FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Optimalizace výkonu OLTP pomocí in-memory OLTP technologií MS SQL
Student:	Vojtěch Carbol
Vedoucí:	Ing. Miroslav Prágl, MBA
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Na základě výkonových problémů se stávajícím OLTP reportingem:
- analyzujte problém, zjistěte úzká hrdla a požadavky uživatelů
- popište in-memory OLTP technologii MS SQL s důrazem na očekávaný přínos ve vašem případě
- zdůvodněte/vyberte vhodné "kandidáty", u kterých potřebujete / očekáváte zvýšeni výkonu / spokojenější uživatele díky zavedení této technologie
- navrhněte metodiku testování, technologii nasaďte a otestujte
- vyhodnoťte a diskutujte výsledky testu jak obecně, tak konkrétně pro tento případ
- vytvořte krátké "manažerské" shrnuti s popisem přínosů i nároků řešeni a jeho (ne)doporučením nasazení do ostrého provozu

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 6. února 2019

České vysoké učení technické v Praze
Fakulta informačních technologií
Department of software engineering

Bakalářská práce

Paměťová optimalizace databáze

Vojtěch Carbol

Vedoucí práce: Ing. Miroslav Prágl, MBA

14. května 2019

Poděkování

Chtěl bych poděkovat své rodině a pánům Miroslavu Práglovi a Davidu Hlaváčkovi za pomoc a psychickou podporu během psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

Prague dne 14. května 2019 ……

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Carbol, Vojtěch. *Paměťová optimalizace databáze*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce se zabývá optimalizací databází. Použitá technologie se nazývá Microsoft SQL Server In-memory OLTP. Konkrétně se jedná o změnu hlavního uložiště dat z disku do paměti. Jedná se jak o tabulky, tak o procedury. Na disk se ukládá pouze pro potřeby odolnosti dat při výpadcích/restartech serveru.

Klíčová slova Databáze, optimalizace, Microsoft SQL Server, In-memory, tabulky a procedury.

Abstract

This thesis is concerned with database optimization. Used technology is called Microsoft SQL Server In-memory OLTP. Specifically, it deals with migration of data from disk to memory. This includes tables as well as procedures. Storing to disk is used only for durability purposes in the event of server crash/restart.

Keywords Database, optimization, Microsoft SQL Server, In-memory, tables and procedures.

Obsah

Seznam obrázků

Úvod

Kapitola Úvod pojednává o významu této práce a o vytčených cílech.

Motivace

V minulém století, když byly databázové systémy vytvářeny, byla paměť ve větších velikostech nedostupná, proto bylo logicky rozhodnuto, že data je třeba ukládat na disk, a do paměti načítat pouze to, co se právě zpracovává. To ssebou ovšem nese nevýhody častého čtení a zapisování na disk.

V dnešní době je však paměť výrazně dostupnější, proto je na místě se zamyslet, jestli by nedávalo smysl klíčová data, nebo dokonce celé databáze ukládat přímo do paměti a tím výrazně zrychlit zpracování. Proto se v posledních letech na trhu začínají objevovat řešení, která umožňují,s různou mírou úspěšnosti a rozsahu, paměť využívat mnohem více.

Technologie použitá v této práci je In-memory OLTP od společnosti Microsoft. Jelikož v práci nejde o vytváření zcela nové databáze, ale pouze o optimalizaci již existující, nemá smysl zvažovat jiná řešení.

Cíl

Cílem rešeršní části práce je zaprvé popsat Microsoft SQL Server In-memory OLTP technologii a zadruhé provést analýzu problémové sekce firemní databáze a výběr vhodného úseku k In-memory optimalizaci. Cílem praktické části je transformovat vybraný úsek databáze do In-memory řešení. Následně otestovat jeho výkon a paměťovou náročnost. Nakonec pak provést analýzu výsledků testů a vytvoření doporučení pro případné nasazení nebo nenasazení této technologie.

Kapitola 1

Teoretické podklady

Tato kapitola je shrnutím základů použité technologie

1.1 Microsoft SQL Server

Microsoft SQL Server je systém pro vývoj a správu relačních databází, který podporuje velké množství aplikací pro zpracování transakcí, business intelligence, nebo analytickou činnost. Společnost Microsoft je jedním z největších hráčů na poli databázových technologií, vedle například společnosti Oracle s řešením Oracle Database a společnosti IBM s řešením DB2.

Základem systému je jazyk SQL, který je používán pro dotazování a úpravy v databázích. Microsoft SQL Server konkrétně využívá implementaci tohoto jazyka, nazývanou Transact-SQL (T-SQL). Tato implementace rozšiřuje standardní SQL například o lokální proměnné, procedurální programování, matematické funkce atd.

Microsoft SQL Server je primárně založen na řádkové struktuře tabulek, která umožňuje propojení souvisejících množin dat a tím zabraňuje zbytečnému ukládání dat na více místech v rámci databáze najednou. Relační model databáze mimo jiné zajišťuje vztahové a další integritní omezení, která udržují přesnost a integritu dat. Tato omezení jsou součástí obecného držení se zásad atomicity, konzistence, izolace a odolnosti (Společně známé jako ACID vlastnosti). Tyto čtyři zásady jsou formulovány tak, aby garantovaly, že databázové transakce probíhají spolehlivě a nepoškozují integritu databáze.

1.1.1 SQL Server Database Engine

Klíčovou součástí Microsoft SQL Serveru je tzv. SQL Server Database Engine, který spravuje ukládání, zpracování a bezpečnost dat. Obsahuje relační modul, který zpracovává příkazy a dotazy a úložný modul, který spravuje databázová data, tabulky, stránky atd. Uložené procedury, triggery a další podobné da-

tabázové objekty jsou vytvářeny a spouštěny SQL Server Database Engine modulem.

1.1.2 SQL Server Operating System

Ve vrstvě pod databázovým modulem je tzv. SQL Server Operating System (SQLOS), ten se stará o nízkoúrovňové operace jako je správa paměti, vstupy a výstupy nebo zamykání řádků tabulek pro zabránění konfliktním změnám. SQLOS zprostředkovává i interakci mezi uživatelem a databázovým serverem. Nakonec pak zpracovává i samostatné T-SQL příkazy uživatelů. Například pro vytváření, nebo změny v datech nebo strukturách.

1.2 Jazyk SQL

Structured Query Language je dotazovací jazyk (Query language), který byl vyvinut pro potřebu uživatelů databází na místě jednoduše prohlížet a měnit data pomocí dotazů. Základní dotazy zpravidla splňují tuto šablonu: [<co> <kde>]*. Například dotaz SELECT column1 FROM tblTable zobrazí všechny záznamy ve sloupci column1 v tabulce tblTable.

1.3 In-memory OLTP

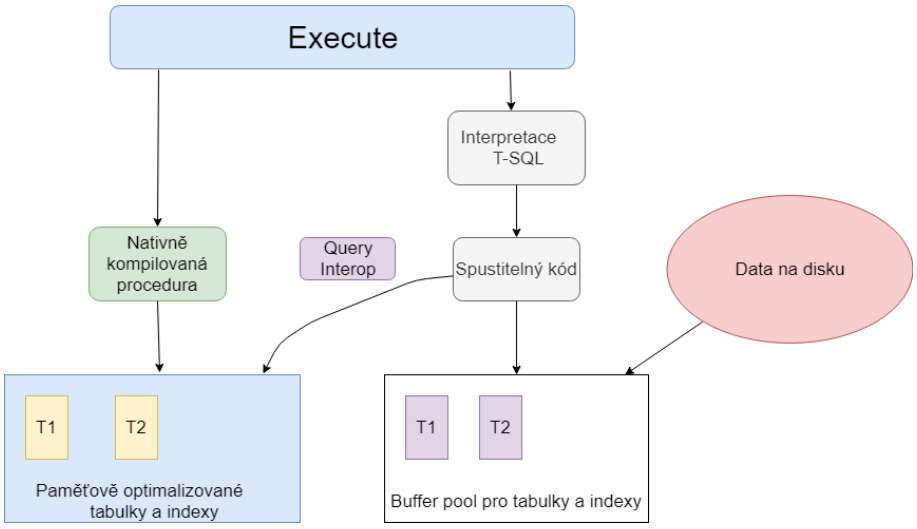
Relační databázové servery byly koncipovány s tím, že paměť je velmi limitována a tedy že data musí být uložena na disku a načtena až když jsou potřeba ke zpracování. V dnešní době je však paměť dostupná v dostatečné kapacitě, aby bylo možné ukládat a zpracovávat klíčová data přímo v ní. Z toho důvodu bylo vydáno i In-Memory OLTP pro nové verze Microsoft SQL Serveru.

Čtyři základní principy In-Memory OLTP:

- Přizpůsobení pro práci s daty, která jsou kompletně uložena v paměti a zároveň jsou odolná vůči výpadkům SQL Serveru (Obnovitelnost)
- Kompletní integrace do SQL Server enginu
- Vysoký výkon pro OLTP operace
- Uzpůsobení pro moderní CPU (Central Processing Unit)

Díky integraci do SQL Server enginu je umožněn přístup k in-memory datům pomocí standardních prostředků (T-SQL, SSMS). Samotné vnitřní chování je však zcela odlišné. In-memory OLTP navíc umožňuje efektivnější přístup k datům v podobě nativně kompilovaných procedur.

1.3.1 Typy procedur v Microsoft SQL



Obrázek 1.1: Triviální diagram typů procedur

Na levé straně diagramu jsou paměťově optimalizované tabulky a indexy, které jsou součástí in-memory OLTP, na straně pravé jsou disk-based tabulky, které používají datové struktury standardně používané SQL Serverem, tedy čtení a zápis úseků dat o fixní velikosti na a z disku.

V zeleném rámečku vidíme další z nových prvků v in-memory OLTP, a to nativně kompilované procedury. Jedná se o typ objektu zkompilovaného do strojového kódu. Tyto procedury umožňují další zrychlení výpočtu nad rámec pouhého použití paměťově optimalizovaných tabulek. Standardním ekvivalentem jsou interpretované T-SQL procedury, zobrazené vpravo, které při každém spuštění vyžadují režii navíc. Nativně kompilované procedury mohou však přistupovat pouze k datům v paměťově optimalizovaných tabulkách.

Komponenta Query Interop umožňuje interpretovanému T-SQL přístup k datům v paměťově optimalizovaných tabulkách. Pokud transakce přistupuje jak k paměťově optimalizovaným, tak k disk-based tabulkám, nazývá se cross-container transakcí.

Klientské aplikace přistupují k databázi na serveru pomocí tzv. TDS Handleru (protokol pro komunikaci s SQL serverem) bez ohledu na to, zda se dotazují na disk-based nebo na paměťově optimalizovaná data nebo na nativně kompilované či interpretované T-SQL procedury. Tímto je zajištěno, že paměťová optimalizace databáze nevynucuje změny ve vyšších vrstvách aplikační logiky.

1.4 Paměťově optimalizované tabulky

1.4.1 Uložení všech dat v paměti

Při přístupu do disk-based tabulek jsou data buď načtena v paměti, nebo musí být potřebné stránky dat načteny z disku. Tento proces načítání potřebných datových stránek do paměti se nazývá caching. Pokud v paměti dojde ke změně dat, ovlivněná datová stránka je uložena zpět na disk. Od tohoto základního předpokladu, že datové stránky jsou uloženy na disku a bude je nejdříve nutné načíst do paměti, se odvíjejí veškeré datové operace nad disk-based tabulkami. Procesy získávají zámky (locks), aby ochránily datové stránky v bufferu před změnami souběžnými procesy, a SQL Server nastavuje závory (latches) na data, která právě zapisuje nebo čte z disku. (Dle pesimistického modelu souběžnosti.)

V paměti je vždy uložena celá tabulka i její indexy. Proto při pokusu o přístup k paměťově optimalizovaným strukturám uživatelský proces vždy nalezne potřebná data v paměti. Souběžné operace na datech se nezasekávají na zamčených datech díky optimistickému modelu souběžnosti.

Při změnách v paměťově optimalizovaných tabulkách provádí SQL Server zápis o těchto změnách na disk pro případ výpadku/restartu serveru.

1.4.2 Řádkově orientované ukládání

Pro disk-based tabulky jsou data shlukována do 8 KB jednotek, které se nazývají datovými stránkami. Tyto datové stránky jsou základní jednotkou jak pro ukládání do paměti (bufferu), tak pro ukládání na disk. Jedna datová stránka může obsahovat data pouze z jedné tabulky nebo indexu. SQL Server průběžně přepisuje na disk nejprve log transakcí a pak i změněné datové stránky z paměti. Tento tzv. Checkpoint proces zapříčiňuje velká nahodilá množství čtení a zápisů na disk.

Pro paměťově optimalizované tabulky naopak žádné datové stránky neexistují. Používají se datové řádky zapsané sekvenčně do paměti v pořadí, ve kterém probíhají transakce, které je vytvořily, a každá obsahuje index (ukazatel) na další řádku v pořadí. Veškeré I/O operace pak probíhají ve formě procházení těchto struktur. To znamená, že zde nedává smysl princip, že nějaká data musí být zapsána na nějaké konkrétní místo, které má zrovna na starost nějaký objekt. Data však nejsou náhodně rozházena po paměti, každá paměťově optimalizovaná tabulka musí mít definován alespoň jeden index, dle kterého jsou pak její datové řádky SQL Serverem propojeny.

Datová řádka se skládá ze dvou částí, z hlavičky a pak z datové části, která obsahuje samotná data sloupců tabulky. Hlavička obsahuje identitu výrazu, který ji vytvořil, ukazatele na indexy cílové tabulky a klíčové timestamp hodnoty. Jmenovitě timestamp vložení a timestamp vymazání. SQL Server zaznamenává změny v datech vkládáním nových verzí řádek a označováním

starých pro vymazání. Skutečné promazávání nepotřebných verzí řádek probíhá průběžně spoluprací uživatelských vláken a vláken vyhrazených pro sběr odpadu (Garbage collection).

Z toho vyplývá, že v paměti existuje v jednu chvíli vícero verzí jedné řádky. Díky tomu je možný souběžný přístup k jedné řádce během modifikace dat, kdy SQL Server zobrazuje verzi řádky relevantní ke každé transakci dle času, kdy začala, a timestamp hodnotám dané verze řádky.

1.4.3 Nativně kompilované tabulky a indexy

Pojem nativní kompilace označuje proces konverze naprogramovaných konstruktů do nativního kódu. Nativní kód jsou procesorové instrukce, které jsou přímo spustitelné v CPU, bez jakékoli další kompilace nebo interpretace.

Krom tabulkových a indexových struktur jsou SQL Serverem v paměti drženy i DLL soubory (Dynamic Link Libraries neboli dynamické linkovací knihovny) pro přístup a změny v těchto strukturách. Při vytvoření paměťově optimalizované tabulky nebo indexu jsou přístupové rutiny k této nové struktuře zkompilovány do nativního kódu a uloženy do těchto DLL souborů. Existence těchto souborů je důvodem, proč při každé změně struktury tabulky musí být celá zahozena a vytvořena nová (včetně výše zmíněných DLL).

1.4.4 Nativní kompilace procedur

Pro práci s nativně kompilovanými tabulkami existuje možnost použít nativně kompilované procedury připravené v paměti v podobných DLL souborech, jako tabulky. Nativně kompilované procedury sestávají z procesorových instrukcí, které jsou přímo spustitelné v CPU bez další kompilace nebo interpretace. Zároveň nativně kompilované procedury budou na CPU vyžadovat výrazně méně instrukcí než ekvivalentní interpretovaná T-SQL procedura.

DLL soubory procedur a tabulek spolu přímo spolupracují, čímž je umožněn velmi efektivní průběh úkolu. Exekuční plán procedury je vytvořen velmi podobně jako u interpretovaného T-SQL, avšak kvůli tomu, že ne všechny prostředky T-SQL je možné u nativně kompilovaných procedur použít, má dotazový optimalizátor menší výběr čím plán postavit. Exekuční plán pro nativně kompilovanou proceduru je nakonec také zkompilován do DLL.

Existují však T-SQL konstrukty, které není možné v nativně kompilovaných procedurách použít. Navíc není možné přistupovat k datům v disk-based tabulkách přes nativně kompilované procedury. Pro přístup k oběma typům tabulek je možné použít pouze standardní interpretované T-SQL procedury. Dále, protože exekuční plán nativně kompilovaných procedur je vytvářen hned pří vytvoření procedury, není na rozdíl od interpretovaného T-SQL brán zřetel na hodnoty parametrů. Nakonec také nativně kompilované procedury nejsou automaticky rekompilovány při změně statistik tabulkových dat, ale rekompilaci je třeba v takovém případě vyvolat ručně.

1.5 Průběh transakcí

Každá databáze, která podporuje paměťově optimalizované tabulky, má dva základní čítače (countery):

- Transaction-ID counter – pomocí nějž je identifikována každá transakce
- Global Transaction Timestamp – určující timestamp kdy transakce provedla commit

Pokud je během transakce vytvořena řádka, je v paměti vytvořena nová verze řádky a její timestamp vytvoření nastaven na Transaction-ID této transakce. Pokud má být naopak řádka vymazána, je její timestamp vymazání nastaven na Transaction-ID transakce. Operace změna řádku je v podstatě vymazáním staré a vložením nové verze řádky.

Ve chvíli, kdy transakce úspěšně proběhla, provede commit. Poté je validována dle následujících kroků:

- Zkontrolovat změny provedené transakcí
- Vyčkat na vyřešení commit závislostí
- Zaznamenat transakci na disk
- Nastavit transakci jako validovanou
- Odstranit závislosti jiných transakcí.

Proběhne-li validace úspěšně, jsou v ovlivněných verzích řádek v paměti nastaveny relevantní timestampy na timestamp commitu transakce.

Dojde-li kdykoliv k problému v průběhu transakce nebo validace transakce, provede se roll-back, tedy všechny dosud provedené změny jsou navráceny do původního stavu.

1.5.1 Optimistický model souběžnosti

Systém předpokládá, že pokud transakce provedla commit, její validace proběhne úspěšně. Tedy, že změny, které provedla, budou k dispozici ostatním transakcím ve chvíli, kdy transakce provede commit.

Dle názvu je patrné, že tento model je optimistický, ne naivní, proto existují tzv. commit závislosti. Pro přehlednost bude objasněno příklady.

Transakce T1 změní řádku A a provede commit, transakce T2 pak chce řádku A přečíst. T2 jí přečte a provede commit. Než se však řádka uživateli zobrazí, musí být T2 validována. Protože však čte řádku, kterou vytvořila transkace T1, která není validována, musí čekat, až bude T1 dokončena. Po proběhnutí validace T1 je dokončena i validace T2 a uživateli se zobrazí řádka A. Zde je navíc poukázáno na řešení kolizí typu čtení-zápis.

Chtěla-li by T2 řádku A také změnit, bude SQL Server předpokládat, že T1 proběhne, odhalí kolizi typu zápis-zápis a T2 je okamžitě zrušena.

Jednalo-li by se o pesimistický model souběžnosti, který je používán na standardní disk-based tabulky, byla by T2 zablokována do ukončení validace T1 a následně podle výsledku buď spuštěna, nebo zrušena. Všechny problémy tohoto typu jsou obecně řešeny pomocí zámků a závor.

1.6 Indexy u paměťově optimalizovaných tabulek

Indexy u paměťově optimalizovaných tabulek slouží ke stejnému účelu jako indexy u disk-based tabulek. Fungují však rozdílně a jako tabulky, se kterými pracují, jsou celé pouze v paměti.

Na diskovém uložišti jsou řádky dat uloženy po stránkách, které se shlukují do jedné struktury. U paměťově optimalizovaných struktur však žádné takové stránky neexistují. Místo toho se řádky dat patřící jedné tabulce propojují pomocí indexů, proto musí každá paměťově optimalizovaná tabulka mít alespoň jeden index.

Na rozdíl od tabulek, změny v indexech, kromě columnstore indexů, nejsou logovány na disk ani ukládány v checkpointech a po výpadku systému jsou znovu sestaveny jako při vytvoření.

1.6.1 Nonclustered vs clustered index

Označení nonclustered se používá pro typy indexů, které nejsou přímo spjaty se samotnými daty, tedy, že klíčová hodnota v indexu obsahuje také ukazatel na místo, kde je příslušná řádka uložena. Oproti tomu clustered index je uložen přímo s řádky tabulky, tedy, že u klíčové hodnoty indexu jsou rovnou i příslušná data, která jsou tedy uložena v pořadí udávaném indexem. Jakákoliv tabulka by měla mít pouze jeden clustered index. Pokud by byly potřeba dva clustered indexy, bylo by potřeba udělat kopii dat v tabulce.

1.6.2 Typy indexů u paměťově optimalizovaných tabulek

1.6.2.1 Nonclustered hash index

Hash index je užitečný při hledání konkrétních klíčových hodnot. Je uložen jako standardní hashovací tabulka, tedy v podstatě pole hashovacích chlívečků, kde každý chlíveček ukazuje na místo v paměti, kde je nějaká řádka dat. Na indexové klíčové hodnoty v tabulce je použita hashovací funkce, která je přiřadí příslušným chlívečkům. Spadne-li více řádku do jednoho chlívečku, vytvoří z něj řetízek a každá ukazuje na další. Počet indexů je pevně nastaven uživatelem.

1.6.2.2 Nonclustered range index

Range index je vhodný pro hledání rozmezí klíčových hodnot. Je uložen ve speciální struktuře, nazývané Bw-strom, připomínajícím jak stavbou, tak metodami údržby klasické řadící stromy nebo haldy. Tato struktura je velmi podobná B-stromu, který se používá u disk-based tabulek. Skládá se z kořenové stránky, která se větví na další stránky, až konečně k listové úrovni. Na rozdíl od B-stromu mohou být stránky různé velikosti a nikdy se nemění, pouze se přidávají nové.

1.6.2.3 Clustered columnstore index

Columnstore index je používán pro reporting a analýzu velkých množství dat. Nedívá se na tabulku řádkově, ale sloupcově. Columnstore index se vyplatí použít v případě, je-li třeba nalézt, kolikrát se v tabulce v konkrétním sloupci objeví nějaká konkrétní hodnota. Columnstore index vytváří velmi efektivně zkomprimovanou kopii dat (přibližně 10% původní velikosti)

Použití columntore indexů však není kompatibilní s nativně kompilovanými procedurami a je umožněno pouze pomocí interop T-SQL procedur.

Columnstore index je jako jediný z indexů u paměťově optimalizovaných tabulek zálohován na disk.

1.6.3 Použití indexů

Je-li v indexovaných hodnotách mnoho duplicit nebo bude-li dotazováno na rozmezí hodnot, vyplatí se použít range index, jinak se obvykle používá hash index. Columnstore index je používán pouze ve specifických případech.

1.7 Odolnost a obnovitelnost dat

Je-li server restartován nebo pokud spadne, není možné přistupovat k datům, která byla před výpadkem v paměti. U standardních disk-based struktur toto není takový problém, neb data jsou uložena na disku. U paměťově optimalizovaných struktur jsou však všechna data uložena primárně v paměti. Proto SQL Server loguje operace nad paměťově optimalizovanými tabulkami na disk.

Logování In-memory dat je však výrazně efektivnější. Například se nezapisují změny v tabulkách jejichž transakce ještě neprovedly commit, a změny se nezapisují po jedné, ale vícero změn je zkombinováno do jednoho logového záznamu.

SQL Server také průběžně ukládá data z tabulek ve speciálních checkpoint souborech. Proto se používá zvláštní vlákno na pozadí, které zapisuje sekvenčně na disk.

Pro tyto typy souborů je pro databázi třeba definovat nové skupiny souborů na disku, do kterých budou ukládány.

V In-memory OLTP existuje možnost vytvořit neodolnou tabulku, u níž se na disk uloží pouze její struktura. Při práci s těmito tabulkami je přístup na disk minimalizován, data však nepřežijí výpadek.

1.7.1 Obnovení dat

Nejprve jsou nalezeny poslední checkpoint soubory, které jsou následně paralelně nahrány do paměti. Následně jsou znovu od timestampu checkpointu provedeny změny v logu transkací a tímto je databáze uvedena zpět do stavu před pádem.

1.8 Výkonnost

Kombinací speciálních datových struktur, indexů, eliminací zámků a závor a schopností vytvářet nativně kompilované procedury a funkce je umožněno výrazné zvýšení rychlosti výpočtů a operací s daty. Při použití neodolných tabulek a vhodných operací je možné dosáhnout i padesátinásobného zvýšení počtu odbavených požadavků za sekundu.

Kapitola 2

Analýza Konkurenčních řešení

Pro efektivní online zpracovávání transakcí (OLTP) existují na trhu i další řešení.

- Main-memory databáze (TimesTen Oracle, SolidDB IBM)
- Systémy pro Cache nebo sklady klíčových hodnot, které se snaží převést část práce na aplikace a do paměti střední vrstvy z databázového systému (App Fabric Cache, Gigaspaces)
- Některé databázové systémy jsou schopny pracovat s předkompilovanými procedurami, a rozmělnit práci i data do individuálních hardwarových vláken (VoltDB, MongoDB)

Většina konkurenčních řešení neposkytuje žádný ekvivalent cross-container transakcí, není tedy dobře možné paměťově optimalizovat pouze klíčové části databáze. Další například umožňují zápis do paměti po stránkách pevně dané velikosti.

Hlavní nevýhodou In-memory OLTP od společnosti Microsoft je prozatím nekompletní integrace do SQL server enginu. Některé funkcionality T-SQL jsou proto zatím v In-memory OLTP nedostupné (COALESCE, OFFSET, migrace dat z jiné databáze přímo do paměťově optimalizovaných tabulek…).

Firemní databáze je spravována pomocí Microsoft SQL Serveru, proto není možné zvolit jiné řešení. Konkurenční řešení jsou krátce uvedena jen pro úplnost.

Kapitola 3

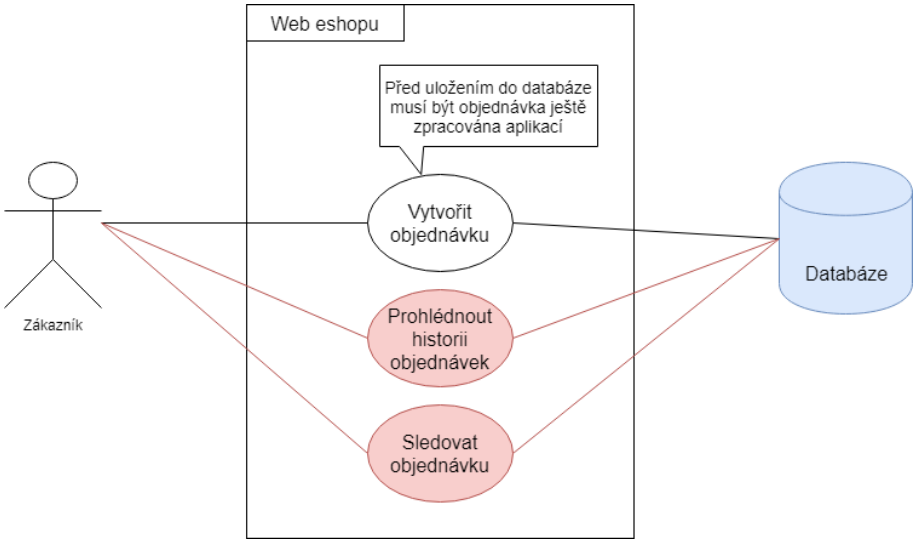
Analýza problémového úseku a současného řešení

Tato kapitola se nejdříve zabývá problémovým procesem a následně zkoumá jeho stávající řešení z pohledu databáze.

3.1 Problémový úsek

Databáze, kterou jsem se rozhodl zoptimalizovat je součástí Backend softwaru pro internetový obchod. Konkrétně má na starost archivaci objednávek v různých fázích zpracování. Data jsou z aplikační vrstvy posílána bezprostředně po provedení změny, vytvoření, nebo zrušení objednávky.

Právě tento úsek byl zvolen nejen proto, že jeho testovací běh s daty zaměstnanců firmy nedopadl uspokojivě, ale i protože se jedná o menší samostatný celek, což umožňuje převedení veškerých důležitých částí do paměti na současné konfiguraci serveru.



Obrázek 3.1: Use case diagram optimalizovaného úseku

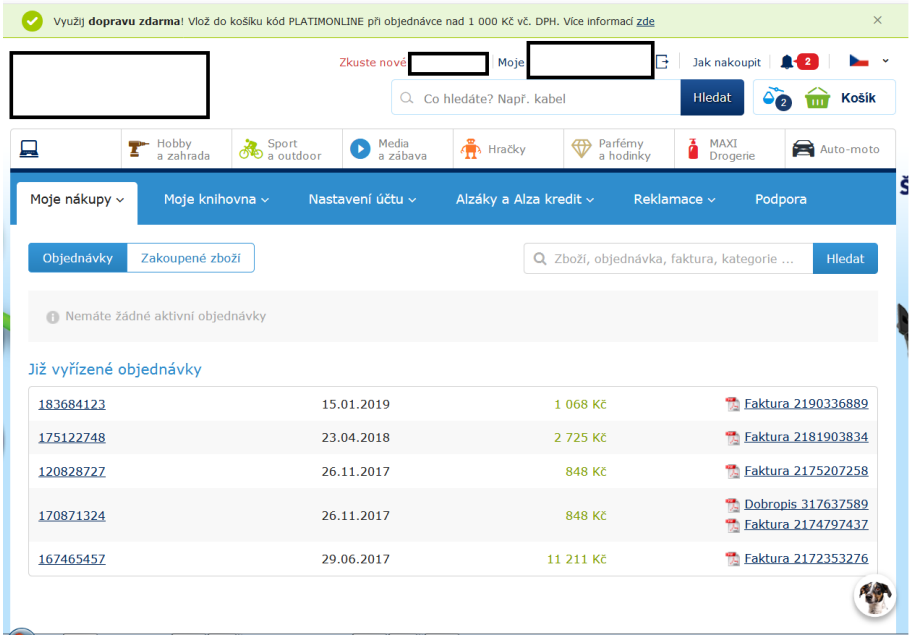
Zákazník nepřímo interaguje s databází v těchto případech:

- Objednání zboží
- Zobrazení historie objednávek
- Sledování stavu objednávky

Uložení nové objednávky do databáze nemá smysl optimalizovat, neboť úzkým hrdlem tohoto procesu je zpracování aplikací.

Prohlížení již vytvořených objednávek však není třeba tolik zpracovávat, a proto by se optimalizace tohoto procesu měla projevit.

Jinými slovy, optimalizován bude přístup k datům zobrazeným na stránce na následujícím obrázku.



Obrázek 3.2: Printscreen stánky historie objednávek

3.2 Současné řešení

Databáze obsahuje zejména tabulku s hlavičkami objednávek. Tato tabulka je rozhodně nejobjemnější a operace nad ní zatěžují systém nejvíce. Momentálně obsahuje množství řádků v jednotkách milionů a toto číslo se bude jen zvětšovat. Tato tabulka obsahuje například stav objednávky, kontakt a ID zákazníka, čas vytvoření atd.

Dalšími tabulkami jsou například tabulka položek objednávky, která obsahuje mimo jiné kód produktu, cenu a odkaz na výše zmíněnou hlavičku objednávky. Tabulka stavů objednávek obsahující přepínače, jaké informace se mají k objednávce zobrazit, a hlášky vyjadřující, že objednávka čeká na vyzvednutí nebo byla předána dopravci.

Klíčová procedura a zároveň procedura, na jejíž optimalizaci je kladen největší důraz přímo vrací data, která se buď pro konkrétní objednávku nebo pro všechny objednávky zákazníka zobrazí na webu.

Dosavadní řešení využívá standardní disk-based tabulky a procedury Microsoft SQL Serveru. Toto řešení jako hlavní uložiště požívá disk, proto není nijak výrazně náročné na kapacitu paměti, ale kvůli diskovým operacím je znatelně pomalejší, než by mohlo být po paměťové optimalizaci.

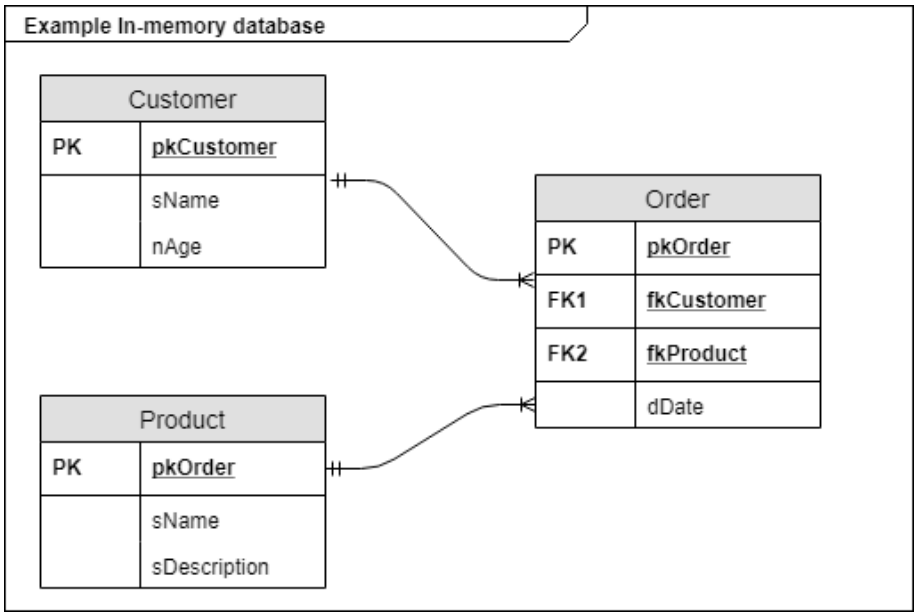
Kapitola 4

Testovací databáze

Před samotnou implementací bylo vytvořeno testovací protředí, na kterém se dalo bezpečně prozkoumat, jaké nové možnosti tato technologie nabízí.

Jelikož mi nebylo umožněno k bakalářské práci přiložit skutečné kódy na kterých jsem pracoval, rozhodl jsem se přidat alespoň databázi z tohoto testovacího prostředí na které bude základy zpracovávané technologie možné pozorovat.

Jedná se o databázi se třemi tabulkami a jednou procedurou, ve dvou verzích, jedna obsahuje standardní disk-based tabulky se standardní procedurou a druhá jejich paměťově optimalizované protějšky s nativně kompilovanou procedurou.

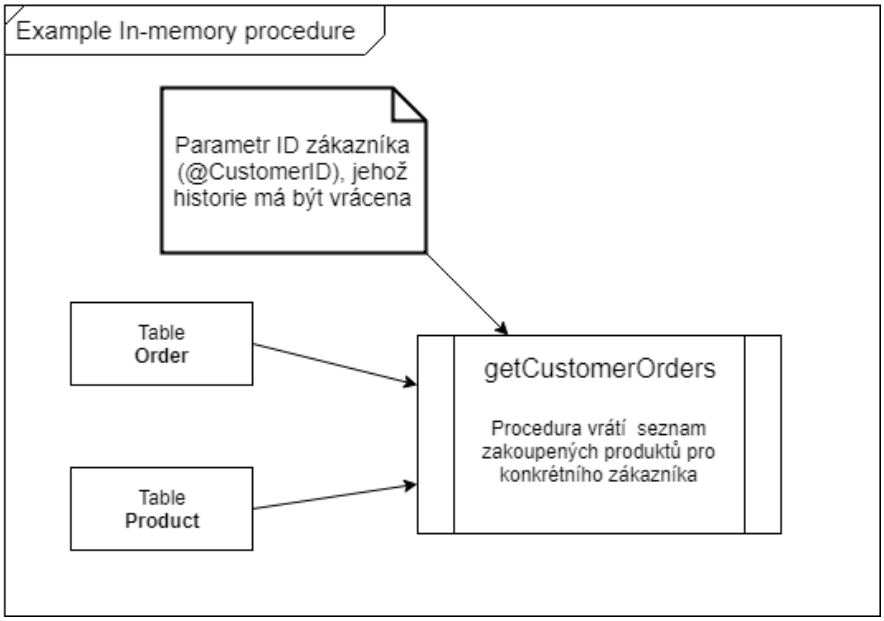


Obrázek 4.1: Diagram databáze

Toto schéma obsahuje tabulky a sloupečky, pro primární a cizí klíče je použit datový typ bigint, pro stringové sloupečky nvarchar, pro datum datetime a pro věk int.

Tabulka Customer obsahuje zákazníky a jejich osobní údaje. Tabulka Product obsahuje komodity, které je možné zakoupit. Tabulka Order představuje vazbu mezi komoditou a zákazníkem.

Všechny paměťově optimalizované tabulky používají hash index s počtem chlívečků 100 na primární klíč.



Obrázek 4.2: Diagram procedury

Procedura vybere název, popis produktu a čas nákupu pro všechny nákupy, které byly provedeny zákazníkem, jehož primární klíč je předáván paramtertem.

Byla-li by do tabulek vložena velká množství řádků, byl by mezi verzemi patrný výkonový rozdíl.

V příloze jsou k dispozici create scripty tabulek a procedur. Spustilené na Microsoft SQL Server ver. 2017.

V příloze nejsou připojeny konfigurační příkazy které by vytvořily potřebné filegroupy (Filegroups) na disku pro uložiště a zálohování.

Kapitola 5

Implementace

5.1 Použité nástroje

Pro provedení optimalizace byl použit program Microsoft SQL Server Management Studio, který byl vytvořen přímo pro práci s Microsoft SQL, tudíž byl jasnou volbou.

5.2 Postup optimalizace

Samotný postup proběhl v následujících fázích

Analýza kódu

Identifikace objektů k převedení do paměti. Hlavní procedura a všechny tabulky se kterými pracuje.

Převedení do paměti

Jelikož nativně kompilované procedury mohou pracovat pouze s tabulkami v paměti, musely být všechny tabulky se kterými se pracuje převedeny do paměti s prozatím výchozími indexy.

Následně byla převedena samotná procedura. Tento proces obsahoval převedení tabulkových proměnných a dočasných tabulek na tabulkové typy v paměti. Dále bylo třeba přepsat úseky kódu, kde se vyskytovaly funkcionality T-SQL nekompatibilní s In-memory enginem.

Zajímavým problémem byla například nemožnost použít funkci OFFSET, která umožňuje vynechat určené množství prvních výsledků dotazu. Tento problém byl po delším zkoumání vyřešen zavedením nové tabulkové proměnné, do které jsou výsledky přesypány v opačném pořadí, na což se pak dá použít funkce TOP, která propustí pouze určené množství prvních výsledků.

Testování

Jednoduché testy pro kontrolu ekvivalence výsledků původního a nového řešení.

Optimalizace

Dle potřeb dotazů v rámci procedury byly k tabulkám přidány indexy, nejčastěji hash indexy na množinu sloupečků v podmínce dotazu.

5.3 Shrnutí

Klíčové tabulky byly paměťově zoptimalizovány, tedy data a instrukce pro práci s nimi jsou kompletně a nastálo uloženy v paměti. Pro tabulkové proměnné byly vytvořeny paměťově optimalizované, funkcionálně ekvivalentní tabulkové typy.

Hlavní procedura vracející informace o objednávkách byla převedena na nativně kompilovanou, tedy zkompilována do strojového kódu a uložena do paměti pro rychlý přístup.

Kapitola 6

Testování

Použité nástroje

Pro testování a sledování vlastností jednotlivých řešení byly primárně použity nativní nástroje Microsoft SQL Server Management Studio, doplněné programem SQL Query stress, který umožňuje jednoduché spouštění testované procedury mnohokrát najednou.

Specifikace serveru

Jako disková uložistě jsou použity SSD disky. Paměť má kapacitu 2TB. Použitý model procesoru je Intel Xeon E7-8891 v4 operující na výchozí frekvenci 2,8GHz, obsahuje 10 fyzických jader a 2 vlákna na jádro.

Výkonnost

Jelikož výpočetní kapacita testovacího serveru výrazně překračuje požadavky testované procedury i při mnoha spuštěních najednou, nebylo ho možné tímto způsobem výrazně zatížit.

Jako vhodný reprezentativní test bylo zvoleno spuštění dvaceti iterací na padesáti vláknech, tedy celkem 1000 spuštění s různorodou, předdefinovanou množinou vstupních parametrů. Disková verze dosahovala v průměru času 6046,631 ns, zatímco In-memory 260,736 ns. Bylo tedy dosaženo přibližně 23násobného zrychlení.

Požadavky na úložný prostor

In-memory objekty zabíraly v paměti celkem 60 345 MB, a na disku 77 210 MB pro zálohy, včetně rezervovaného místa.

Diskové objekty zabíraly v paměti místo pouze operativně, byly-li zrovna používány, a na disku celkem 102,31 GB, včetně rezervovaného místa.

Rychlost obnovení

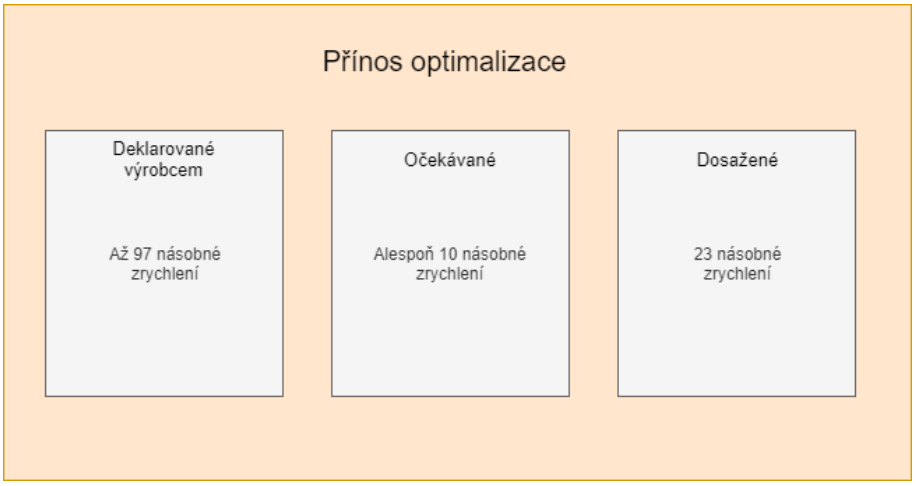
Bylo testováno pomocí prostého odpojení a připojení databáze na server.

Jelikož In-memory objekty je nejdříve nezbytné všechny obnovit ze záloh na disku, doba než je databáze opět k dispozici je u nich výrazně delší než u diskových objektů, které se do paměti před použitím ukládat nemusejí.

V testovaném případě trvalo obnovení In-memory databáze 56s, a diskové téměř okamžitě.

Kapitola 7

Analýza výsledku



Obrázek 7.1: Tabulka znázorňující výsledky primárního cíle práce

Jak je patrné z tabulky výše, hlavní cíl optimalizace, tedy zrychlení zpracování, předčil očekávání. Ke zrychlením deklarovaným společností Microsoft se však nebylo možné přiblížit, protože jich bylo dosaženo na úlohách k tomu zvláště vytvořeným (Malé transakce, žádné zálohování dat atd.).

Primárně je třeba zmínit 23násobné zrychlení optimalizovaného procesu. Jedná se sice jen o nepatrný kousek celého systému, ale možnost snazšího odbavení dotazů vrámci tohoto procesu by přispěla k nižší zátěži serveru a tedy k plynulejšímu provozu.

Díky interop komponentě by nejen nebylo třeba předělávat procedury pracující s nově optimalizovanými objekty, ale dokonce i ty by byly zrychleny díky výhodám In-memory OLTP popsaným v předchozích kapitolách.

Je však třeba brát v potaz, že toto řešení by trvale využívalo část paměti serveru, konkrétně by bylo rozumné předpokládat hodnoty v rozmezí 100-200GB. Tento objem by ovšem mohl za několik let výrazně narůst. Na druhou stranu je třeba připomenout, že produkční server má kapacitu paměti v řádu jednotek TB, kterou by navíc bylo možné několikanásobně rozšířit.

Dále je také nezbytné zmínit, že nutnost po výpadku serveru načíst veškeré In-memory objekty do paměti znatelně zvyšuje dobu obnovy serveru. Při nečekaném výpadku se však navíc musí i disková databáze synchronizovat například s logem transakcí a vezmeme-li v potaz jak malý úsek byl paměťově optimalizován, můžeme jeho negativní dopad na dobu obnovy v takovém případě téměř zanedbat.

Nakonec je třeba vzít v potaz i 30 až 40 hodin práce na optimalizaci a testování.

Shrnutí

Jelikož toto řešení nabízí prokazatelný přínos a produkční server má v současnosti více než dostatek paměti, je na místě toto řešení doporučit pro živý provoz.

Závěr

Cílem práce bylo nad vhodným úsekem firemní databáze provést paměťovou optimalizaci pomocí In-memory OLTP, otestovat tuto novou implementaci a zvážit její nasazení.

Tato optimalizace byla provedena pomocí technologie Microsoft SQL Server, následně byla otestována jak z funkčního tak z výkonového hlediska. Nakonec byla zvážena pozitiva i negativa tohoto řešení a bylo vysloveno doporučení pro jeho produkční nasazení. Cíl práce byl tedy splněn.

Do budoucna by toto řešení mohlo sloužit jako předloha pro paměťovou optimalizaci dalších klíčových úseků firemní databáze a tato práce jako celek by mohla napomoci při rozhodování, zda tuto možnost při vytváření nebo optimalizaci databází zvolit, či nikoliv.

Po nasazení tohoto znatelně rychlejšího řešení lze očekávat příznivé ohlasy od koncových uživatelů, ke kterým se budou informace dostávat rychle a efektivně i během například vánočních špiček.

Zdroje

Použitá literatura:

[1] Bolton, Ch., Langford J., Berry, G.: PROFESSIONAL SQL Server® 2012 Internals and Troubleshooting [book], John Wiley & Sons, Inc., 2013
[2] Delaney, K.: SQL Server Internals: In-Memory OLTP [book], Simple Talk Publishing, 2017
[3] Chmel, M., Mužný V.: SQL Server 2017 Administrator's Guide [book], Packt Publishing Ltd., 2017

Použitý software:

[4] Microsoft Corporation: Microsoft SQL Server Management Studio [software], dostupné z: `https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017`
[5] Pascal Brachet: Texmaker [software], dostupné z: `http://www.xm1math.net/texmaker/download.html`
[6] JGraph Ltd.: draw.io [software], dostupné z: `https://www.draw.io`
[7] Adam Machanic: SQL query stress simulator [software], dostupné z: `https://github.com/ErikEJ/SqlQueryStress`

Obsah přiloženého CD

- readme.txt - Stručný popis obsahu CD
- src - Složka obsahující LaTeX soubor a složku obrázků
- scripts - Složka obsahující skripty pro vytvoření a testování testovací databáze
- bachelors.pdf - Tato práce ve formátu PDF