. *Microsoft Docs* [online]. 2018, 28.02.2018 [cit. 2018-08-05]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/?view=aspnetcore-2.1
11. Dependency injection into controllers in ASP.NET Core. *Microsoft Docs* [online]. 2016, 14.10.2016 [cit. 2018-08-05]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/mvc/controllers/dependency-injection?view=aspnetcore-2.1
12. .NET Framework Versions and Dependencies. *Microsoft Docs* [online]. 2018, 25/07/2018 [cit. 2018-08-05]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/versions-and-dependencies
13. CONOLLY, Thomas, Carolyn E BEGG a Richard HOLOWCZAK. *Mistrovství - databáze: profesionální průvodce tvorbou efektivních databází*. Brno: Computer Press, 2009. ISBN 978-80-251-2328-7.
14. KROENKE, David a David J AUER. *Databáze*. Brno: Computer Press, 2015. ISBN 978-80-251-4352-0.
15. SINGH, Rahul Rajat. An Introduction to Entity Framework for Absolute Beginners. *CodeProject* [online]. 2012, 09.04.2012 [cit. 2018-08-20]. Dostupné z: https://www.codeproject.com/Articles/363040/An-Introduction-to-Entity-Framework-for-Absolute-B
16. HAMILTON, Bill a Matthew MACDONALD. *ADO.NET in a nutshell*. Sebastopol, Calif.: O'Reilly, 2003. ISBN 05-960-0361-7.
17. Home repository for .NET Core. *GitHub* [online]. [cit. 2018-08-22]. Dostupné z: https://github.com/dotnet/core/releases

18. LACKO, Ľuboslav. *Mistrovství v SQL Server 2012: [kompletní průvodce databázového experta]*. Brno: Computer Press, 2013. ISBN 978-80-251-3773-4.
19. LERMAN, Julia. *Programming Entity framework*. 2nd ed. Sebastopol: O´Reilly, c2010. ISBN 05-968-0726-0.
20. ŠUBRT, Tomáš. *Ekonomicko-matematické metody*. Plzeň: Vydavatelství a nakladatelství Aleš Čeněk, 2011. ISBN 978-80-7380-345-2.
21. SHAPOVALOV, Alex. Micro ORM vs ORM. *Yaplex* [online]. 2017, 21.06.2017 [cit. 2018-07-06]. Dostupné z: https://yaplex.com/blog/micro-orm-vs-orm
22. ABU GHAIDA, Saddam. Implement ORM using C#. *CodeProject* [online]. 2004, 04.12.2004 [cit. 2018-07-06]. Dostupné z: https://www.codeproject.com/Articles/849980/Implement-ORM-using-Csharp
23. VOGEL, Peter. High Performance Object-Oriented Data Access with Dapper. *Visualstudiomagazine* [online]. 2018, 22.03.2018 [cit. 2018-07-09]. Dostupné z: https://visualstudiomagazine.com/articles/2018/03/19/dapper-orm.aspx
24. KANJILAL, Joydip. How to use the Dapper ORM in C#. *InfoWorld* [online]. 2017, 12.10.2017 [cit. 2018-07-09]. Dostupné z: https://www.infoworld.com/article/3025784/application-development/how-to-work-with-dapper-in-c.html
25. TENNY, Lawrence J. a Zeeshan. HIRANI. *Entity framework 4.0 recipes: a problem-solution approach*. United States: Apress, c2010. ISBN 978-1-4302-2703-8.
26. Entity Framework Core Quick Overview. *Microsoft Docs* [online]. 2016, 27.10.2016 [cit. 2018-07-11]. Dostupné z: https://docs.microsoft.com/cs-cz/ef/core/
27. Entity Framework Loading Related Entities. *Microsoft Developer Network* [online]. 2016, 23.10.2016 [cit. 2018-07-11]. Dostupné z: https://msdn.microsoft.com/en-us/library/jj574232(v=vs.113).aspx
28. RIORDAN, Rebecca M. *Microsoft ADO.NET krok za krokem*. Brno: Mobil Media, c2002. iDnes internet knihy. ISBN 80-865-9320-7.
29. JOHNSON, Glenn. *MCTS self-paced training kit (exam 70-516): accessing data with Microsoft .NET Framework 4*. Redmond, Washington: Microsoft, [2011]. ISBN 978-0-7356-2739-0.

8 Přílohy

Příloha č. 1 – Počet řádků nutných na testované operace

Operace	ADO.NET	Dapper	Entity Framework Core
Create	9	3	2
Read	25	11	1
Update	7	3	2
Delete	6	3	2
Async Create	9	3	2
Async Read	24	11	1
Async Update	7	3	2
Async Delete	6	3	2
Proc Create	9	2	2
Proc Read	24	10	3
Proc Update	7	2	2
Proc Delete	6	2	2
DbContext	0	0	27
Celkem	139	56	50
Celkový počet řádků při 20 opakování	**2780**	**1120**	**487**

Zdroj: vlastní zpracování

Příloha č. 2 – Seznam vybraných funkcí

Funkce	ADO.NET	Dapper	Entity Framework Core
Transakce	ANO	ANO	ANO
Asynchronnost	ANO	ANO	ANO
Bulk operace	ANO	NE	NE
Migrace	NE	NE	ANO
Lazy loading	NE	NE	ANO
Generování SQL	NE	NE	ANO
Mapování entit	NE	ANO	ANO
Procedury	ANO	ANO	ANO

Zdroj: vlastní zpracování

Příloha č. 3 – Průměrné časy pro testované operace

SELECT				
	DAPPER	ADO.NET	EFCORE	**Celkem**
Synchronize	0:00:49,654	0:00:57,068	0:01:32,934	**0:03:19,656**
Asynchronize	0:00:52,480	0:00:59,238	0:02:12,071	**0:04:03,789**
Procedure	0:00:48,934	0:00:55,521	0:01:15,384	**0:02:59,839**
CELKEM	**0:02:31,068**	**0:02:51,827**	**0:05:00,389**	**-**
Pořadí	**1.**	**2.**	**3.**	**-**

Zdroj: vlastní zpracování

INSERT				
	DAPPER	ADO.NET	EFCORE	**Celkem**
Synchronize	0:00:00,354	0:00:00,294	0:00:08,741	**0:00:09,388**
Asynchronize	0:00:00,372	0:00:00,315	0:00:05,441	**0:00:06,128**
Procedure	0:00:00,363	0:00:00,292	0:00:00,974	**0:00:01,630**
CELKEM	**0:00:01,089**	**0:00:00,901**	**0:00:15,156**	**-**
Pořadí	**2.**	**1.**	**3.**	**-**

Zdroj: vlastní zpracování

UPDATE				
	DAPPER	ADO.NET	EFCORE	**Celkem**
Synchronize	0:00:00,413	0:00:00,285	0:00:00,842	**0:00:01,540**
Asynchronize	0:00:00,459	0:00:00,338	0:00:03,791	**0:00:04,587**
Procedure	0:00:00,400	0:00:00,284	0:00:00,980	**0:00:01,664**
CELKEM	**0:00:01,272**	**0:00:00,907**	**0:00:05,612**	**-**
Pořadí	**2.**	**1.**	**3.**	**-**

Zdroj: vlastní zpracování

DELETE				
	DAPPER	ADO.NET	EFCORE	**Celkem**
Synchronize	0:00:00,262	0:00:00,255	0:00:01,382	**0:00:01,900**
Asynchronize	0:00:00,282	0:00:00,277	0:00:01,511	**0:00:02,070**
Procedure	0:00:00,283	0:00:00,256	0:00:00,952	**0:00:01,492**
CELKEM	**0:00:00,828**	**0:00:00,789**	**0:00:03,845**	**-**
Pořadí	**2.**	**1.**	**3.**	**-**

Zdroj: vlastní zpracování

Příloha č. 4 – Saatyho matice pro modelovou situaci B

Kritérium výkonu	ADO.NET	EF CORE	DAPPER	Geometrický průměr	Vážený geometrický průměr
ADO.NET	1,00	9,00	2,00	2,620741394	0,589126932
EF CORE	0,11	1,00	0,13	0,240374928	0,054034841
DAPPER	0,50	8,00	1,00	1,587401052	0,356838227
CELKEM	-	-	-	4,448517375	1

Zdroj: vlastní zpracování

Kritérium Počet řádků	ADO.NET	EF CORE	DAPPER	Geometrický průměr	Vážený geometrický průměr
ADO.NET	1,00	0,11	0,33	0,333333333	0,065793742
EF CORE	9,00	1,00	7,00	3,979057208	0,785391188
DAPPER	3,00	0,14	1,00	0,753947441	0,14881507
CELKEM	-	-	-	5,066337982	1

Zdroj: vlastní zpracování

Kritérium Funkce	ADO.NET	EF CORE	DAPPER	Geometrický průměr	Vážený geometrický průměr
ADO.NET	1,00	0,33	1,00	0,693361274	0,2
EF CORE	3,00	1,00	3,00	2,080083823	0,6
DAPPER	1,00	0,33	1,00	0,693361274	0,2
CELKEM	-	-	-	3,466806372	1

Zdroj: vlastní zpracování

Kritérium Náročnost nástroje	ADO.NET	EF CORE	DAPPER	Geometrický průměr	Vážený geometrický průměr
ADO.NET	1,00	3,00	1,00	1,44224957	0,428571429
EF CORE	0,33	1,00	0,33	0,480749857	0,142857143
DAPPER	1,00	3,00	1,00	1,44224957	0,428571429
CELKEM	-	-	-	3,365248997	1

Zdroj: vlastní zpracování