TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: Informační technologie
Studijní obor: Informační technologie

Efektivní přístup k datům v rozsáhlé databázi měření elektrických veličin

Effective Data Access in Large Database of Electrical Quantities Measurements

Bakalářská práce

Autor: **Pavel Polívka**
Vedoucí práce: Ing. Jan Kraus
Konzultant: Ing. Pavel Štěpán

V Liberci 12. 5. 2011

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Abstrakt

Cílem práce je zefektivnění přístupu k datům v rozsáhlé databázi měření elektrických veličin, která je vytvořená na systému Microsoft SQL Server. Porovnání efektivnosti různých přístupových technologií. Navržení optimalizace struktury databáze pomocí indexace a komprese, případně úprava kódování.

Toto porovnání je realizováno pomocí testovací klientské aplikace pro prostředí Windows, vytvořené pomocí technologie WinForms obsažené v Micosoft .NET Frameworku.

Klíčová slova

Microsoft SQL Server, indexace, komprese, FILESTREAM, ENVIS

Abstract

This paper focuses on creating more effective ways of accesing data in large databse of electrical quantities measurements, witch is created on Microsoft SQL Server. Comparison of effectiveness of different access technologies. Optimization of database structure design by adding indexes and compresion. And suggests possible coding adjustment.

This comparison is realized by using a test client application for Windows created by using technology WinForms contained in Micosoft. NET Framework.

Keywords

Microsoft SQL Server, indexing, compression, FILESTREAM, ENVIS

Typografické konvence

V textu jsou použity různé styly textu a formátování, které pomáhají rozlišit různé typy informací:

- Pro normální text je použito běžné proporcionální patkové písmo.
- *Kurzíva* je používána v poznámkách, které obsahují doplňující informace k textu. Taktéž je užita v názvech ovládacích prvků, aplikací apod.
- Celistvé úryvky zdrojových kódů a výrazy ze zdrojových kódů, které se vyskytují v textu, je zvykem označovat `neproporcionálním písmem`.

Obsah

1 Úvod

Tato práce se zabývá efektivním přístupem k datům v rozsáhlé databázi měření elektrických veličin. Tato databáze je součástí programu ENVIS od firmy KMB Systems, který zajišťuje vizualizaci daných naměřených veličin. Práce s takto rozsáhlou databází, může být pomalá, proto tato práce hledá různé optimalizace.

Tyto optimalizace jsou z několika kategorií. První uvažovanou věcí jsou rozdílné technologie pro dotazování na databázi. Zde může být dopad na výkon výrazný, přeci jen každá technologie může generovat rozdílnou kvalitu SQL dotazů. Tato práce zpracovává pouze ORM frameworky (viz kapitola 2.1.). Další možností pro zlepšení výkonu jsou různé optimalizace struktury databáze, komprese dat a případné změny kódování. Tyto změny by neměly příliš zasahovat do současné struktury databáze, z důvodů zpětné kompatibility.

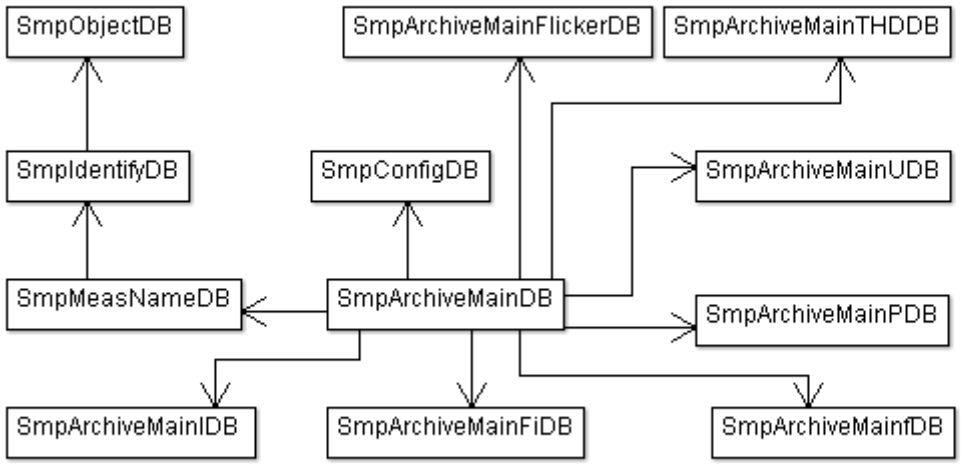
Optimalizace databáze je v této práci zaměřená na strukturu v databázi, která uchovává data pro přístroje SMV, SMP a SMPQ od společnosti KMB Systems.

U každé technologie, popřípadě navržené změny či optimalizace, je potřeba otestovat její reálný dopad na rychlost zpracování v ENVIS aplikaci. ENVIS však není stavěný na toto testování, proto za tímto účelem vznikla aplikace EnvisStressTest, která umí pracovat se všemi navrženými technologiemi. EnvisStressTest simuluje chování uživatele programu ENVIS v opakující se smyčce a zaznamenává čas, za který daná operace proběhla. Toto chování je z větší části konfigurovatelné. Následné exporty a zpracování zaznamenaných časů, umožňuje rozpoznávat, po patřičném statistickém zpracování, jak moc je daná technologie, optimalizace kódu či struktury, popřípadě komprese efektivní.

Na základě zhodnocení výsledků z aplikace EnvisStressTest u jednotlivých navrhovaných změn, tato práce navrhuje vhodná opatření pro další vývoj sytému ENVIS a jeho databáze.

1.1 ENVIS databáze

Na začátku práce je vhodné se seznámit se strukturou databáze, ve které jsou uchovávána všechna data.



Obrázek 1-1 Schéma databáze

První důležitou tabulkou je *SmpObjectDB* ve které jsou uchovávány informace o jednotlivých objektech, kde jsou přístroje použity. Tabulka obsahuje pouze celočíselný primární klíč a řetězec reprezentující jméno objektu. Samotné přístroje jsou ukládány v tabulce *SmpIdentifyDB* kde se kromě cizího klíče z tabulky objektů nachází takové informace jako výrobní číslo, číslo softwaru nebo přesný typ zařízení.

Další důležitou tabulkou je soubor jednotlivých měření, jenž je k nalezení v tabulce *SmpMeasNameDB*. Zde je každé měření označeno svým unikátním identifikátorem, je přiřazeno ke konkrétnímu přístroji a pro jednoduchou orientaci pojmenováno.

Velmi důležitou částí celé databáze je hlavní archiv (tabulka *SmpArchiveMainDB*). Hlavní archiv je v podstatě soubor referencí na tabulky obsahující jednotlivá měření. Například tabulka *SmpArchiveMainUDB* slouží k ukládání hodnot napětí. Jednotlivé řádky jsou identifikovány pomocí času měření (*keyTime*) a příslušnosti ke konkrétnímu měření (*keyMeasName*).

Jednotlivé řádky však nejsou hodnoty v přesně daném časovém okamžiku, ale jedná se o časový interval (omezený druhou časovou hodnotou *endTime*). V pomocných tabulkách jako je *SmpArchiveIDB* se však neukládají všechny hodnoty za daný časový interval, ale pouze jejich minimální, průměrná a maximální hodnota. Pro většinu vyhodnocení jsou tyto hodnoty dostačující. Kompletně celá data se ukládají do *varbinary(max)* sloupce *Data*, kde jsou uložená pomocí speciální komprese.

Jelikož do koncových tabulek se neukládají data v jednotkách, v nichž byla naměřena, kvůli rozdílným intervalům atd.,v tabulce *SmpConfigDB* se nachází informace pro následný převod na potřebnou jednotku.

Databáze obsahuje obdobné struktury i pro přístroje SIMON, elektroměr atd. Také zde je několik dalších tabulek, které ovšem přímo neovlivňují výkon databáze a jsou pro tuto práci bezvýznamné.

2 Přístup k datům

2.1 ADO.NET Entity Framework

ADO.NET je sada tříd a komponent, které se používají k přístupu k relačním datům a datovým servisům, čímž slouží k tvorbě aplikací používajících sdílená data. ADO.NET je součástí knihoven Microsoft .NET Frameworku a nachází se v *System.Data.dll*.

ADO.NET obsahuje poskytovatele dat (data providers) pro jednotlivé databázové systémy (MS SQL server, Oracle, atd.) Použití poskytovatelů je jednoduché, každý poskytovatel má třídu pro připojení (*Connection*), pro příkaz (*Command*) a pro čtení dat (*DataReader*). Jednotlivé databázové systémy se rozlišují jmenným prostorem a mírně odlišnými jmény jednotlivých tříd. Například pro Microsoft SQL Server je jmenný prostor *SqlClient* a jména tříd *SqlConnection*, *SqlCommand* a *SqlDataReader*. Pomocí *SqlConnection* se připojíme k databázi přes *SqlCommand* provedeme SQL dotaz a *SqlDataReader* nám slouží k následnému zpracování dat.

Nevýhodou tohoto přístupu k datům je, že výsledná data jsou relační. Pokud je tedy chceme použít v objektově orientovaném kódu, musíme je složitě převádět a upravovat. Tento problém řeší další velká součást ADO.NET (od verze 3.5) a to Entity Framework (dále jen EF).

EF patří do rodiny ORM (Objektově-relační mapování) nástrojů. Jeho úkolem je automatizovat konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. Toto je prováděno abstrakcí relačního (logického) schématu databáze do konceptuálního schématu, které následně používá aplikace. Tyto schémata specifikují entitní datový model (EDM). Každé databázové tabulce je přidělena třída (entita), ve které jsou datové složky namapované na jednotlivé atributy dané tabulky.

EDM u Entity Frameworku se skládá ze tří vrstev (souborů). Tyto vrstvy jsou:

- Logická
- Konceptuální
- Mapovací

Konkrétní entity jsou specifikovány pomocí konceptuálního schématu dat, kde jsou uloženy informace o jednotlivých objektech a vztazích mezi nimi. Pro tuto vrstvu se používá jazyk CSDL (Conceptual Schema Definition Language). Reálná databáze, její tabulky, vazby a procedury jsou popsané v logickém schématu pomocí jazyka SSDL (Store Schema Definition Language). Vztahy mezi dvěma předešlými vrstvami jsou popsány jazykem MSL (Mapping Schema Language) ve třetí mapovací vrstvě.

Visual Studio také nabízí Entity Model Wizard, který slouží k visuální tvorbě EDM a mapování. Výstupem tohoto nástroje je XML soubor (*.edmx). Soubor EDMX obsahuje všechna metadata, která EF potřebuje (CSDL,SSDL,MSL soubory). Tyto tři soubory mohou být vytvořeny i ručně.

Entity Model Wizard nejprve většinou generuje 1:1 mapování, ne však nutně, mezi databázovým schématem a konceptuálním schématem. V relačním schématu jsou prvky složeny z tabulek s primárními a cizími klíči, které spojují další tabulky. Na rozdíl od toho jsou entity definované jako konceptuální schéma dat.

V EF můžeme k datům v databázi přistupovat dvěma způsoby:

2.1.1 EntityClient

EntityClient je v podstatě poskytovatel dat, jak jsou popsány výše, ale na rozdíl od ostatních (*SqlClient*, *OracleClient*) se nedotazuje proti koncové databázi, ale proti entitnímu modelu. To sebou nese určité změny, protože *EntityCommand* nevrací relační data, nýbrž entity. Proto nepoužíváme standardní SQL, ale speciálně upravenou verzi Entity SQL. Která například nepodporuje konstrukci `join`, EF umí získat data i bez ní.

2.1.2 Linq to Entities

Druhý elegantnější a rozhodně používanější způsob je pomocí Linq to Entities (dále jen LTE). LTE patří do rodiny Linq (Language Integrated Query), což je sada rozšíření do .NET Frameworku , které zahrnují dotazy integrované do jazyka, nastavení a transformační operace. Linq je součástí jazyků C# a Visual Basic. Pomocí Linq se programátor jednotnou syntaxí (v rámci jednoho jazyka) může dotazovat na širokou škálu datových úložišť. V základu máme k dispozici hned několik variant:

- Linq to Objects – dotazy nad objekty v aplikaci, k dispozici vždy
- Linq to XML – dotazy nad XML daty, objektově
- Linq to SQL – dotazy nad SQLserverem, ORM (více dál)
- Linq to DataSet – dotazy nad DataSetem, pomocí ADO.NET
- Linq to Entities – dotazy nad EDM

Díky možnosti implementovat Linq nad libovolný zdroj vzniká spousta implementací i mimo Microsoft. Například Linq to Google, Linq to Oracle nebo Linq to Amazon.

Sytaxe Linq je velmi podobný syntaxi SQL, opět zde ale máme různě rozdíly. Podstatné je, že Linq se drží syntaxe jazyka ve kterém je použit. V C# tedy musíme dodržovat velká a malá písmena (všechny příkazy jsou malými) zatímco Visual Basic se drží své syntaxe (první písmeno velké).

2.1.3 Předgenerování pohledů

Předtím než EF může provést dotaz proti konceptuálnímu modelu, nebo uložit jakékoliv změny do datového zdroje, musí vygenerovat lokální pohledy (views) k přístupu k databázi. Pohledy jsou součástí metadat a jsou ukládány do mezipaměti dané aplikace. Pokud vytvoříme více instancí objektu datového zdroje, použijí se znovu pohledy z mezipaměti (nemusí se znovu generovat). Protože generování pohledů zabírá velmi podstatnou část času vykonání dotazů, umožňuje nám EF si pohledy na daný EDM předgenerovat a přiložit je ke kompilovanému projektu.

Za předgenerování těchto pohledů je zodpovědný stejný nástroj jako za generování celého EDM. K tomu aby se pohledy předgenerovaly se musí ve vlastnostech projektu v záložce *Build Events* do textového pole *Pre-Build Event Command Line* přidat následující kód specifikující přesnou verzi generátoru, cílový adresář a cílové jméno souboru. Přesný postup je v [7].

Po sestavení projektu, program *EdmGen.exe* do adresáře projektu vygeneruje soubor s pohledy, ten stačí následovně přidat do projektu.

2.1.4 Výhody a nevýhody EF

Výhody

- Větší míra abstrakce nad daty, nemusíme se již starat o práci s relačními daty, veškerá práce s nimi probíhá v Entitním modelu
- Pohodlná práce s EDM ve vlastní aplikaci
- Pohodlný, intuitivní a plně automatický generátor Entitního modelu
- Podpora více databázových systémů (SQLServer, Oracle atd.). Při změně EDM do jiného systému, nemusíme měnit kód pro práci s daty v aplikaci
- Velmi obsáhlá a dostupná dokumentace
- Dokáže vylepšovat výkon (rychlost dotazů) za běhu

Nevýhody

- Bez předgenerovaných pohledů velmi pomalé (u větších databází).
- Generátor EDM nepojmenovává cizí klíče podle jejich jména v databázi, ale jen je čísluje za sebou. U větších databází velmi nepřehledné, musí se ručně přejmenovat
- Každá tabulka v databázi musí kvůli generátoru obsahovat primární klíč, i když třeba není potřeba

2.2 Linq to SQL

Linq to SQL (dále jen LTS) je další ORM nástroj od Microsoftu, zároveň (jak již název napovídá) spadá do rodiny Linq nástrojů. Jde o jednoduchý nástroj, který překládá Linq dotazy na model databáze do T-SQL jazyka pro Microsoft SQL server.

Rozhodně se nejedná o komplexní nástroj jako je EF, hlavní rozdíl je především v mapování. LTS nemá žádný konceptuální model, vždy mapuje jeden objekt na jednu tabulku. Výsledkem mapování je tedy něco jako 1:1 model. Jakákoliv změna v databázi musí být provedena i na modelu, žádný objekt nemůže mít namapováno více tabulek apod. LTS navíc umí mapovat pouze tabulky, zatímco EF zvládá i replikace, reporty a spousty dalších. Model se ukládá pouze do jednoho souboru (*.dbml*).

Visual Studio obsahuje i velmi jednoduchý editor LTS modelů, problém editoru je ovšem v tom, že není automatizovaný. To znamená, že i když chceme 1:1 model databáze, musíme každou tabulku přetáhnout ručně. Je to zdlouhavé a nepraktické.

2.2.1 Výhody a nevýhody LTS

Výhody

- Abstrakce nad daty
- Velmi jednoduché na použití
- Dostupná dokumentace

Nevýhody

- Pouze 1:1 model, žádný konceptuální
- Pomalejší dotazy
- Nevylepšuje svůj výkon za běhu
- Podpora pouze MS SQL Serveru
- Složitější generace modelu
- Ukončený vývoj, Microsoft se plně soustředí na EF

2.3 eXpress Persistants Objects

Další ORM Framework použitý v této práci je eXpress Persistants Objects (dále jen XPO) od společnosti DevExpress. Tento Framework byl do testů zařazen, jelikož ho používá současná aplikace.

Mapování dat probíhá přes takzvané persistentní objekty, které reprezentují jak objekty používané v samotném kódu aplikace tak i pokyny k mapování na konkrétní databázi. Tyto objekty se dají tvořit ručně, nebo se pomocí integrovaného průvodce dají vygenerovat.

Přístup k datům v XPO není řešen pomocí Linq nebo nějaké mutace SQL, ale vlastní, poměrně komplexní, sadou kolekcí a objektů.

2.3.1 Výhody a nevýhody XPO

Výhody

- Abstrakce nad daty
- Jednoduchý princip mapování
- Podpora více databázových systémů (SQLServer, Oracle atd.). Při změně EDM do jiného systému, nemusíme měnit kód pro práci s daty v aplikaci
- Rychlé dotazy

Nevýhody

- Persistentní objekt může obsahovat pouze jeden index, více indexů se řeší jako struktura obsahující všechny indexy. Generátor, tabulky s více indexy ani nebere v potaz, musí se dopisovat ručně
- Složitější a neintuitivní architektura dotazů
- Součást placeného balíčku od třetí strany
- Nedostupná a málo vysvětlující dokumentace

3 Použité technologie

3.1 Komprese

SQL server 2008 poskytuje dva druhy komprese, komprese zálohy a komprese dat. Komprese zálohy přímo neovlivňuje výkon databáze za běhu, není pro nás tedy zajímavá. Datová komprese byla prvně představena v SQL serveru 2005 SP2 pro datové typy *vardecimal* (proměnná délka numerických dat). V SQL serveru 2008 je nyní datová komprese k dispozici pro fixní datové typy (integer, float, char) i nějaké další. Konkrétní rozpis je k nalezení v [8]. Hlavní výhodou datové komprese je, že redukuje místo na disku, jenž databáze zabírá a zlepšuje výkon výstupních dotazů (menší datové přenosy, menší obsazení paměti RAM). Nevýhoda spočívá ve větší vytíženosti CPU.

SQL server 2008 podporuje dva druhy datové komprese, řádkovou (ROW) a stránkovou (PAGE). Komprese může být použita na následujících databázových objektech:

- Tabulky bez indexů (Heap Table).
- Tabulky s clustered indexy.
- Tabulky s nonclustered indexy.
- Indexované pohledy.
- Rozdělené tabulky i indexy (musí být nadefinováno pro každou část).

3.1.1 ROW komprese

Řádková datová komprese prakticky ukládá všechny datové typy s fixní délkou jako typy s proměnou délkou. Pokud povolíme řádkovou kompresi, změna se projeví pouze na fyzickém uložení daného datového typu, není potřebná žádná změna na úrovni aplikace. Konkrétní rozpis efektů na jednotlivé datové typy v [8].

Jako příklad si můžeme uvést tabulku, co má sloupec s datovým typem *CHAR(100)*. Pokud do něj uložíme jakékoliv množství znaků (do 100), vždy se fyzicky uloží jako 100 znaků. Například hodnoty *(„Test komprese“)* a *(„Test řádkové komprese“)* se uloží jako sto znaků. Pokud ovšem povolíme řádkovou kompresi, uloží se první hodnota jako 13 znaků (což je 87% úspora) a druhá hodnota jako 21 znaků (úspora 79%). Nulové a *NULL* hodnoty se neukládají vůbec.

3.1.2 PAGE komprese

Stránková datová komprese minimalizuje místo zabrané redundantními hodnotami v jednotlivých sloupcích na stránce tabulky. Dosahuje tím, že redundantní data uloží pouze jednou a v jednotlivých sloupcích na ně poté akorát ukazuje. Úspora tedy záleží na množství

redundantních dat, a jestli jsou ty data větší než následný ukazatel (jednička v integeru při ROW kompresi, nemusí být větší než ukazatel). Při použití stránkové komprese se automaticky použije i řádková.

Jako příklad si můžeme uvést následující tabulku:

Zdrojový kód 3-1 SQL tabulka zaměstanenc

```
Table zamestanec (jmeno varchar(100),pracovni_pomer varchar (10) default 'plny')
```

Pokud do této tabulky vložíme více řádků, většina z nich díky klíčovému slovu *default* bude mít jako *pracovni_pomer* vyplněno *plny*. Pokud zde použijeme stránkovou kompresi, hodnota *plny* se uloží jen jednou.

3.1.3 Práce s datovou kompresí

SQL Server obsahuje vnořenou proceduru sp_estimate_data_compression_savings, která přehledně zobrazuje přibližnou úsporu místa, kterou nám ta či ona komprese poskytne.

Zdrojový kód 3-2 Vložená procedura pro praci s kompresí

```
GO
EXEC sp_estimate_data_compression_savings 'Prodeje', 'DetailObjednaky', NUL
L, NULL, 'ROW' ;
GO
```

Parametry jsou:

1. Jméno schématu.
2. Jméno objektu.
3. Index ID.
4. Index partitionu.
5. Typ komprese.

Pro aplikaci komprese obsahuje SQL Server Managment Studio jednoduchého průvodce, který se zobrazí po kliknutí pravého tlačítka na požadovanou tabulku.

3.2 FILESTREAM

Pokud chceme ukládat nestrukturovaná data, jako jsou obrázky, textové soubory nebo videa máme dvě základní možnosti. První je uložit data mimo databázi, zatímco druhá je ukládat přímo do databáze. První možnost sebou může nést různé bezpečností a výkonové problémy, nehledě na následnou přehlednost kódu. Druhá možnost všechny tyhle problémy eliminuje, ale pokud jsou soubory příliš velké, může nepříjemně zpomalovat výkon samotné

databáze (databázový soubor je příliš velký). Právě pro případy velkých souborů Microsoft do SQL Serveru 2008 implementoval mechanizmus FILESTREAMu.

FILESREAM integruje databázový engine SQL Serveru spolu s NTFS souborovým systémem. Data (BLOBy) normálně ukládané do *varbinary(max)* se ukládají jako samostatné soubory v souborovém systému. Můžeme je vkládat, editovat a mazat pomocí T-SQL. Pro čtení můžeme použít nejen T-SQL, ale i Win32 rozhraní pro čtení souborů. Jelikož je FILESTREAM je definován jako databázový datový typ *varbinary(max) v*elikost daných dat je omezená pouze velikostí oddílu na disku, standardní varbinary omezení na 2GB se zde neaplikuje.

Při zpracování dotazů jsou data z FILESTREAMu ukládána v cache paměti systému, takže databázový buffer zůstává nepoužitý, což dává více prostoru pro zpracování dotazu.

3.2.1 Používání FILESTREAMu

Předtím než může FILESTREAM začít používat, musíme ho povolit na konkrétní instanci databázového enginu. K tomu slouží vnořená procedura *sp_configure*. Přesný návod je k nalezení v [10].

Následně při vytváření databáze musíme specifikovat, kterou skupinu souborů budeme používat a kam přesně se budou data ukládat. Přesný návod je k nalezení v [11].

Pro následnou práci s daty můžeme použít T-SQL, kde s FILESTREAMem pracujeme jako se standartním *varbinary* typem. Návod v [12]. Nebo T-SQL může vrátit pouze ukazatel na soubor s kterým se poté pracuje pomocí standardních win32 funkcí. Návod v [13].

3.3 Indexace

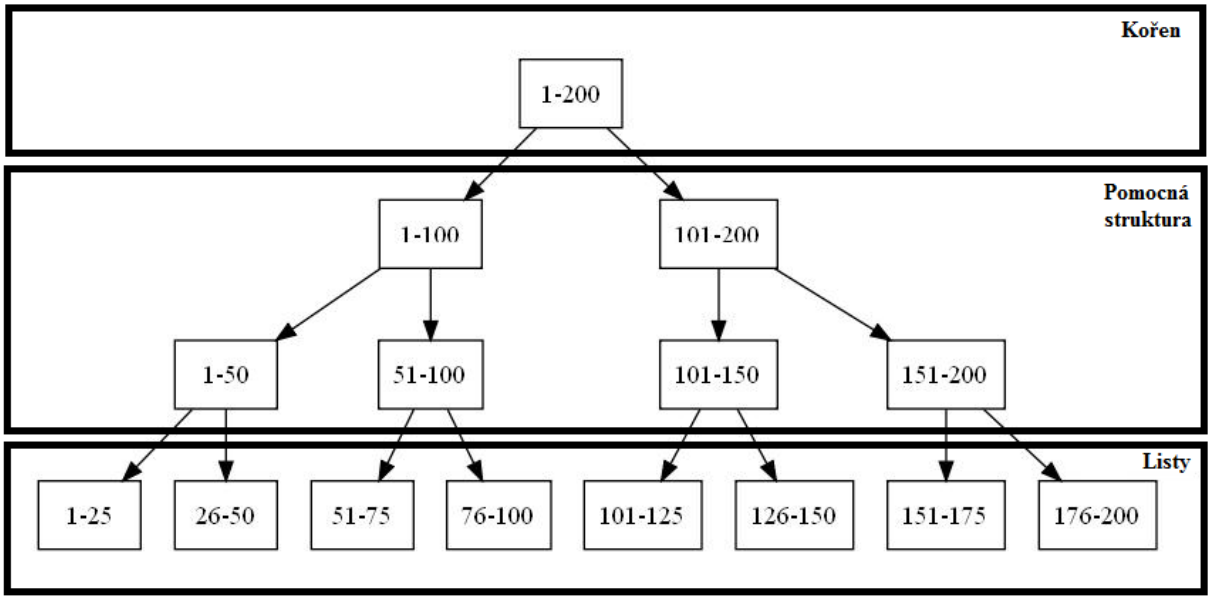
Indexace dat uložených v databázi může znatelně zrychlit procházení databází, tím že posykují mnohem rychlejší přístup ke konkrétním řádkům v tabulkách.

Indexy se vytváří na sloupcích v tabulkách nebo pohledech, indexy následně slouží k rychlejšímu vyhledání dat v daném sloupci. Pokud například máme index na primárním klíči a hledáme specifickou řádku, SQL server nejprve najde námi hledaný primární klíč v indexu a podle něho celou řádku. Pokud by zde index nebyl, SQL server by procházel tabulkou řádek po řádku.

Indexy mohou většinou být vytvořeny na všech sloupcích tabulky, výjimkou bývají jen sloupce s datovými typy pro větší data (*image*, *text*, *varbinary*).

Po vytvoření indexu se jeho data uspořádají do hierarchické stromové struktury, která usnadňuje následné procházení.

Při zpracování dotazu na oindexovaný sloupec, SQL server začne procházet index od kořene přes pomocnou navigační strukturu. Každou další vrstvou se vyřazuje polovina zbylých řádků. Server takto pokračuje, dokud nedojde k takzvanému listu, který už neobsahuje velké množství dat a není náročné ho projít celý.



Obrázek 3-1Struktura indexu

3.3.1 Druhy indexů

3.3.1.1 Clustered indexy

Clustered indexy ukládají všechna data na řádku do listu. Hodnoty ve sloupci mohou být setříděny sestupně nebo vzestupně. Každá tabulka může mít pouze jeden clustered index. Tabulky s clustered indexem jsou označovány jako clustered tabulka (clustered table), tabulky bez něj jsou označovány jako heap (hromada).

3.3.1.2 Nonclustered indexy

Na rozdíl od clustered indexů, nonclustered indexy do listů ukládají pouze hodnotu daného sloupce a referenci na daný řádek v tabulce. Což pro SQL server znamená jeden krok navíc. Nonclustered indexy nepodporují třídění. Na každé tabulce může být více nonclustered indexů (SQL server 2005 podopruje 249 a SQL server 2008 podporuje 999 nonclustered indexů).

Při vytváření nonclustered indexu můžeme pomocí příkazu *include columns* přidat další sloupce co se budou ukládat přímo do indexu, čím můžeme zrychlit specifické dotazy.

3.3.2 SQL Server Profiler

SQL Server Profiler je nástroj, běžící na serveru spolu s SQL serverem. Který poskytuje množství informací, které můžou být použity k vyřešení různých problémů v databázi, nebo následné generování indexů. Uživatel nastaví Profiler, aby zachycoval různé události (event), které chce monitorovat. Mezi tyto události patří třeba T-SQL dotazy, chyby nebo vložené procedury. Samotný nástroj obsahuje mnoho předdefinovaných šablon pro různé typické situace (tuning, error report, atd.).

Data, která zachytí, může ukládat v textovém souboru nebo do SQL tabulky. Aby se neovlivňoval příliš výkon běžící databáze, doporučuje se ukládat na jiný počítač, nebo alespoň do jiné instance SQL serveru. Data se zachycují a ukládají v podobně SQL Trace specifikace [14], která může být generovaná i mimo SQL Profiler přímo v klientské aplikaci.

Kompletní návod na SQL Profiler je k nalezení ve zdroji [15].

3.3.3 Database engine tuning advisor

Microsoft SQL Database engine tuning advisor pomáhá vybrat a vytvořit optimální sadu indexů, indexovaných pohledů a oddílů. Toho dosahuje analýzou zátěže a fyzické implementace jedné nebo více databází. Zátěží je myšlen soubor T-SQL dotazů proti databázi nebo databázím, které jsou laděny. Můžou být použity trace soubory, trace tabulky nebo samostatné T-SQL skripty. Zátěžové T-SQL skripty mohou být vytvořeny v dotazovém editoru management studia. Trace soubory nebo tabulky pomocí SQL Server Profileru za použití šablony *Tuning Template*.

Po analýze zátěže a databáze může Tuning advisor doporučit přidání, modifikaci nebo smazaní nějaké fyzické struktury v databázi. Může doporučit, které statistiky by měli být sbírány kvůli záloze. A může designovat nové fyzické struktury, jako jsou clustered indexy, nonclustered indexy, indexované pohledy nebo nové oddíly. Navržené změny by měly redukovat dobu provádění dotazů zahrnutých v zátěži.

Přeloženo z [16], podrobnosti k nalezení v [17].

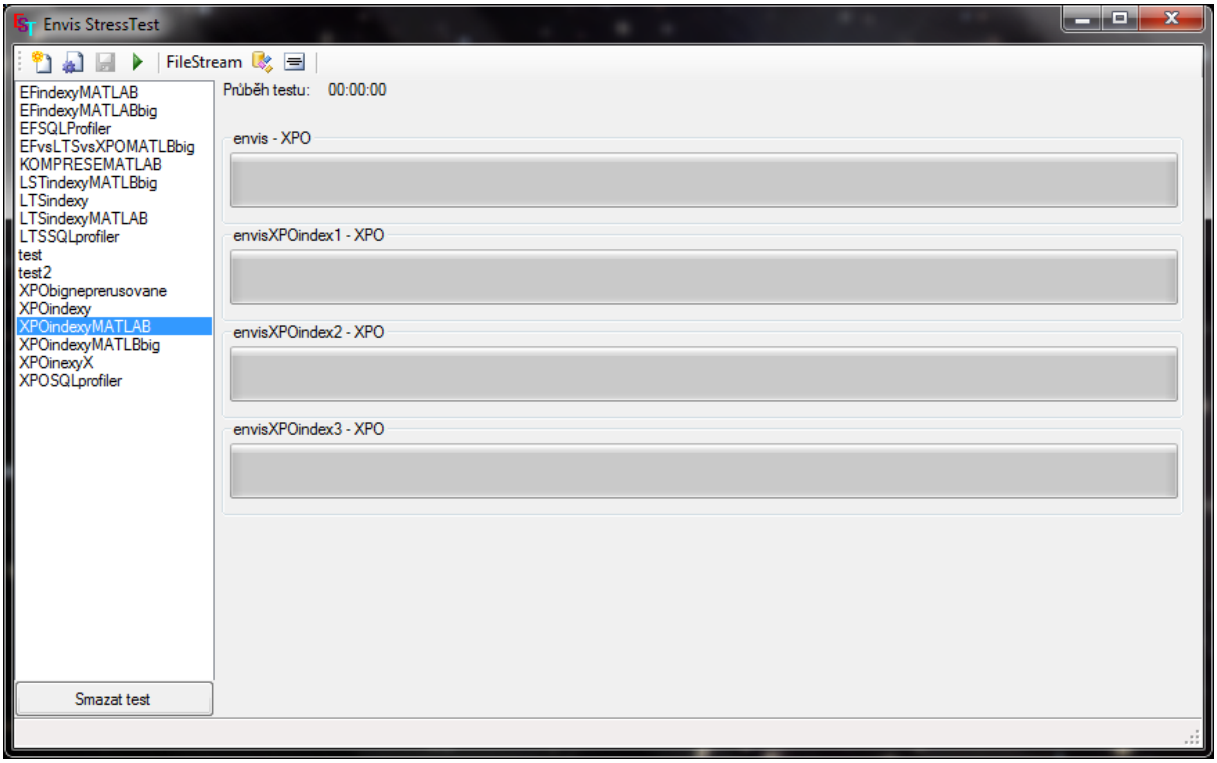
4 Testovací aplikace Envis Stress Test

EnivStressTest je program vytvořený za účelem provádění rychlostních testů na ENVIS databázi. Podporuje všechny zde popsané technologie (Entity Framework, Linq to SQL a eXpress Persitant Objects) pro přístup k databázi. Program jako vedlejší činnosti umí dělat testy pro FILESTREAMy a obsahuje i velmi jednoduchý nástroj pro přesun dat z aktuální verze databáze do databáze uzpůsobené právě FILESTEAMům. Celá aplikace je napsaná v .NET Frameworku 3.5 ve Visual Studiu 2010 technologií WinForms.

Všechny výsledky testů se defaultně ukládají do souborů používaných softwarem MATLAB, pro pozdější zpracování.

4.1 Popis grafického rozhraní programu

V horní části hlavního okna se nalézá menu, které obsahuje většinu příkazů, které jsou pro práci s EnvisStressTest programem potřeba. Je zde tlačítko pro nový test, nastavení, uložení a spuštění testu. Následně pak tlačítka pro práci s FILESTREAMem.



Obrázek 4-1 Hlavní okno

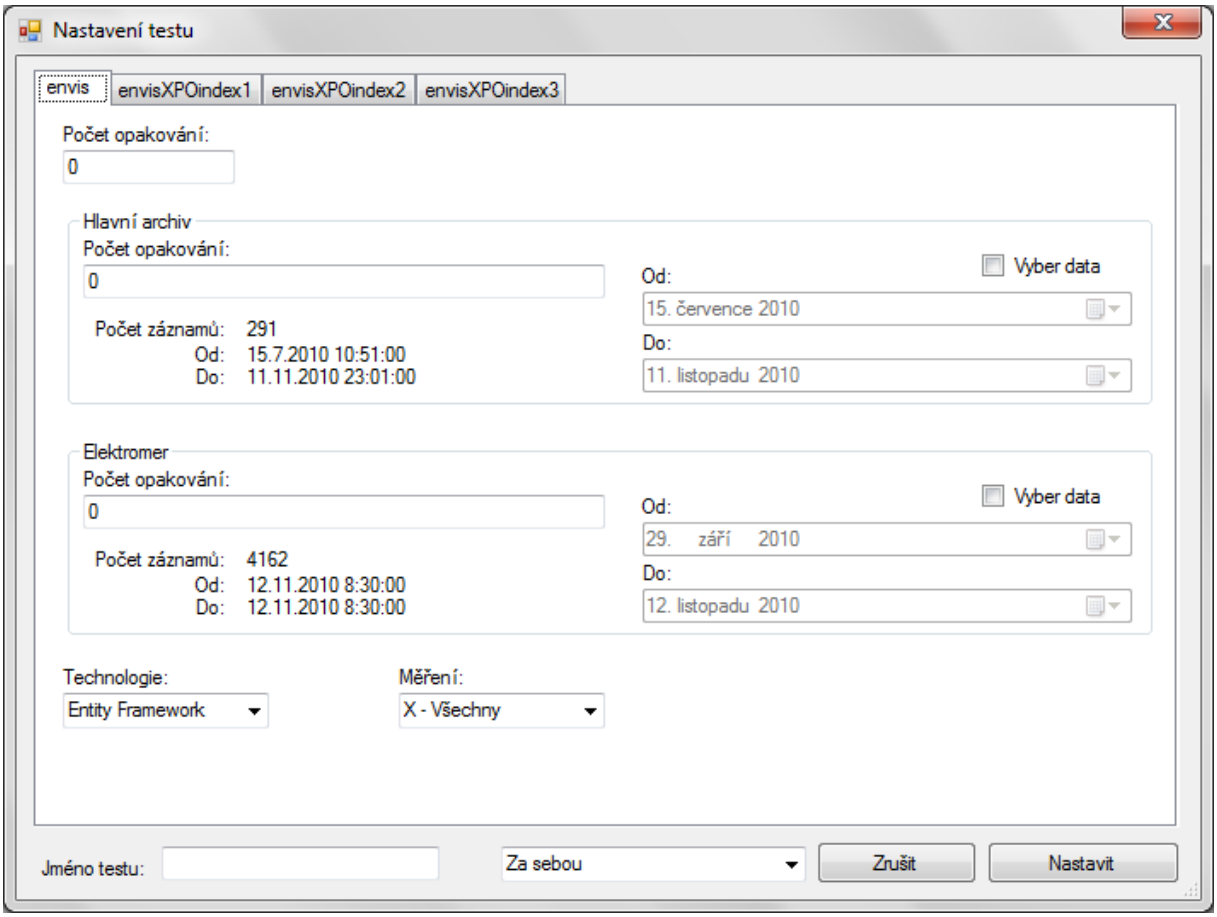
V levé části okna je jednoduchý seznam, kde jsou vypsány všechny uložené testy a tlačítko pro jejich smazání.

Uprostřed se defaultně nenalézá nic, avšak po nahrání testu se zde objeví vypsané jednotlivé databáze, na kterých má test probíhat a ukazatelé jejich průběhu.

V dolní části se ještě nachází status bar, do kterého se za běhu vypisují doplňující informace (počet databází, konkrétnější popisky, atd.).

Při vytváření nového testu se zobrazí velmi jednoduché okno, ve kterém se uživatel připojí k SQL Serveru a vybere databáze na kterých chce test provádět.

Aplikace uživatele následně přesměruje na formulář nastavení testu, kde se u každé databáze dají nastavit parametry specifikující metodu testování.



Obrázek 4-2 Nastavení testu

Počet opakování udává kolikrát se má na jedno spuštění provádět daný test (např. sto změřených výsledků). Počty opakování u hlavního archivu a elektroměru udávají kolikrát se má během jednoho opakování celého testu opakovat test na dané oblasti. Výběr dat specifikuje, na kterých datech v závislosti na čase se má test provádět. V listboxu technologie se vybírá, kterou technologií se bude daná databáze procházet. Na záložce měření se vybírá, na kterých konkrétně měřeních bude celá operace probíhat. Následuje již jen jméno a rozhodnutí zda testy na databázích provádět zároveň nebo za sebou.

Aplikace obsahuje ještě několik dalších formulářů, ale jejich popis zde není tolik důležitý. Jedná se například formuláře pro ukončení testu, pro měření FILESTREAMu, atd.. Kompletní program je k nalezení v CD příloze.

Testy se ukládají do souborového systému do složky *tests* v kořenovém adresáři aplikace, výstupy do MATLABu se ukládají do složky *matlab* v tom samém umístění.

4.2 Popis logiky aplikace

4.2.1 Tvorba modelů

Pro každou zvolenou technologii musí být vytvořený ORM model. Pro Entity Framework a Linq to SQL byly modely vytvořeny pomocí vestavěných grafických editorů v 1:1 relaci (jeden objekt odpovídá jedné konkrétní tabulce). Pro XPO byl zvolený již existující model společnosti KMB Systems, kvůli možnosti porovnání jiných technologií se současnou implementací.

Každý model byl tvořen v samostatném projektu ve Visual Studiu jako *.dll* modul. Především z důvodů velmi dlouhé kompilace (řádově desítky minut u Entity Frameworku) samotného modelu, nehledě na to že tento přístup umožňuje znovu použitelnost modelů v jiných příbuzných aplikacích.

V projektu modelu pro Entity Framework se ještě generují před generované pohledy (viz kapitola 2.1.3), bez kterých je na takto rozsáhlé databázi EF prakticky nepoužitelný.

4.2.2 Připojení k databázi

ORM frameworky jako takové se k databázi explicitně nepřipojují. Před prvním použitím se definuje databázový server, konkrétní databáze a přihlašovací informace. K databázi se aplikace připojuje pouze v okamžiku odeslání a zpracování dotazu. Což znamená, že když je odeslán dotaz a na model, tak k připojení k databázi dojde, až v momentě kdy se data skutečně zpracovávají. Proto se v rámci testování musejí všechna data ukládat do paměti i po zdánlivém zpracování dotazu.

Modely se často defaultně připojují pouze k databázi, ze které byly generovány, což je u většiny aplikací požadované chování. Ale v rámci testovací aplikace kde se pomocí modelu připojujeme k rozdílným databázím, které si však zachovávají strukturu, je minimálně nechtěné. Tyto připojující informace se uchovávají v *App.Config* souboru, je proto nutné nějakým způsobem měnit tato data za běhu aplikace. Pro tyto účely je v .NET Frameworku třeba třída *EntityConnectionStringBuilder*, která přesně tuto funkcionalitu poskytuje. Pro ostatní technologie existují obdobné třídy.

4.2.3 Samotné testování

Po nastavení a spuštění konkrétního testu, se pro každou testovanou databázi vytvoří samostatné vlákno. To umožňuje jednoduché přepínání mezi módy *Za sebou* a *Současně* a řeší problém případných konfliktů mezi testovanými technologiemi.

Po spuštění testu se buď spustí všechna vlákna najednou nebo se spustí první a po jeho ukončení se spustí další. Na každém vlákně proběhne načtení tříd pro konkrétní zvolenou

technologii. Následně se do seznamu uloží měření. Buď zadaná, nebo všechna. Pokud je zvolena možnost všechna, test projde všechny objekty, přístroje a následně až měření (simuluje se reálné chování uživatele v reálné aplikaci). Pro každé měření se naleznou jeho záznamy v hlavním archivu omezené datovými intervaly. Pomocí těchto záznamů se ukládají do paměti průměry, minima a maxima všech veličin. Konkrétní přesná data testy neberou v úvahu, protože indexy, komprese, atd. již nezlepší výkon na *varbinary(max)* komprimovaných datech. Pro testování rychlosti zpracování těchto dat jsou speciální testy pro FILESTREAM, které fungují velmi podobně, jen jsou strukturou upraveny i pro mírně změněné databáze.

4.2.4 FILESTREAM Data Relocator

Aplikace též obsahuje část nazvanou FILESTREAM Data Relocator. Slouží k přesunutí dat z aktuální verze databáze, do databáze upravené pro používání FILESTREAMů. Jedná se především o přesun dat na tabulce hlavního archivu. Přesun dat je realizován pomocí *SQLClienta* a *varbinary* složka se kopíruje pomocí parametrů v T-SQL dotazu.

4.2.5 Ovládání z příkazové řádky

Pro účely některých testů je potřeba restartovat aplikaci po každém jednotlivém průběhu testu. S tímto aplikace počítá a umožňuje spuštění konkrétního testu (podle jeho pořadového čísla v levém sloupci) z příkazové řádky pomocí parametrů.

Pokud tedy chceme spustit test s pořadovým číslem 4, stačí v příkazové řádce Power Shellu, po přesunutí do příslušné složky, zadat následující.

Zdrojový kód 4-1 Příkazová řádka

```
Start-Process EnvisStressTest.exe 4
```

Export do Matlabu je v případě použití příkazové řádky upraven tak, aby zaznamenával celkový čas průběhu testu a místo přepisů starých výsledků pouze toto číslo přidal do pole. Což při správném nastavení testů umožňuje celkem efektivně testovat pomocí smyček v *Power Shellu*.

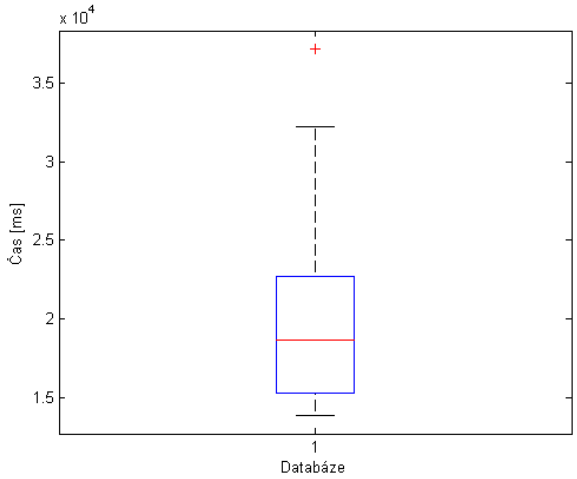
5 Metodika měření

U většiny prováděných testů je použit následující postup. Pokud je cokoliv provedeno jinak, je to řádně označeno v příslušné podkapitole. Konkrétní nastavení každého testu je k nalezení v přílohách a soubor s nastavením daného testu v CD příloze.

5.1 Testy

Testy jsou ve většině případů nastaveny na 100 opakování (100 konkrétních naměřených hodnot), 5 opakování hlavního archivu (celý hlavní archiv se během testu projde 500*) a 0 opakování u elektroměru. Používá se datové omezení, ale ve většině případů se používá plný rozsah intervalu (použito kvůli přesným a věrohodným dotazům). Měření jsou vybrána všechna, většina databází používá jen jedno měření, a pokud jsou vybrána všechna měření, lépe se simuluje chování uživatele. Aby si testy mezi sebou nebrali výpočetní výkon, jsou většinou nataveny tak, aby jednotlivé databáze šli za sebou, ne současně.

5.2 Grafy



Obrázek 5-1 Ukázkový graf

U většiny měření je k zobrazení výsledků použitý takzvaný *krabičkový diagram*. Ukázka takového grafu je na obrázku 5. Dolní a horní strana základního obdélníka (*krabičky*) odpovídají dolnímu a hornímu kvartilu. *Kvartil jsou tři hodnoty, jež rozdělují datovou sadu na čtyři přibližně stejně dlouhé díly.* Červená čára uvnitř této krabičky odpovídá mediánu dané datové sady. Boční strana obdélníku (výška) je označována jako *mezikvartilové rozpětí*. Dolní a horní svislá čára (*fous*) mimo krabičku odpovídá hodnotám, které leží pod nebo nad krabicí v maximální vzdálenosti rovné 1,5-násobku mezikvartilového rozpětí. Pod a nad fousy se vyskytují takzvané *odlehlé hodnoty*, na obrázku jsou vyznačeny červeným křížkem.

5.3 Seznam dat

V každé databázi, se může vyskytovat různé množství dat. Během popisu testů se jednotlivá používaná data se budou označovat podle tabulky v příloze II.

5.4 Použitý software a hardware

Většina testů je prováděna na počítači s operačním systémem *Windows 7 SP1* od společnosti Microsoft. U většiny databází je použit Microsoft SQL Server 2008 R2.

Pro většinu testů je použita tato sestava.

Procesor
2,4 GHz Intel Core i3 370M, 2 jádra, 4 thready, 3MB L3 cache
Operační paměť
4 GB DDR3
Pevný disk
500 GB SATA, 7200 RPM

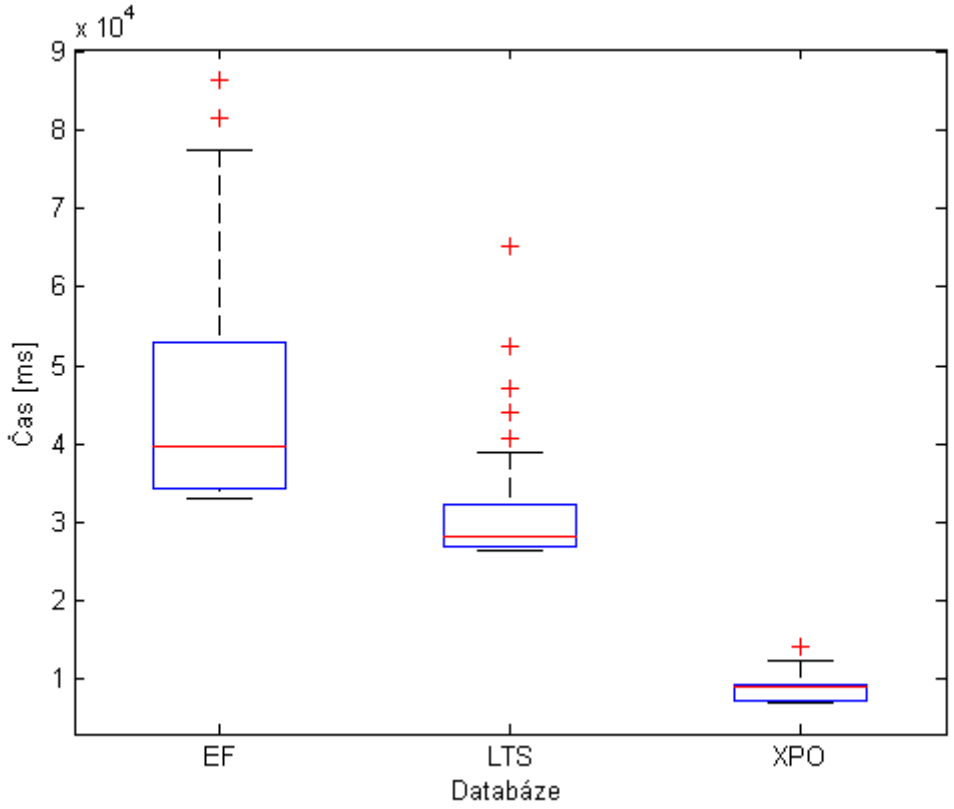
6 Jednotlivé testy

6.1 Entity Framework vs Linq to SQL vs eXpress Persistant Objects

6.1.1 Hlavní archiv a elektroměr

První prováděný test má za úkol rozhodnout, která z technologií pro přístup k datům je nejefektivnější. Test je v aplikaci EnvisStressTest nastaven podle popisu v kapitole 5.2.. Přesný popis je k nalezení v příloze C.a.

Všechny tři testy jsou testovány na vzorové neupravené databázi na velkých datech.



Graf 1 EF vs LTS vs XPO

Entity Framework je číslo 1, Linq to SQL číslo 2 a eXpressPersistant Objects číslo 3.

Obdobný test byl provedený i pro hodnoty na elektroměru. Graf příloha C.1.

Z těchto testů je vidět že XPO je výrazně nejrychlejší co se týká hlavního archivu, u elektroměru je však výrazně nejpomalejší. Entity Framework je na tom prakticky opačně.

6.1.2 SQL server 2005

Stejný test jako v 6.1.1. ale proveden na SQL serveru 2005 místo SQL Server 2008 R2. Test slouží k porovnání výkonu těchto dvou databázových nástrojů. Graf k nalezení v příloze C.2.

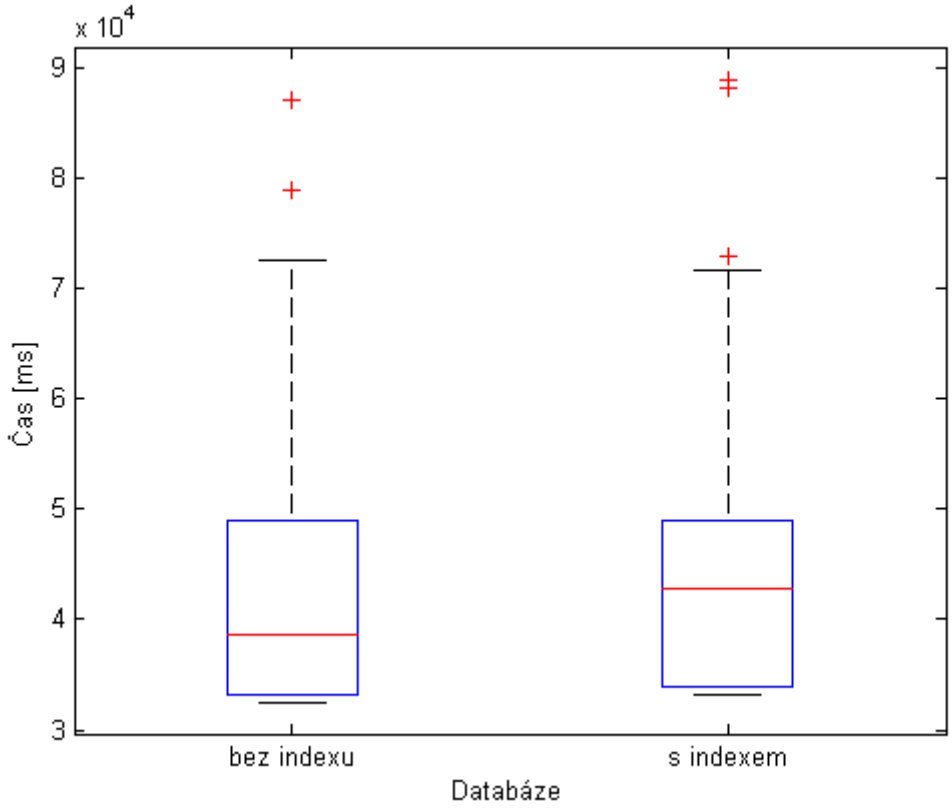
Z grafu je patrné, že SQL Server 2005 je u Entity Frameworku a LTS pomalejší, u XPO mírně rychlejší.

6.1.3 Porovnání na jiném hardware

Pro porovnání vlivu výkonu hardwaru, na kterém test běží, na výsledky testů, jsou testy s nastavením 6.1.1. provedeny na dalších dvou počítačích. Jeden je slabší a druhý silnější než popsaný použitý hardware v kapitole 5.5. Přesné hardwarové specifikace jsou k nalezení v přílohách C.c a C.d. Grafy jsou v příloze C.3 a C.4.

6.2 Entity Framework testy

6.2.1 Database Tuning Advisor Indexy



Graf 2 EF Advisor Indexy

Tento test porovnává výkon Entity Frameworku na vzorové databázi s velkými daty s výkonem po aplikování úprav doporučených Tuning Advisorem. Test je opět nastaven podle výchozího schématu. A přesné nastavení je k nalezení v příloze D.a.

Pod číslem jedna je testována neupravená vzorová databáze a pod číslem dva je databáze s aplikovanými změnami.

Graf s tímto samým testem, jen na databázích s malými daty, je k nalezení v příloze D.1.

Z grafů je patrné, že navržené změny mají spíše negativní dopad. S větším množství dat se negativní dopad zmenšuje.

6.2.2 Průběžné zlepšení

Tento test, zkoumá možnosti sebezlepšování Entity Frameworku. Tedy jak moc se zlepšuje výkon dotazů, pokud jich mnoho stejných běží za sebou bez přerušení spojení. Testovací mechanizmus je jednoduchý provádí se stejný test, jen s rozdílem že poprvé je spuštěn normálně a podruhé je po každém jednom opakování aplikace vypnuta a poté opětovně spuštěna. Toto se realizuje pomocí příkazové řádky *Windows Power Shell* a mechanizmu pro spouštění z konzole programu *EvisStressTest*.

Kompletní nastavení je v přílohách D.c a D.d, křivky jednotlivých průběhů jsou k nalezení v příloze D.2.

Z křivek je patrné, že nepřerušované spouštění testu na EF je efektivnější a méně kolísavé.

6.3 Linq to SQL testy

6.3.1 Database Tuning Advisor Indexy

Tento test je identický, jako test z kapitoly 6.2.1. jen je provedený pod Linq to SQL vrstvou. Nastavení v příloze E.a. Graf pro malá data E.1.

Navržené změny pro Linq to SQL vrstvu mají malý pozitivní efekt na výsledný výkon databáze, jak na malých, tak na velkých datech. Změna výkonu není nikterak výrazná.

6.3.2 Průběžné zlepšení

Tento test je identický, jako test z kapitoly 6.2.2. jen je provedený pod Linq to SQL vrstvou. Nastavení a graf v přílohách E.b a E.c, graf E.2.

Při dotazech pomocí Linq to SQL nedochází k vylepšování dotazů za běhu.

6.4 eXpress Persitant Object testy

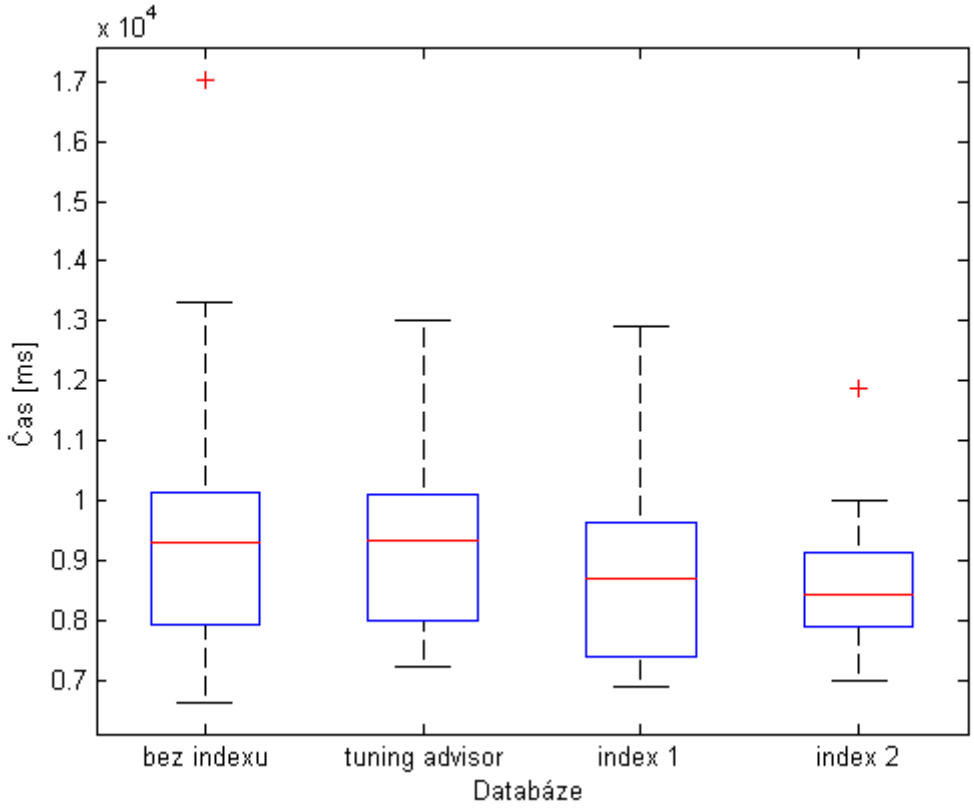
6.4.1 Indexy

Součástí tohoto testu je nejen část s doporučeními pomocí Tuning Advisoru (podle kapitoly 6.2.1), ale i další dvě databáze s různými jinými indexy.

Kompletní popis nastavení je k nalezení v příloze F.a.

Graf 3 pochází z testu na velkých datech. Sloupec jedna je původní vzorová databáze. Sloupec dva jsou změny doporučení Tuning Advisorem. Sloupec tři je nonclustered index na primárním klíči hlavního archivu (*keyMeans* a *keyTime*) a sloupci určující konečný čas měření (*endTime*). A sloupec čtyři je nonclustered index pouze na koncovém čase (*endTime*). Přesné T-SQL dotazy pro tyto indexy jsou k nalezení na CD příloze pod názvy *SQLIndex1.sql* a *SQLIndex2.sql*.

Graf ke stejnému testu, ale na malých datech je k nalezení v příloze číslo F.1.



Graf 3 XPO indexy

Z grafů je patrné, že indexy zde dělají opravdové změny ve výkonu na velkých datech je nejúčinnější nonclustered index na koncovém čase, zatímco na menších datech jsou nejúčinnější indexy navržené Tuning Advisorem. Výsledky dalších indexů na malých datech, také nejsou špatné.

6.4.2 Průběžné zlepšení

Tento test je identický, jako test z kapitoly 6.2.2., jen je provedený pod eXpress Persitatnt Objects vrstvou. Nastavení přílohy F.c a F.d graf F.2.

Při dotazech pomocí XPO nedochází k vylepšování dotazů za běhu.

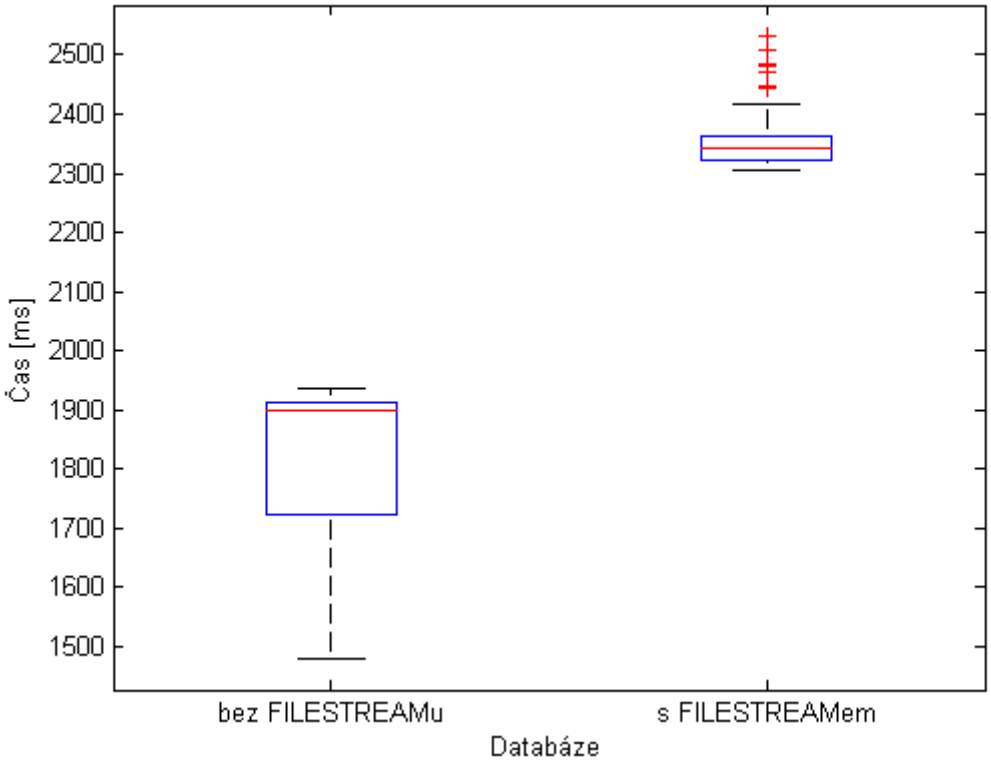
6.4.3 Read-only databáze

Účelem tohoto testu je porovnání výkonu mezi normálně nastavenou databází a databází nastavenou do režimu, ve kterém se z ní dá pouze číst. Přesné nastavení příloha F.e a graf F.3.

Hodnoty v grafu jsou si velmi podobné, u read-only databáze jsou však mírně horší.

6.5 FILESTREAM

V tomto testu se porovnává rychlost přístupu ke komprimovaným skutečně naměřeným hodnotám mezi tradičním přístupem pomocí T-SQL dotazu na databázi a metodou přístupu pomocí FILESTREAMu k datům přímo v souborovém systému.



Graf 4 FILESTREAM - velká data

Graf 4 znázorňuje stokrát opakovaný test čtení všech řádků v hlavním archivu databáze velkých dat a následné uložení komprimovaných dat (*Data*) do binárního pole. Jako technologie přístupu k datům je normální *SQLClient*, volba nějaké jiné konkrétní technologie nemá přímý výsledek na ovlivnění testu.

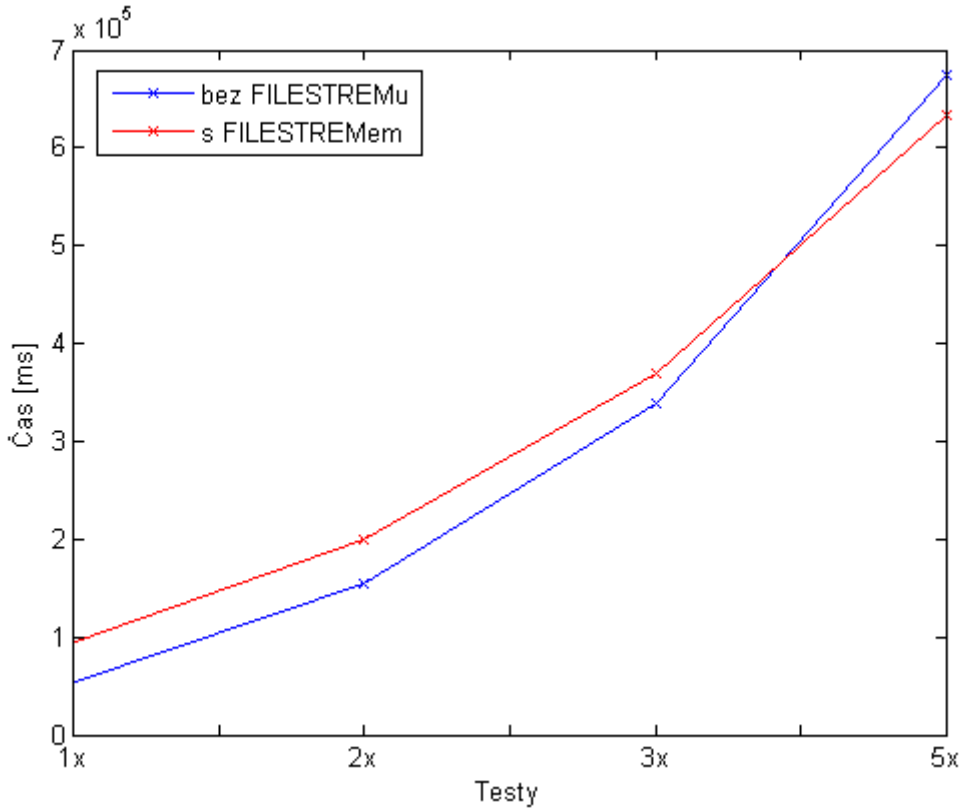
První sloupec znázorňuje současný přístup a druhý sloupec znázorňuje přístup pomocí FILESTREAMu.

Graf pro totožné nastavení testu, ale provedené na malých datech je k nalezení v příloze G.1.

Z výsledků se dá odvodit, že nasazení FILESTREAMu není příliš efektivní, dá se to přičítat nedostatečné velikosti uložených dat. Pro menší datové soubory je neefektivní přistupovat k jejich datům na souborový systém.

Další test této kapitoly, určuje za jakých podmínek (jak velké musí být datové soubory), je FILESTREM efektivní.

Tento test probíhal na osmi různých databázích. První použité databáze jsou ty samé databáze jako v první části této kapitoly. V druhém případě byl datový blok (967 kB) dvojnásobně zvětšený, v třetím třikrát a ve čtvrtém pětkrát. Na každém z těchto případů byl proveden desetkrát opakovaný test. Konečný čas průběhu testu, všech deseti opakování, byl zanesen do grafu. Modrá čára znázorňuje databáze s normálním přístupem k datům, červená čára znázorňuje časy po použití FILESTREMu.



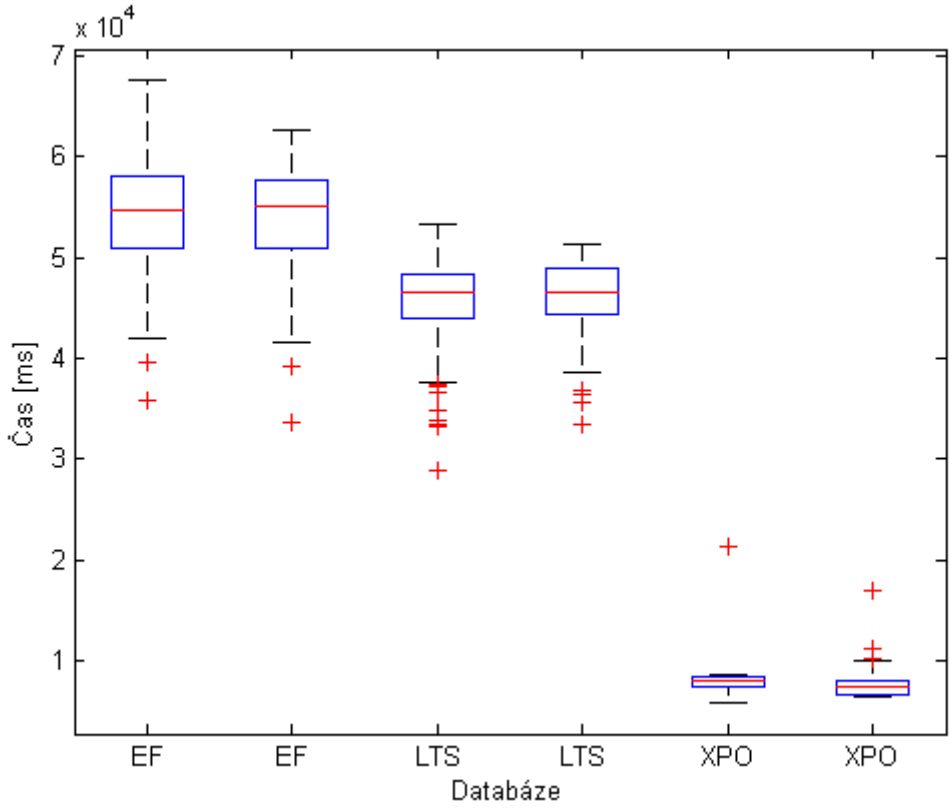
Graf 5 FILESTREAM různé velikosti datových bloků

Graf 5 ukazuje, že FILESTREAM je projeví svou účinnost až u datových bloků okolo 4500 kB (5x 967 kB).

6.6 Komprese

Test komprese spočívá v použití datové PAGE a ROW komprese na dvě databáze o stejných datech jako je vzorová a následné otestování zlepšení výkonu jednotlivých technologií pro přístup na těchto databázích. Nastavení je velmi podobné tomu z kapitoly 6.1., přesné nastavení je k nalezení v příloze H.

Graf 5 ukazuje výsledky testu na velkých datech, první dva sloupce jsou Entity Framework, druhé dva Linq to SQL a poslední eXpress Persistant Objects. První sloupec z dvojice je PAGE komprese, druhý sloupec je ROW komprese.



Graf 6 Komprese EF,LTS,XPO Velká data

Při porovnání Grafu 5 s Grafem 1 (EF vs LTS vs XPO) je zřejmé, že komprese má kladný vliv na výkon. U EF a LTS má PAGE komprese mírně lepší výsledky u XPO má mírně lepší výsledky ROW komprese.

7 Závěr

Z testů technologií přístupu k datům je jednoznačně znát, že eXpress Persitant Objects (XPO) je výrazně nejrychlejší technologie při testech na hlavním archivu, výsledky na elektroměru nejsou již tak podstatné. Nejlepší variantou tedy bude zůstat u současné technologie a datové vrstvy, implementace jiných možností není efektivní. (6.1)

Porovnání výkonu mezi SQL Serverem 2005 a 2008 R2 ukázalo, že současná implementace SQL Serveru 2005 má u XPO mírně příznivější výsledky, na rozdíl od ostatních technologií. Toto může být způsobeno rozdílnou zátěží hardwaru během dvou testů nebo lepší optimalizací XPO na SQL Server 2005. Dá se tedy usoudit, že vylepšení na SQL Server 2008 R2 není nutné. (6.1.2)

Ani implementace indexů navržených nástrojem Database Tuning Advisor nezefektivnila použití Entity Frameworku nebo Linq to SQL natolik, aby mohly nahradit XPO. Výsledky testů u indexů použitých pro XPO ukazují, že nonclustered index na koncovém čase u velkých dat má největší efekt na velkých datech. Na malých datech ovšem, již není tak efektivní. Zde jsou nejefektivnější změny navržené Tuning Advisorem (medián je o polovinu menší). Nonclustered index na primárních klíčích a koncovém čase, má výsledky u malých i velkých dat vždy jen nepatrně horší, jak nejlepší varianta. Implementace tohoto indexu je kompromisní řešení pro malá i velká data. (6.4.1)

FILESTREAM může být velmi užitečný nástroj, ale v případě této databáze není jeho použití příliš vhodné. Data, jež se do něho ukládají, jsou příliš malá, čtení souborů je tedy stále pomalejší než vyhledávání v souboru databáze. Pokud se bude v sloupci *data* v budoucnu uchovávat větší počet dat, použití FILESTREAMu by bylo vhodné uvážit. V současné době se jedná pro tyto účely o neefektivní nástroj. (6.5)

Použití ROW či PAGE datové komprese výrazně vylepšilo výkon všech testovaných technologií. Nároky na větší výpočetní výkon, jsou v dnešní době rychlých procesorů zanedbatelné. Mírně pomalejší zapisovací čas je oproti zrychlení čtení dat také přijatelná nevýhoda. Test ukazuje, že u XPO je mírně efektivnější pouze řádková komprese, přesto je zvážení hodné implementovat stránkovou kompresi, která obsahuje i řádkovou. Rozdíl již není tak výrazný a při datech s větší redundancí, může být stránková komprese efektivnější. (6.6)

Dodatky

Seznam literatury

[1] Mike Hotek: Microsoft SQL Server 2008. Computer Press, Brno, 2009. ISBN 978-80-251-2309-6

[2] Vidya Vrat Agarwal, James Huddleston: Databáze v C# 2008. Computer Press, Brno, 2009. ISBN 978-80-251-2309-6

[3] KMB Systems. *Manual SMV, SMP, SMPQ* [online]. KMB Systems, Liberec, 2009 [cit. 2011-04-08]. Dostupné z WWW: <http://www.kmb.cz/07/doc/SMV_SMP_SMPQ-Manual-v4-cze.pdf>.

[4] Christian Nagel: C# 2008 Programujeme profesionálně. Computer Press, Brno, 2009. ISBN 978-80-251-2401-7

[5] Robert E. Walters, Michael Coles, Robert Rae, Fabio Ferracchiati, Donald Farmer: Mistrovství v Microsoft SQL Server 2008. Computer Press, 2009. ISBN 978-80-251-2329-4

[6] Paoůo Pialorsi, Marco Russo: Microsoft LINQ. Computer Press, Brno, 2009. ISBN 978-80-251-2735-3

[7] Microsoft. *msdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. How to: Pre-Generate Views to Improve Query Performance. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/bb896240.aspx>.

[8] Microsoft. m*sdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. Row Compression Implementation. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/cc280576.aspx>.

[9] Microsoft. m*sdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. Page Compression Implementation. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/cc280464.aspx>.

[10] Microsoft. *msdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. How to: Enable FILESTREAM. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/cc645923.aspx>.

[11] Microsoft. m*sdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. How to: Create a FILESTREAM-Enabled Database. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/cc645585.aspx>.

[12] Microsoft. *msdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. Managing FILESTREAM Data by Using Transact-SQL. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/cc645962.aspx>.

[13] Microsoft. *msdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. Managing FILESTREAM Data by Using Win32. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/cc645940.aspx>.

[14] Microsoft. *msdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. Introducing SQL Trace. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms191006.aspx>.

[15] Microsoft. m*sdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. Using SQL Server Profiler. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms187929.aspx>.

[16] Microsoft. *msdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. Database Engine Tuning Advisor Overview. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms173494.aspx>.

[17] Microsoft. *msdn.microsoft.com* [online]. 2010 [cit. 2011-04-08]. Database Engine Tuning Advisor Features. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms174215.aspx>.

Seznam obrázků

Obrázky

Grafy

Seznam zdrojových kódů

Seznam příloh

Přílohy

A Rozdíly mezi Entity Frameworkem a Linq to SQL

Vlastnost	LINQ to SQL	Entity Framework
Model	doménový	konceptuální
Podporované databáze	SQL server	většina
Datové zdroje	pouze tabulky	tabulky, replikace, reporting Services atd.
Komplexnost	jednoduché na použití	složitější na použití
Vývojový čas	rychlý vývoj	pomalejší vývoj, ale více možNestí
Mapování	jedna třída jedna tabulka	třída pro více tabulek
Dědičnost	těžko aplikovatelná	jednoduše aplikovatelná
Typy souborů	pouze dbml	po kompilci generován edmx soubor s 3 sekcemi reprezentující schéma: csdl, msl a ssdl
Komplexní properties	není podporováno	podporováno ve VS2010, nebo manuální editací edmx souboru
Dotazy	1. LINQ to SQL (pro select) 2. Data Context (pro update, create, delete, store procedure, view)	1. LINQ to Entities (pro select) 2. Entity SQL 3. Object Services (pro update, create, delete, store procedure, view) 4. Entity Client
Umí synchronizovat model pokud se změní schéma databáze	nepodporováno	podporováno
Výkon	pomalý pro první dotaz	pomalý pro první dotaz, ale možnost předgenerovaných pohledů
Bude se vylepšovat do budoucna?	NE	ANE

Přeloženo z http://social.msdn.microsoft.com/Forums/en/adodotnetentityframework/thread/4ffb7c2a-bfa7-4e94-8526-9a924e87f123

B Tabulka dat

Název	Hlavní archiv	Elektroměr	CD příloha	Poznámka
Malá data	122	886	maladata.cea	-
Střední data	169	4076	strednidata.cea	-
Velká data	291	4962	-	Kombinace malých a středních dat

C EF vs LTS vs XPO

C.a Nastavení test hlavní archiv

3* vzorová DB envis
Opakování: 100
Hlavní archiv: 5
Elektroměr: 0
Technologie: EF, LTS, XPO
Měření: X-všechna
Datum povoleno, od začátku do konce.
Za sebou.

C.b Nastavení test elektroměr

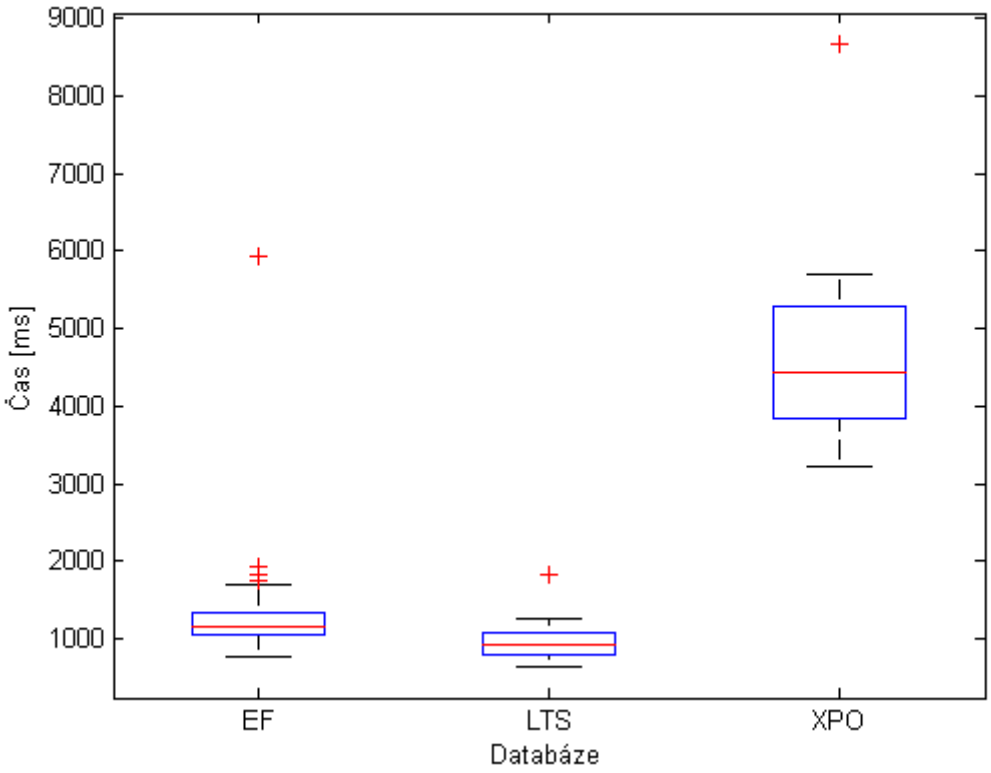
To samé nastavení jako III.a jen s touto změnou.
Hlavní archiv: 0
Elektroměr: 5

C.c Desktop – výkonný

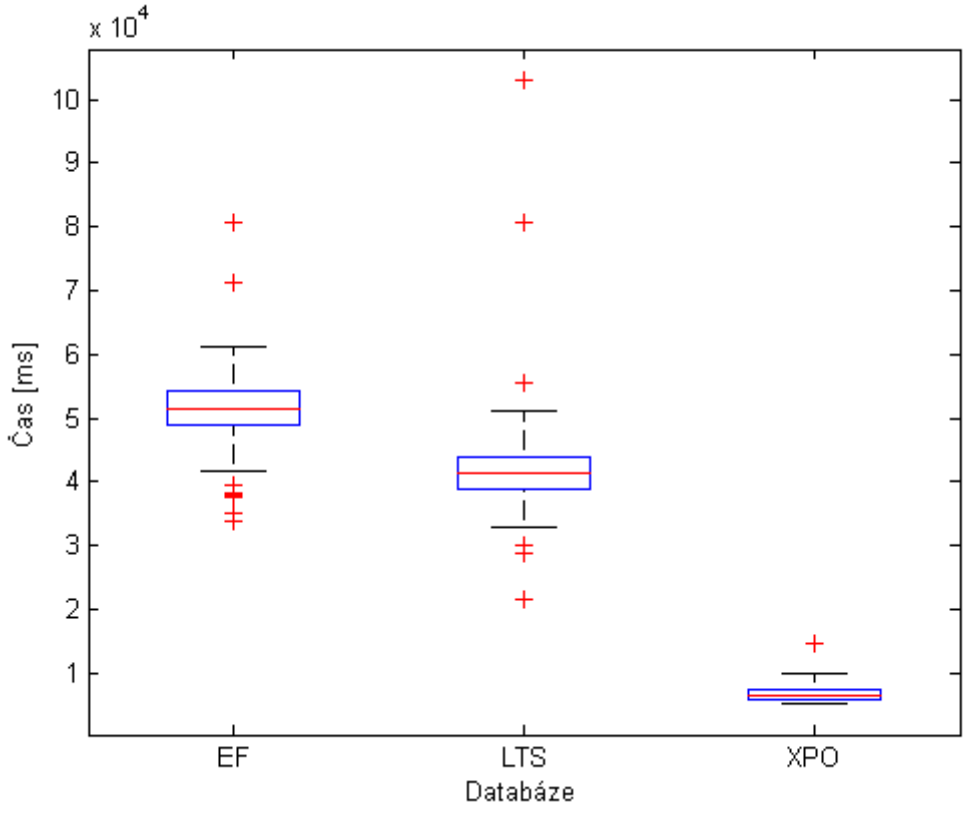
Procesor
2.8 GHz Intel Core i5 – 2300 BOX, 4 jádra, 6 MB L3 Cache
Operační paměť
8 GB DDR3
Pevný disk
500 GB SATA, 7200 RPM

C.d Desktop – pomalý

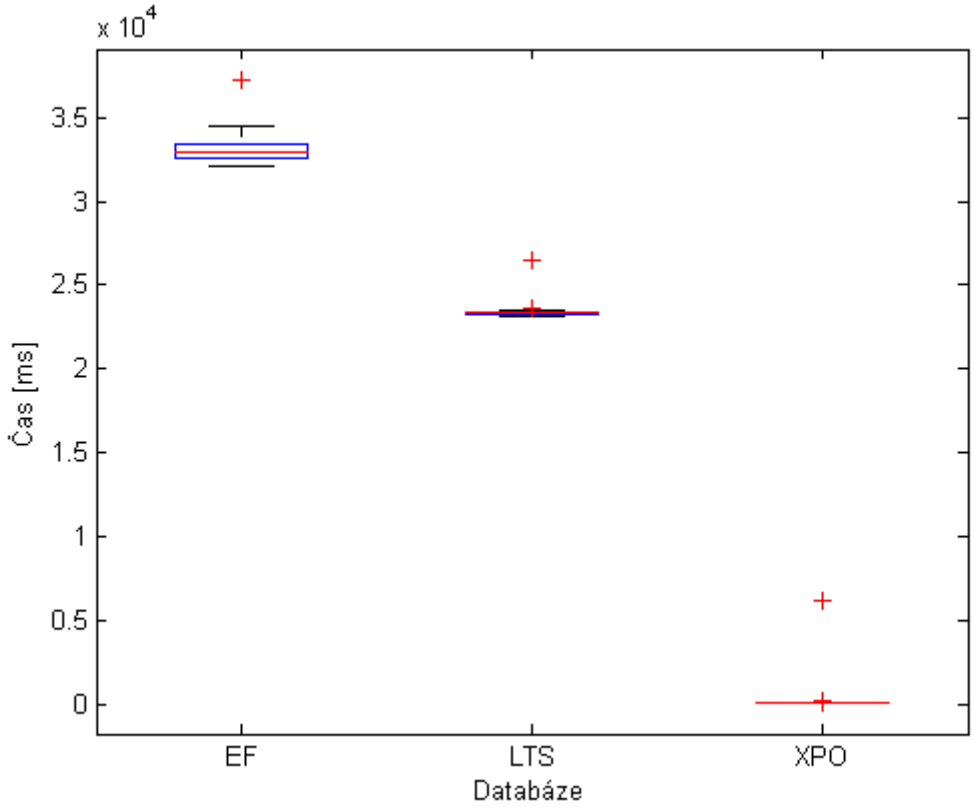
Procesor
2,7 GHz AMD Sempron 140, 1 jádro, 1024 KB L2 Cache
Operační paměť
1 GB DDR2
Pevný disk
250 SATA, 5400 RPM



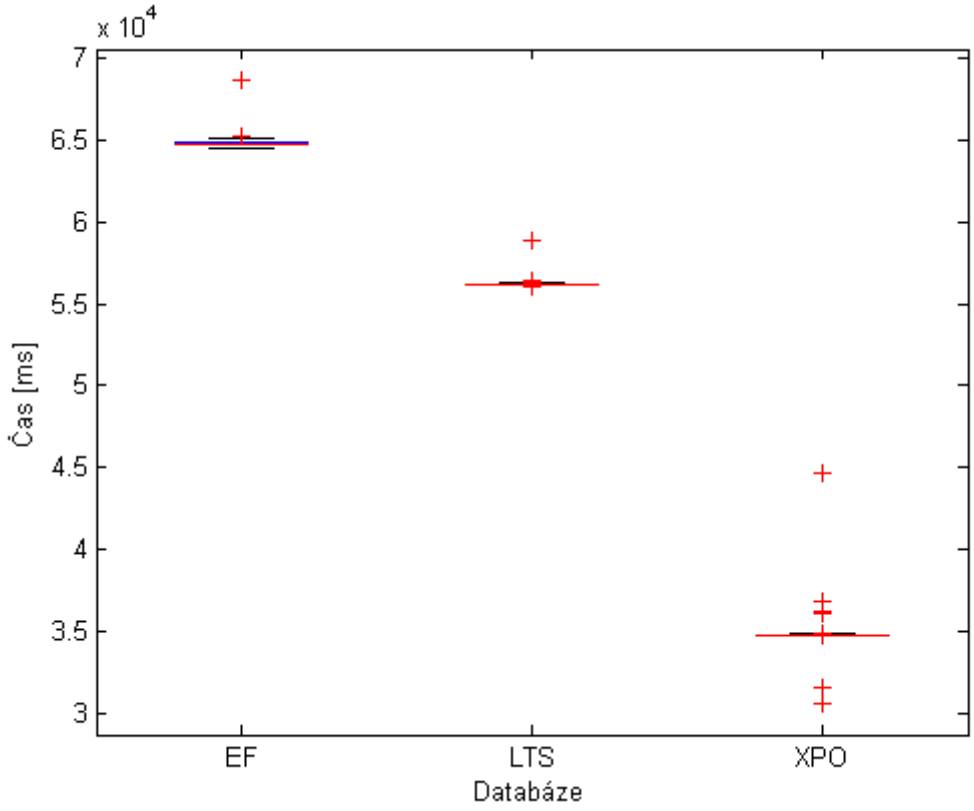
Graf C.1 EF vs LTS vs XPO Elektoměr



Graf C.2 EF vs LTS vs XPO SQLServer2005



Graf C.3 EFvsLTSvsXPO Desktop New



Graf C.4 EFvsLTSvsXPO Desktop Old

D Entity Framework testy

D.a Nastavení test velká data

1* vzorová DB envis, 1* DB envisEFindex1
Opakování: 100
Hlavní archiv: 5
Elektroměr: 0
Technologie: EF, EF
Měření: X-všechna
Datum povoleno, od začátku do konce.
Za sebou.

D.b Nastavení test malá data

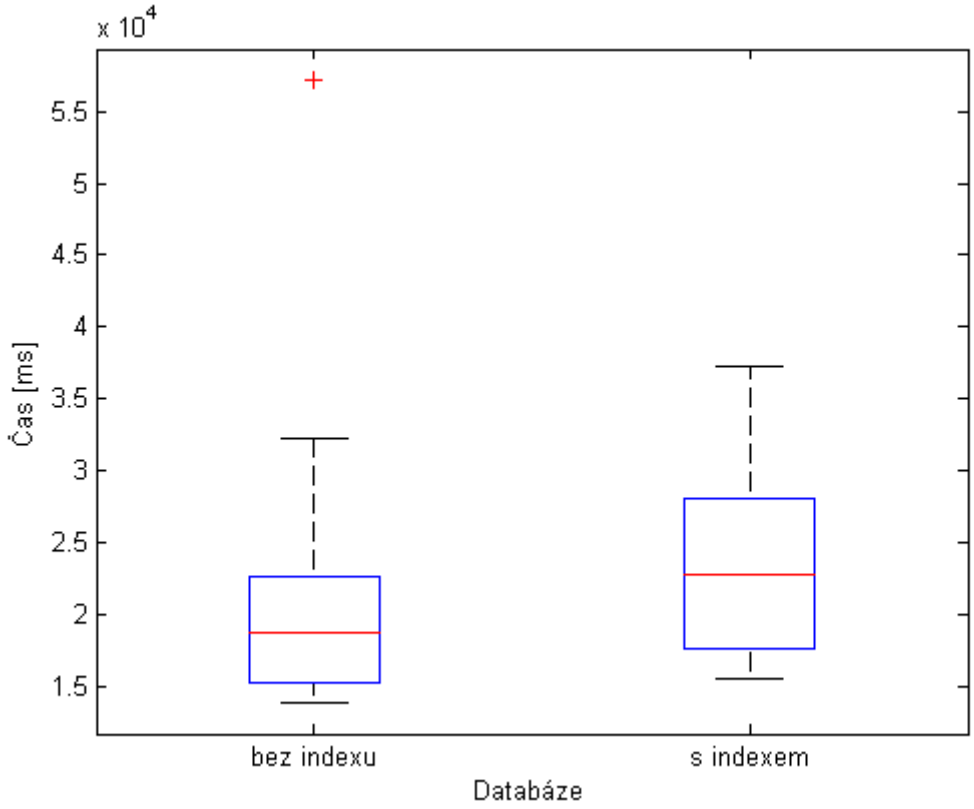
Stejné jako test IV.a, jen s touto změnou:
1* vzorová DB envisMalaData, 1* DB envisEFindex

D.c Nastavení bez přerušování

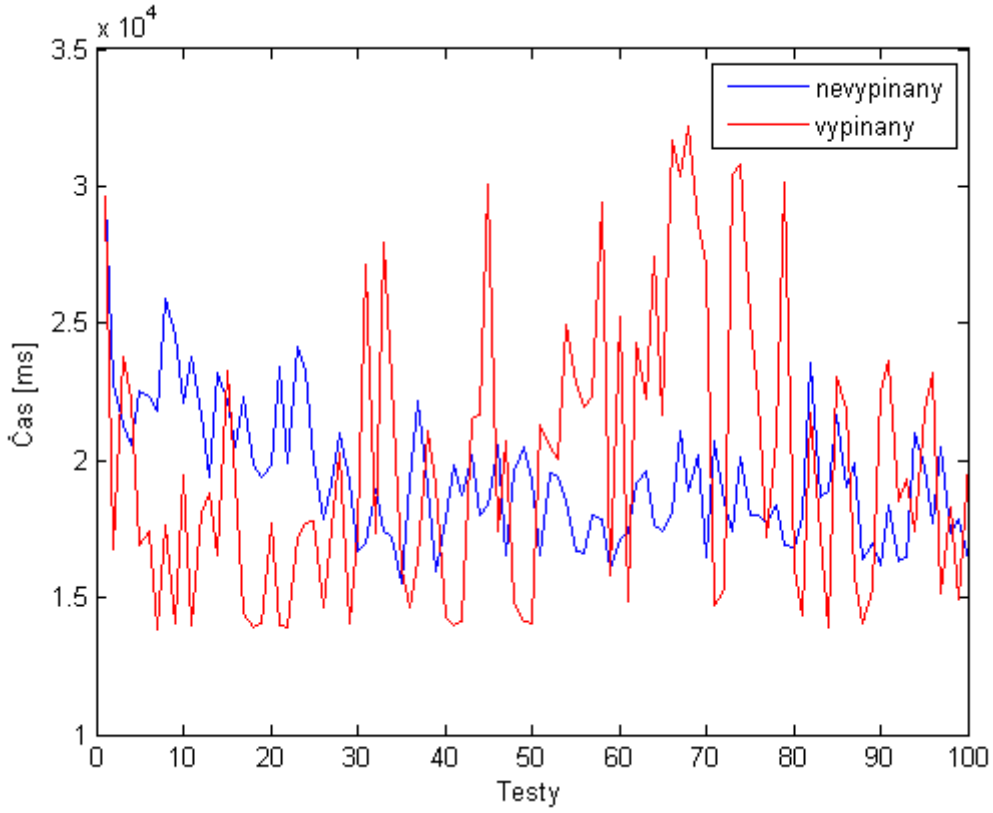
1* vzorová DB envis
Opakování: 100
Hlavní archiv: 5
Elektroměr: 0
Technologie: EF
Měření: X-všechna
Datum povoleno, od začátku do konce.
Za sebou.

D.d Nastavení s přerušováním

Opakování: 1. Současně.



Graf D.1 EF Indexy malá data



Graf D.2 EF přerušování vs nepřerušování

E Linq to SQL testy

E.a Nastavení test velká data

1* vzorová DB envis, 1* DB envisLTSindex1
Opakování: 100
Hlavní archiv: 5
Elektroměr: 0
Technologie: LTS, LTS
Měření: X-všechna
Datum povoleno, od začátku do konce.
Za sebou.

E.b Nastavení test malá data

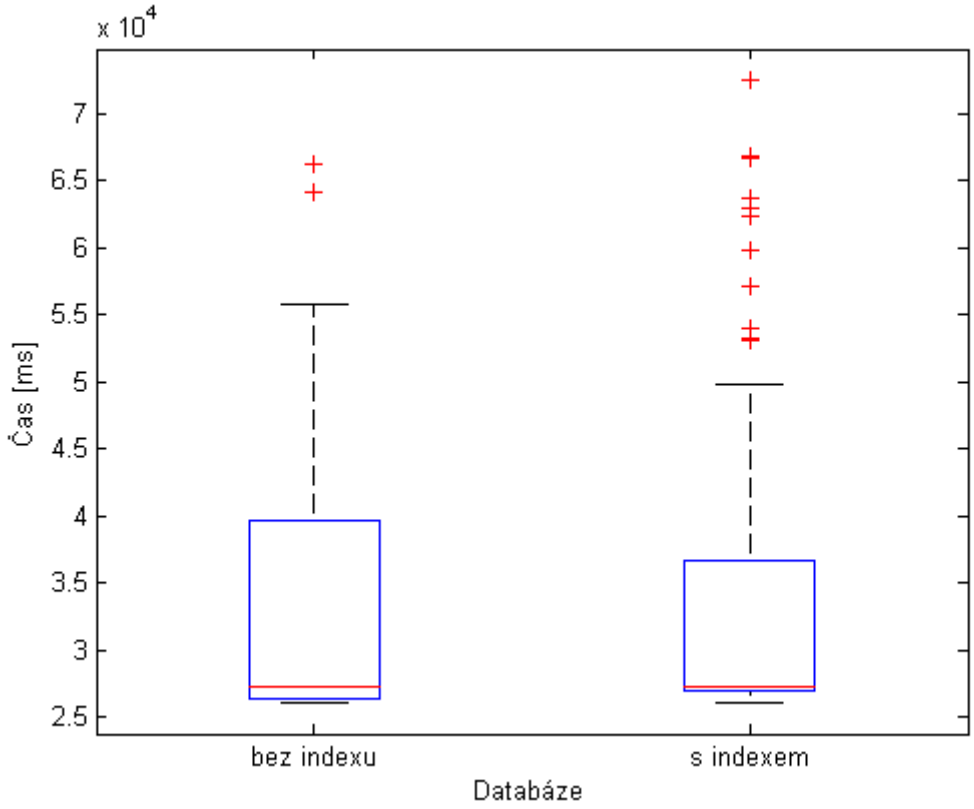
Stejné jako test E.a, jen s touto změnou:
1* vzorová DB envisMalaData, 1* DB envisEFindex

E.c Nastavení bez přerušování

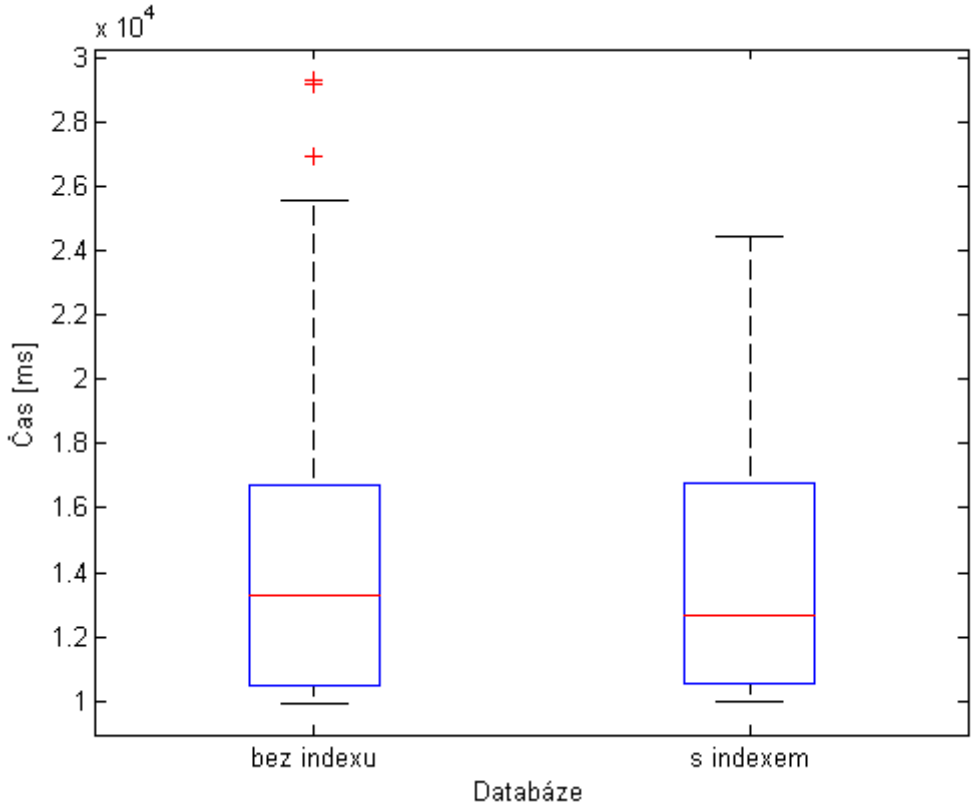
1* vzorová DB envis
Opakování: 100
Hlavní archiv: 5
Elektroměr: 0
Technologie: LTS
Měření: X-všechna
Datum povoleno, od začátku do konce.
Za sebou.

E.d Nastavení s přerušováním

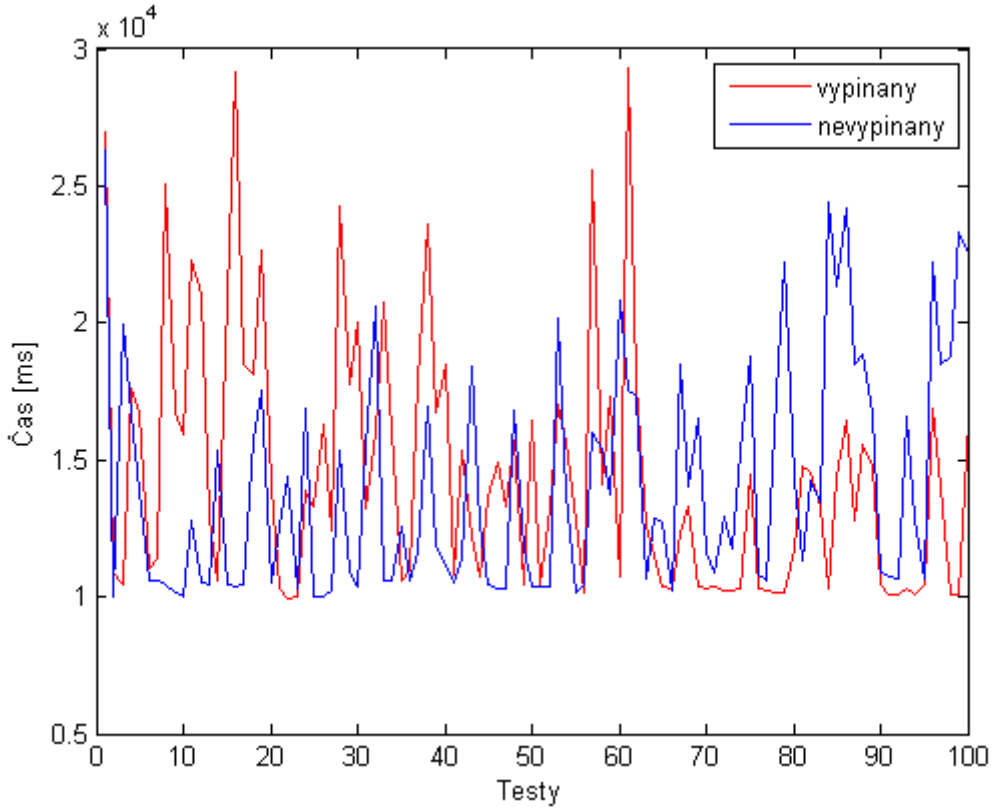
Opakování: 1. Současně.



Graf E.1 LTS indexy velká data



Graf E.2 LTS indexy malá data



Graf E.3 LTS přerušování vs nepřerušování

F eXpress Persistant Objects testy

F.a Nastavení test velká data

1* vzorová DB envis, 1* DB envisXPOindex1, 1* DB envixXPOindex2, 1* DB envisXPOindex3
Opakování: 100
Hlavní archiv: 5
Elektroměr: 0
Technologie: XPO, XPO, XPO, XPO
Měření: X-všechna
Datum povoleno, od začátku do konce.
Za sebou.

F.b Nastavení test malá data

Stejné jako test F.a, jen s touto změnou:
1* vzorová DB envisMalaData, 1* DB envisMDXPOindex1, 1* DB envisMDXPOindex2, 1* DB envisMDXPOindex3

F.c Nastavení bez přerušování

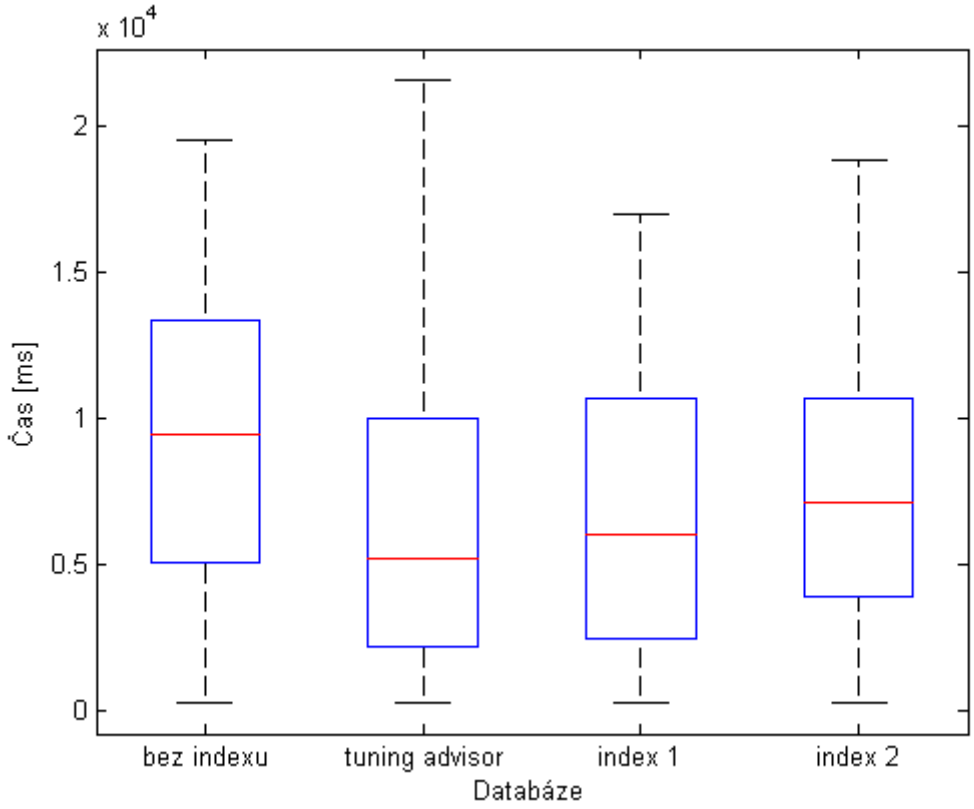
1* vzorová DB envis
Opakování: 100
Hlavní archiv: 5
Technologie: XPO
Měření: X-všechna
Datum povoleno, od začátku do konce.
Za sebou.

F.d Nastavení s přerušováním

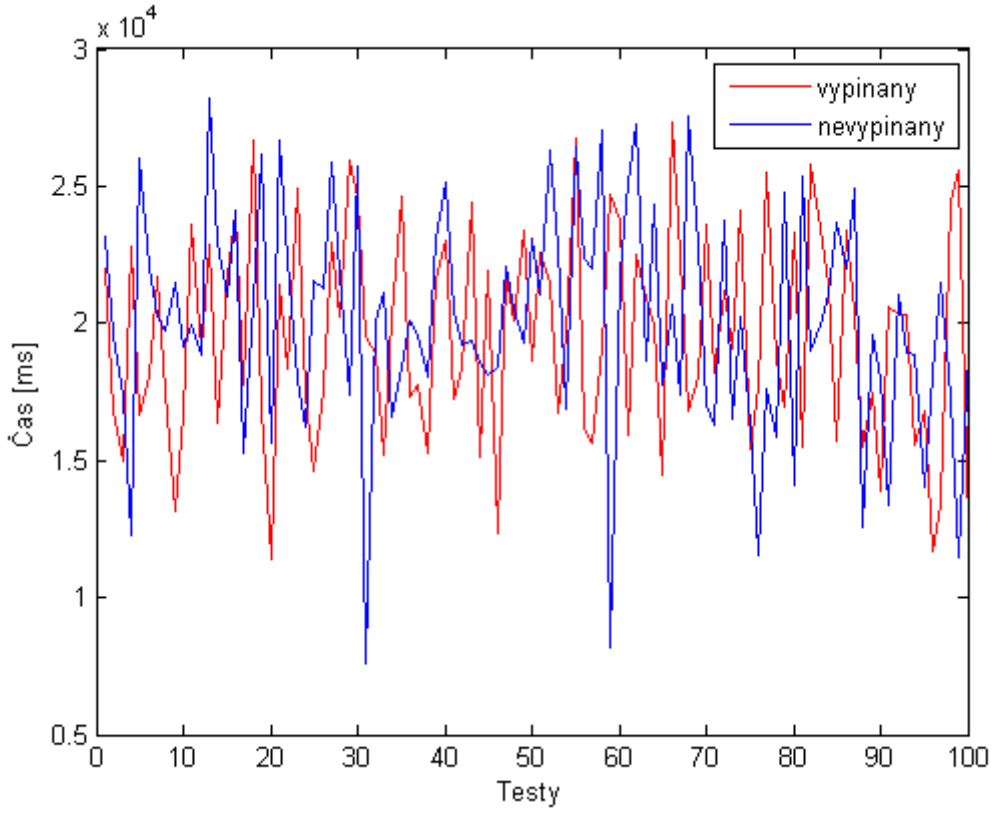
Opakování: 1. Současně.

F.e Nastavení s readonly

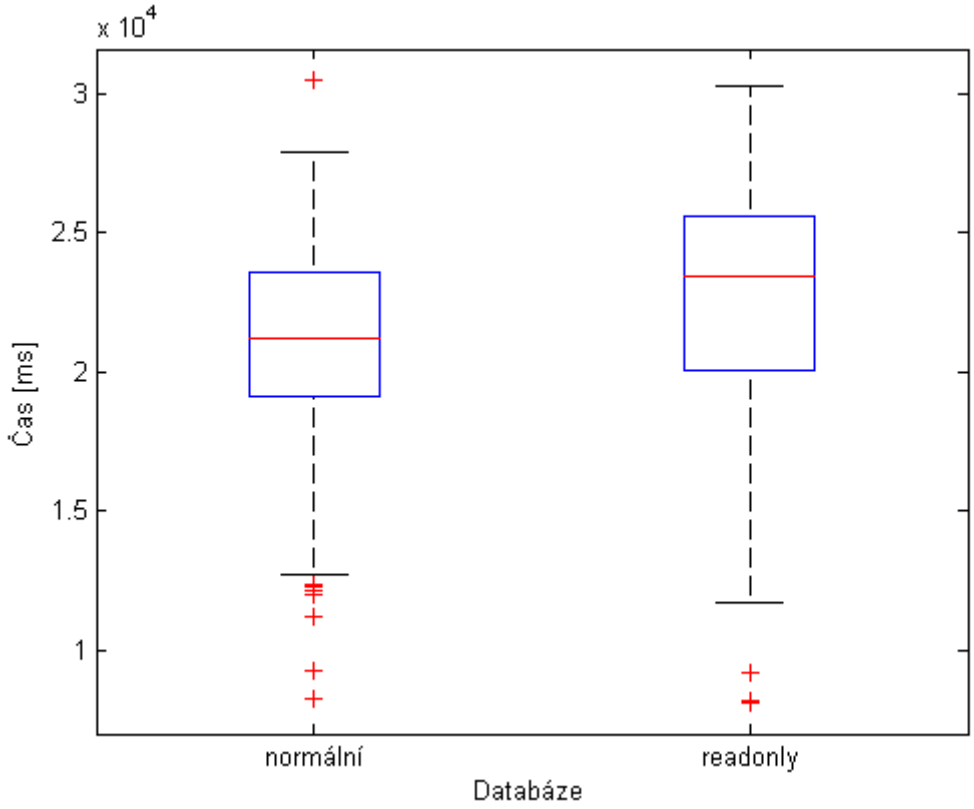
Stejné jako VI.a jen s těmito změnami
1*vzorová DB envis, 1* envisReadOnly
Technologie: XPO, XPO



Graf F.1 XPO indexy malá data

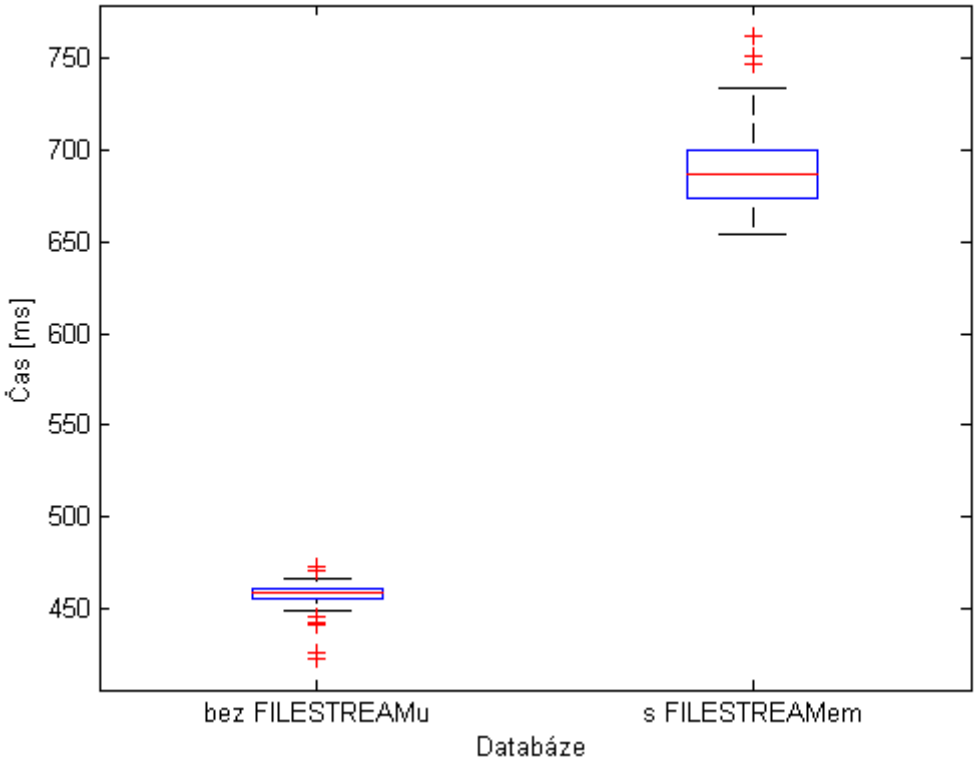


Graf F.2 XPO vypínaný vs nevypínaný



Graf F.3 XPO normální vs readonly

G FILESTREAM



Graf G.1 FILESTREAM - malá data

H Komprese

Nastavení test

envisPAGEkomprese, envisROWkomprese, envisPAGEkomprese, envisROWkomprese, envisPAGEkomprese, envisROWkomprese
Opakování: 100
Hlavní archiv: 5
Elektroměr: 0
Technologie: EF, EF, LTS, LTS, XPO, XPO
Měření: X-všechna
Datum povoleno, od začátku do konce.
Za sebou.