

UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017

Kateřina Lebedová

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

In-memory Database Cache – analýza a zhodnocení

Kateřina Lebedová

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2016/2017

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Kateřina Lebedová**
Osobní číslo: **I13280**
Studijní program: **B2646 Informační technologie**
Studijní obor: **Informační technologie**
Název tématu: **In-Memory Database Cache - analýza a zhodnocení**
Zadávající katedra: **Katedra informačních technologií**

Z á s a d y p r o v y p r a c o v á n í :

Cílem práce je popsat technologie související s memory cache. Výkonnostní srovnání zátěžových dotazů v různých databázových serverech bez využití memory cache a s jejím využitím. Součástí práce je i srovnání výkonu a popis změn ve vnitřním řešení vybraných databázových serverů. Cílem praktické části jsou srovnávací zátěžové testy včetně použitých skriptů s technickým popisem použitého zařízení a výsledná analýza při monitoringu práce s databází.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

DAVIDSON, Louis. Exam ref 70-762 developing sql databases. ISBN 9781509304943.

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

STANEK, William R. Microsoft SQL Server 2012: kapesní rádce administrátora. Brno: Computer Press, 2013. Microsoft (Computer Press). ISBN 978-80-251-3797-0.

WALTERS, R. E. Mistrovství v Microsoft SQL Server 2008: [kompletní průvodce databázového experta]. Brno: Computer Press, 2009. ISBN 978-80-251-2329-4.

Vedoucí bakalářské práce:

Ing. Monika Borkovcová

Katedra informačních technologií

Datum zadání bakalářské práce:

31. října 2016

Termín odevzdání bakalářské práce:

12. května 2017

Ing. Zdeněk Němec, Ph.D.
děkan



L.S.

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.

Byla jsem seznámena s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 14. 2. 2018

Kateřina Lebedová

PODĚKOVÁNÍ

Děkuji vedoucí mé bakalářské práce Ing. Monice Borkovcové, Ph.D. za odborné vedení, vřelý přístup, spoustu cenných rad a velkou dávku trpělivosti při zpracování této bakalářské práce.

ANOTACE

Cílem práce je popsat technologie související s memory cache. Výkonnostní srovnání zátěžových dotazů v různých databázových serverech bez využití memory cache a s jejím využitím. Součástí práce je i srovnání výkonu a popis změn ve vnitřním řešení vybraných databázových serverů. Cílem praktické části jsou srovnávací zátěžové testy včetně použitých skriptů s technickým popisem použitého zařízení a výsledná analýza při monitoringu práce s databází.

KLÍČOVÁ SLOVA

Oracle, Database, Microsoft, MSSQL, MySQL, PostgreSQL

TITLE

In-Memory Database Cache – the analysis and evaluation

ANNOTATION

The aim of this thesis is to describe the memory cache related technologies. Comparing of performance of load queries in various database servers without the use of memory cache and with its use. Part of the thesis is also a comparison of performance and description of changes in the internal solution of selected database servers. The aim of the practical part are the comparative stress tests including the used scripts with the technical description of the equipment used and the resulting analysis of monitoring of the work with the database.

KEYWORDS

Oracle, Database, Microsoft, MSSQL, MySQL, PostgreSQL

Obsah

Úvod.....	12
1 Databázové systémy	13
1.1 Systém řízení báze dat.....	13
1.1.1 Integrita dat	13
1.1.2 ACID.....	14
1.1.3 Architektura	15
1.1.4 Datový model.....	16
1.1.5 Databázové schéma.....	17
1.1.6 SQL.....	18
2 Přehled databázových systémů	20
2.1 Databázový systém ORACLE DATABASE	20
2.1.1 Obecná charakteristika.....	20
2.1.2 Architektura	20
2.1.3 Oracle Database 10g	22
2.1.4 Oracle Database 11g	22
2.1.5 Oracle Database 12c	23
2.1.6 Memory cache.....	24
2.1.7 Zpracování dotazů.....	24
2.2 Databázový systém MICROSOFT SQL SERVER.....	27
2.2.1 Obecná charakteristika.....	27
2.2.2 Architektura	27
2.2.3 Microsoft SQL Server 2012.....	29
2.2.4 Microsoft SQL Server 2014.....	29
2.2.5 Microsoft SQL Server 2016.....	30
2.2.6 Memory cache.....	30
2.2.7 Zpracování dotazů.....	31

2.3	Databázový systém MYSQL.....	32
2.3.1	Obecná charakteristika.....	32
2.3.2	Architektura	32
2.3.3	Verze 5.5.....	33
2.3.4	Verze 5.6.....	34
2.3.5	Verze 5.7.....	34
2.3.6	Memory cache.....	34
2.3.7	Zpracování dotazů.....	35
2.4	Databázový systém POSTGRESQL	36
2.4.1	Obecná charakteristika.....	36
2.4.2	Architektura	36
2.4.3	Verze 9.4.....	37
2.4.4	Verze 9.5.....	37
2.4.5	Verze 9.6.....	38
2.4.6	Memory cache.....	38
2.4.7	Zpracování dotazů.....	38
2.5	SAP HANA.....	40
2.5.1	Memory cache.....	40
3	Metody Testování	41
3.1	White-box.....	42
3.2	Black-box	42
4	Testovací prostředí.....	44
5	Příprava databázového systému.....	45
5.1	Databázové schéma.....	45
5.2	Naplnění daty	45
5.3	Změny v SQL.....	46
6	Příprava dotazů	47

7	Zátěžový test	49
7.1	Popis testovacích částí.....	49
8	Vyhodnocení.....	51
	Závěr	56
	Zdroje.....	57
	Přílohy.....	60

SEZNAM ILUSTRACÍ A TABULEK

Obrázek 1 - Třívrstvá architektura (vlastní zpracování)	16
Obrázek 2 - Relačně-entitní model (vlastní zpracování)	17
Obrázek 3 – Relační model (vlastní zpracování)	17
Obrázek 4 - Architektura Oracle Database (Oracle Corporation 2016)	21
Obrázek 5 - Exekuční plán Oracle Database (vlastní zpracování)	26
Obrázek 6 - Architektura MS SQL (Microsoft 2016)	28
Obrázek 7 - Exekuční plán MS SQL (vlastní zpracování)	31
Obrázek 8 - Exekuční plán MySQL (vlastní zpracování)	35
Obrázek 9 - Exekuční plán PostgreSQL (vlastní zpracování)	39
Obrázek 10 - Databázové schéma (vlastní zpracování)	45
Obrázek 11 - Aplikace (vlastní zpracování)	51
Tabulka 1 – Oracle Database v číslech (Oracle Corporation 2016)	20
Tabulka 2 - Microsoft SQL Server v číslech (Microsoft 2016)	27
Tabulka 3 - MySQL v číslech (MySQL 2016)	32
Tabulka 4 - PostgreSQL v číslech (PostgreSQL Global Development Group 2016)	36
Graf 1 – Insert/Update (vlastní zpracování)	52
Graf 2 – Spojení tabulek (vlastní zpracování)	52
Graf 3 – Vnořený dotaz (vlastní zpracování)	53
Graf 4 – Agregáční funkce (vlastní zpracování)	53
Graf 5 – Order by (vlastní zpracování)	54
Graf 6 – Import dat (vlastní zpracování)	54
Graf 7 – Propustnost databáze (vlastní zpracování)	55
Graf 8 – Doba testu (vlastní zpracování)	55

ÚVOD

Od příchodu první kartotéky se lidstvo zabývá sběrem dat a informací. Co se však mění, je postup k dosažení tohoto cíle. Z důvodu neustálého vývoje a nárůstu množství dat tento proces prochází různými inovacemi. Mezi hlavní otázky tohoto odvětví v dnešní době patří, jakým způsobem mohou být nejrychleji a nejsrozumitelněji jakákoliv data uložena. V rámci informačních technologií tkví základní řešení v ukládání dat do databázového systému a jak k nim co nejrychleji přistupovat.

Aktuální problematikou na poli databázových systémů je architektura s podporou memory cache, tzn. ukládání do mezipaměti. Toto ukládání funguje nejlépe s daty, která se zřídka mění. Tento proces funguje tak, že se během ukládání zhotoví kopie dat, která lze vrátit z mezipaměti mnohem rychleji než z původního zdroje.

Teoretická část této práce je věnována stručnému vymezení základních pojmů souvisejících s databázovými systémy. Součástí představení databázových systémů jsou nároky kladené na databázový systém, jeho architekturu či jazyk SQL, jehož hlavním úkolem je právě komunikace s již konkrétní databází. Veškeré získané informace jsou následně podrobněji popsány na konkrétních příkladech databázových systémů, kdy je blíže vysvětlena samotná memory cache a způsob, jakém ji dané databázové systémy implementují. Součástí základního přehledu nutného pro praktickou část je i porovnání dřívějších verzí databázových systémů. Jistým přechodem mezi teoretickou a praktickou částí tvoří kapitola věnovaná základním testovacím metodám pro lepší pochopení postupů v praktické části.

Výstupem praktické části je aplikace testující výkon vybraných databázových systémů. Nejprve probíhá úvodní seznámení s vytvořeným testovacím databázovým schématem, s jehož pomocí budou databázové systémy testovány. Následně je čtenář seznámen s popisem načtených dat a změnami v rámci syntaxe jazyka SQL u různých databázových serverů. Úkolem vytvořené aplikace je jednotná komparace více databázových systémů najednou a otestování rychlosti zpracování dotazů. K tomuto účelu poslouží jednoduchý test, který postupně spouští své jednotlivé bloky na základě procentuálního ohodnocení a sleduje jednotlivé zátěžové časy.

Mezi hlavní cíle této bakalářské práce bezpochyby patří monitorování činnosti databázového systému s využitím memory cache a bez jeho využití a následné zpracování výsledků se závěrečným vyhodnocením oproti ostatním.

1 DATABÁZOVÉ SYSTÉMY

Tato kapitola je věnována úvodu do problematiky databázových systémů. Jejím cílem je vymezení základních pojmů, vlastností a požadavků na databázové systémy.

1.1 Systém řízení báze dat

Systém řízení báze dat (dále SŘBD, angl. DBMS – Database Management System) je společně s databází jedním ze dvou základních složek databázového systému. Vytváří spojení mezi samotnou databází a uživatelskou aplikací, čímž výrazně ulehčuje jakoukoliv manipulaci s daty. Nejen, že musí být schopen účinně manipulovat s velkým objemem dat najednou, nýbrž musí poskytovat i jejich zpracování, tzn. jejich vkládání, mazání či editaci.

Mezi základní charakteristiky SŘBD patří [1]:

- entity reálného světa – moderní SŘBD využívá reálných entit pro návrh architektury,
- relačně založené tabulky – SŘBD umožňuje entitám a vztahům mezi nimi vytvořit tabulky, kdy uživatel může pochopit architekturu databáze jen při pohledu na její název,
- integrita dat – vložená data v databázi splňují integritní pravidla,
- nižší redundance – na základě matematických procesů snižuje propustnost dat,
- jazyk – SŘBD je vybaven dotazovacím jazykem, na jehož základě probíhá veškerá manipulaci s daty,
- vlastnosti ACID – transakce probíhají v souladu se zásadami atomičnosti, konzistence, izolace a trvanlivosti,
- přístup pro více uživatelů – současně může ke stejným uloženým datům přistupovat více uživatelů najednou,
- množství pohledů – umožňuje uživateli zobrazení potřebných dat dle jejich požadavků, role atp.,
- bezpečnost – zajišťuje kdo, kdy, kde a s jakými daty může manipulovat.

1.1.1 Integrita dat

Jednou z nejdůležitějších funkcí SŘBD je zajištění datové integrity¹. Ta poskytuje uživateli ukládání dat do databáze za dodržení integritních pravidel. Existuje jich několik druhů rozdělených dle místa v databázi, na kterou bude dané pravidlo aplikováno.

Prvním z těchto pravidel je *entitní pravidlo*. To říká, že pro každý řádek záznamů v tabulce existuje unikátní primární klíč² (angl. primary key), který nenabývá prázdné hodnoty NULL.

¹ Datová integrita = zajištění přesnosti a soudružnosti dat

² Primární klíč = určuje současnou hodnotu z databázové tabulky, každá tabulka by měla mít právě jeden jedinečný primární klíč s ne-NULL-ovou hodnotou

Na základě tohoto klíče nemohou být do tabulky vložena stejná data, např. zaměstnanci se stejným rodným číslem.

Další pravidlo se nazývá *doménové*, které je aplikováno na konkrétní sloupce v tabulce. To zajistí jejich omezení na určité datové typy, popř. rozsah hodnot, ve kterých mají být data uchovávána. Nemůže být tedy do sloupce s rodným číslem zaměstnance zadáno jeho jméno.

Poslední pravidlo – *referenční* – se zabývá relacemi³. Jejich typ se určuje na základě cizích klíčů⁴ (angl. foreign key). V tomto případě je vždy jedna z tabulek označena jako tzv. rodič (angl. master), které se další tabulky podřizují (angl. detail). Cizí klíče obsahují buď prázdné hodnoty NULL, nebo hodnoty primárních klíčů sloupců z rodičovské tabulky, např. v tabulce zaměstnanců bude existovat sloupec s ID pobočky, který v tabulce poboček bude primárním klíčem [2].

1.1.2 ACID

Vlastnosti ACID zaručují bezpečný a korektní způsob zpracování všech databázových transakcí. Umožňují provádět více stejných operací nad databází současně pro více uživatelů. Tato zkratka obsahuje 4 základní vlastnosti a to atomičnost (z angl. atomicity; A), konzistence (z angl. consistency; C), izolace (z angl. isolation; I) a trvalost (z angl. durability; D).

Atomičnost transakce zaručuje, že je transakce brána jako celek, tudíž nemůže dojít pouze ke zpracování části operací, které daná transakce obsahuje a nedojde tak k chybám v samotné databázi. Příkladem takové transakce může být převod peněz z účtu A na účet B. Celková transakce může proběhnout pouze v případě, že na účtu A dané peníze odečteme a na účet B připsáme. V průběhu této transakce jsou data v databázi nezměněna až do chvíle, kdy daná transakce proběhne v pořádku. Pokud databázový systém po odečtení peněz z účtu A selže, je proveden tzv. rollback, kdy jsou peníze na účet A vráceny zpět – databázový systém je navrácen do původního stavu.

Vlastnost konzistence nám zaručí, že všechny provedené transakce přenesou databázi z jednoho platného stavu do druhého. Veškerá data v databázi musí být platná v souladu se všemi pravidly, včetně různých omezení, kaskád, spouštěčů (trigger) nebo jakékoliv jejich kombinace. Dále zaručuje správnost provedení transakce s ohledem na sémantiku aplikace.

³ Relace = vztah, v tomto případě mezi různými tabulkami

⁴ Cizí klíč = hodnota primárního klíče nám zaručí, že do sloupce můžeme vložit jen určité hodnoty

Izolační vlastnost je zde z důvodu kontroly souběžně probíhajících transakcí. Říká, že každá transakce probíhá nezávisle na druhé a nemá vliv na její existenci. Tedy i v případě průběhu dvou transakcí na pozadí současně jsou uživateli viditelné jako dvě transakce proběhlé v určitém pořadí za sebou. V případě využívání stejných dat je jedna z transakcí přerušena a zavolána později.

Poslední vlastnost, trvanlivost transakce, zajišťuje, že v případě potvrzení úspěšně provedené transakce je tato uložena a nemůže být zrušena nebo ztracena v případě výpadku energie, pádu databázového systému nebo nějaké jeho chyby. Jakmile je tady proveden jakýkoliv příkaz, jeho výsledek musí být permanentně uložen v databázi, dokud ho uživatel sám nezmění, nebo nevymaže. Tyto operace bývají většinou ukládány do energeticky nezávislých pamětí pro případ výpadku proudu [3].

1.1.3 Architektura

Konstrukce každého SŘBD závisí na jeho architektuře. Ta může být rozdělena do jedné nebo více vrstev. N-vrstevní architektura rozděluje celý databázový systém do souvisejících, ale samostatných N modulů, které mohou být nezávisle na sobě dále modifikovány, změněny nebo nahrazeny.

Jednovrstvá architektura obsahuje datovou, aplikační i uživatelskou část v jednom softwarovém balíčku. Veškeré změny provedené nad touto vrstvou jsou okamžitě provedeny v celé databázi.

Dvouvrstvou architekturou se rozumí, že je SŘBD složen ze dvou různých částí, kdy k databázi přibude i samotná aplikace, přes kterou je možno k databázi přistoupit. Aplikace je zcela nezávislá na databázi v případě operací, designu a programování.

Třívrstvá architektura, též někdy označována jako klient/server architektura, je nejrozšířenější a nejpoužívanější architekturou pro návrh databázového systému. Odděluje od sebe vrstvy na základě pokročilosti uživatelů, kteří k nim přistupují. Tyto vrstvy mezi sebou vzájemně spolupracují. Architektura se skládá z:

- datové vrstvy,
- aplikační vrstvy,
- a uživatelské vrstvy.

Datová vrstva obsahuje samotnou databázi. Údaje v této vrstvě jsou nezávislé na aplikačním serveru či business logice (jakým stylem jsou data vytvářena, ukládána nebo upravována). Dále tato vrstva obsahuje vztahy, které nám definují data a jejich vazby.

V aplikační vrstvě je umístěn aplikační server spolu s programy, které přistupují k databázi. Pro uživatele představuje abstraktní pohled na databázi. Uživatel tedy nemá žádné vědomí o databázi za hranicemi této vrstvy. V rámci dané architektury tedy slouží jako prostředník mezi koncovým uživatelem a samotnou databází.

V uživatelské vrstvě už probíhají samotné operace. Umožňuje vytváření různých pohledů na databázi. Všechny tyto pohledy jsou následně umístěny v aplikační vrstvě. Na následujícím obrázku č. 1 je demonstrována komunikace jednotlivých vrstev. Veškerá komunikace mezi datovou a uživatelskou vrstvou probíhá pouze přes vrstvu aplikační [1].



Obrázek 1 - Třívrstvá architektura (vlastní zpracování)

1.1.4 Datový model

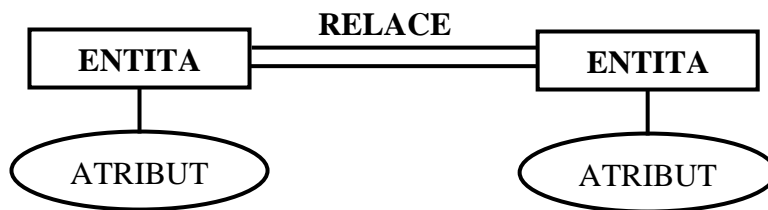
Datový model definuje podobu logické struktury databáze. Určuje, jakým způsobem jsou data zpracovávána, ukládána a jaké jsou mezi nimi vztahy. Mezi úplně první datové modely patřil tzv. plochý datový model, ve kterém byla veškerá data ukládána v rámci jedné úrovně. Obsahovaly mnoho duplicitních dat a značné problémy při jejich aktualizaci.

Relačně-entitní model je založen na představě o reálných entitách a vztazích mezi nimi. Při formulování scénáře z reálného světa a jeho následném vložení do databázového modelu si model sám vytváří množinu entit, množinu vztahů mezi nimi, obecné vlastnosti a omezení. Tento model je tedy založen na entitách, včetně jejich atributů a vzájemných vztazích (relací) mezi jednotlivými entitami.

Entitou se rozumí objekt z reálného světa, která je v rámci modelu reprezentována tabulkou. Každá tabulka má vlastnosti dané entity uložené v pojmenovaných sloupcích – attributech. Pro každý atribut je definována množina přípustných hodnot, které může obsahovat. Tato množina se nazývá doména. Relací se rozumí vztah mezi entitami. Na základě počtu vztahů mezi dvěma entitami je určena kardinalita relací, podle níž jsou vztahy rozděleny na 4 druhy a to:

- 1 : 1 (angl. one to one),
- 1 : N (angl. one to many),
- N : 1 (angl. many to one),
- a N : N (angl. many to many).

Na následujícím obrázku č. 2 jsou demonstrovány relace (vztahy) mezi entitami (tabulkami), které se váží na tabulky jako celek, vytváří se na základě daného atributu (hodnoty sloupce), který přebírá roli primárního klíče.



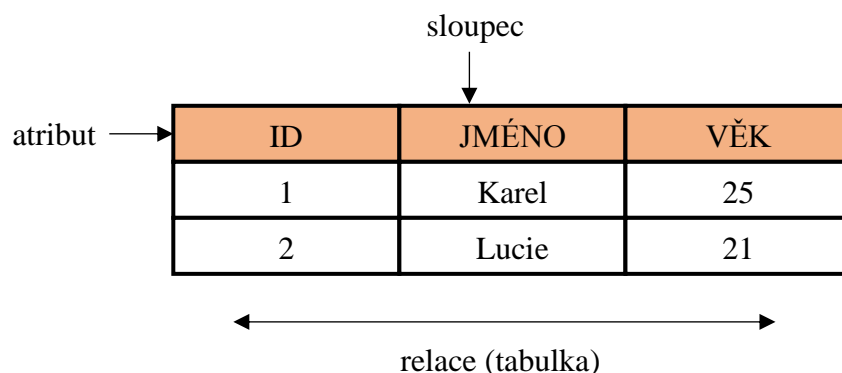
Obrázek 2 - Relačně-entitní model (vlastní zpracování)

Relační model je nejčastěji používaným modelem pro SŘBD. Je založen na predikátové logice a definuje tabulku jako N-relaci.

Hlavními rysy tohoto modelu jsou:

- data uložena v tabulkách, nazývány jako relace,
- relace mohou být normalizovány,
- v normalizovaných relacích jsou data uložena jako atomické jednotky,
- každý řádek v relaci obsahuje jedinečné atributy,
- každý sloupec v relaci obsahuje atributy ze stejné domény.

Na následujícím obrázku č. 3 je vyobrazen zjednodušený vzor relačního modelu [4].



Obrázek 3 – Relační model (vlastní zpracování)

1.1.5 Databázové schéma

Databázové schéma tvoří páteřní strukturu celé databáze. Definuje entity a entitní vztahy, na jejichž základě může být vytvořen diagram, který pomáhá pochopit celkovou stavbu databáze. Může být rozřazen na dva typy – logické a fyzické schéma.

Fyzické schéma popisuje formu, ve které jsou data ukládána – obsahuje všechny informace o databázi potřebné k vytvoření vztahů mezi tabulkami, popř. informace potřebné k ladění výkonnosti. Mezi tyto informace patří definice omezení, propojovací tabulky či zda daná databáze využívá indexy pro zrychlení vyhledávání informací, kdy je definována unikátní hodnota sloupce tabulky. Dále definuje, jakým způsobem budou data ukládána do sekundárního úložiště. Tato data jsou modifikovaná nezávisle na logickém schématu, např. v případě vylepšení úložiště, kdy probíhá výměna HDD za SSD, nijak neovlivňuje data z logického schématu.

Oproti tomu *logické schéma* obsahuje soubor všech logických omezení, která je potřeba aplikovat na uložená data. Definuje tabulky, pohledy a integritní omezení. Nezávislost na fyzickém schématu v tomto případě zajišťuje, že případné změny formátu v tabulce neovlivní uložená data v pevném úložišti [1].

1.1.6 SQL

SQL (angl. Structured Query Language) je programovací jazyk využíván v relačních databázích. Jeho příkazy se dělí do více skupin na základě prováděných činností.

Příkazy DDL (z angl. Data Definition Language) se využívají pro definování databázového schématu. Obsahuje příkazy jako [5]:

- CREATE – pro vytvoření databáze, tabulky, pohledu atp.
- DROP – pro smazání databáze, tabulky, pohledu atp.
- ALTER – pro modifikaci databázového schématu, tabulky atp.

Příklad použití příkazů DDL:

```
CREATE TABLE nazev_tabulky;  
DROP TABLE nazev_tabulky;  
ALTER TABLE nazev_tabulky ADD nazev_sloupce datovy_typ;
```

Příkazy ze skupiny DML (z angl. Data Manipulation Language) jsou určeny pro manipulaci s daty. Uvnitř této skupiny příkazů se nalézají příkazy jako [5]:

- SELECT/FROM/WHERE – pro vybrání konkrétních dat ze zvoleného umístění v databázi za splnění určitých podmínek
- INSERT INTO/VALUES – pro vložení hodnot do řádků v tabulce (relaci)
- UPDATE/SET/WHERE – pro aktualizaci hodnot sloupců v tabulce (relaci)
- DELETE FROM /WHERE – pro smazání jednoho či více řádků

Příklad použití příkazů DML:

```
SELECT nazev_sloupce FROM nazev_tabulky WHERE nazev_sloupce = hodnota;  
INSERT INTO nazev_tabulky(nazev_sloupce) VALUES (hodnota);  
UPDATE nazev_tabulky SET nazev_sloupce = „příklad“  
  WHERE nazev_sloupce = „hodnota“;  
DELETE FROM nazev_tabulky WHERE nazev_sloupce = hodnota;
```

Další skupinou příkazů je DCL (z angl. Data Control Language) pro kontrolu přístupových práv. Obsahuje 2 hlavní příkazy a to [5]:

- GRANT – pro povolení přístupu a manipulace s daty pro uživatele
- REVOKE – pro odebrání přístupových a manipulačních práv uživatele

Příklad použití příkazů DCL:

```
GRANT UPDATE (nazev_sloupce) ON nazev_tabulky TO jmeno_uzivatele;  
REVOKE DELETE (nazev_sloupce) ON nazev_tabulky TO jmeno_uzivatele;
```

Dále jazyk SQL obsahuje příkazy pro řízení transakcí (TCL, z angl. Transaction Control Language). Mezi tyto příkazy patří [5]:

- BEGIN Transaction – pro otevření dané transakce,
- COMMIT Transaction – pro vykonání dané transakce,
- ROLLBACK Transaction - pro navrácení do stavu před provedením transakce.

2 PŘEHLED DATABÁZOVÝCH SYSTÉMŮ

Následující kapitola se zabývá obecným přehledem vybraných databázových systémů společností Oracle Corporation, Microsoft, MySQL, PostgreSQL a SAP HANA.

2.1 Databázový systém ORACLE DATABASE

2.1.1 Obecná charakteristika

Oracle Database je multiplatformní⁵, objektově-relační databázový systém vybaven pokročilými nástroji pro zpracování dat, vysokým výkonem a snadnou škálovatelností⁶. Jeho spuštění je možné na OS Windows, MacOS a Linux. Aktuální verzí je Oracle Database 12c s datem vydání 1. března 2017. Označení „c“ znamená *cloud* – tato verze podporuje nově multitenantní architekturu, která umožňuje vytvářet zásuvné databáze (PBD) v multi-kontejnerové databázi (CBD). Starší verze byly označovány písmeny „i“ (toto označení odráželo podporu pro *Internet* v rámci vestavěného JVM) a „g“ (pro *grid computing*, který je blíže popsán v podkapitole 2.1.3). Tento databázový systém podporuje jak dotazovací jazyk SQL, ale také firemní rozšíření Oracle, např. pro hierarchické dotazy, PL/SQL pro vytváření uložených procedur, uživatelských funkcí, spouštěčů (triggers) a balíků. Podporuje také databáze uložené v hierarchickém modelu dat (XML databáze). Vše je zákazníkovi zpřístupněno pomocí dobře propracovaného cloudu. Podporuje programovací rozhraní jako JAVA, C/C++ i např. Visual Basic či propojení s .NET.

Tabulka 1 – Oracle Database v číslech (Oracle Corporation 2016)

LIMIT	HODNOTA
Maximální velikost databáze	2 PB (se standardními 2k bloky)
Maximální velikost tabulky	4 GB
Maximální velikost řádku	8 KB
Maximální počet řádků v tabulce	Omezeno velikostí tabulky
Maximální počet sloupců v tabulce	1000
Maximální počet indexů v tabulce	Neomezeno

2.1.2 Architektura

Oracle Database využívá třívrstvého typu architektury, popsaného v předchozí kapitole č. 1.1.3.

Klient tedy v této architektuře iniciuje požadavky na operace, které budou provedeny na serveru. Může to být například webový prohlížeč nebo jakýkoli koncový uživatelský proces.

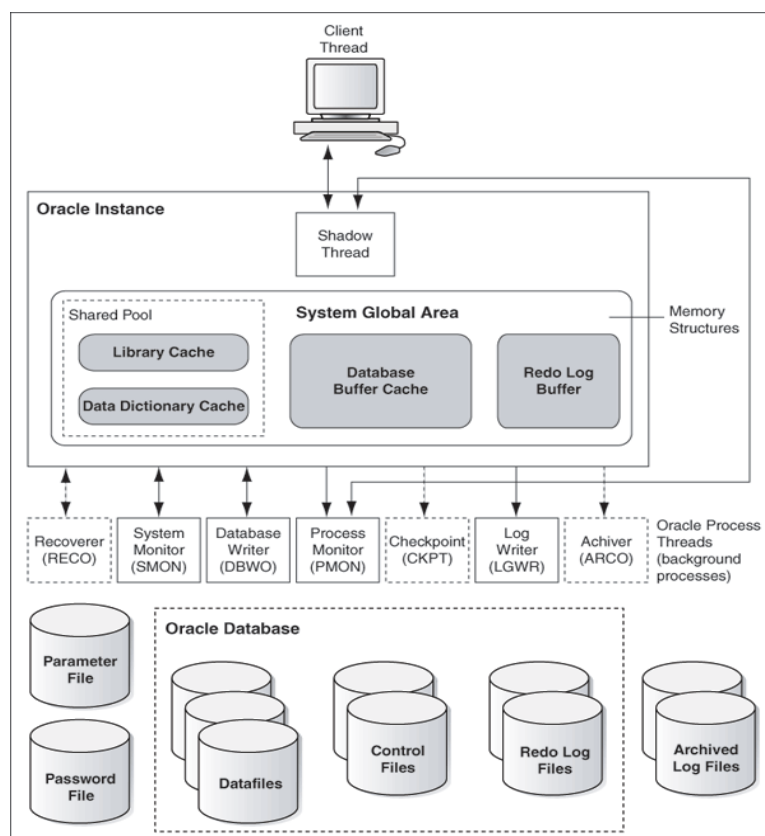
⁵ Multiplatformní – lze spustit na více než jedné platformě (př. Microsoft Windows s architekturou x86 i Mac OS X s architekturou PowerPC)

⁶ Škálovatelnost – schopnost pracovat s náhlými změnami zátěže SW, vlastnost hospodárnosti SW – pokud není aktuálně potřeba vysoký výkon

Připojuje se k databázovému serveru přes jeden nebo více aplikačních serverů. Viz obrázek č. 4, část Client Thread.

Aplikační server zajišťuje klientovi přístup k datům. Slouží jako prostředník mezi klientem a databázovým serverem, který poskytuje další úroveň zabezpečení. Dále umožňuje uchovávání zpracovaných dotazů pro klienta do tzv. front, které odstraňují další nároky na databázový server. Viz obrázek č. 4, část Oracle Instance.

Databázový server poskytuje data požadovaná aplikačním serverem na účet klienta a zpracovává dotazy. Může kontrolovat operace, které provádí aplikační server sám, např. připojení k databázovému serveru, nebo které mu nařídil uživatel, tj. zobrazení požadovaných dat. Databázový server spouští software společnosti Oracle a ovládá funkce potřebné pro souběžně sdílený přístup k databázi. Viz obrázek 4, spodní část.



Obrázek 4 - Architektura Oracle Database (Oracle Corporation 2016)

Navzdory tomu, že lze uživatelskou aplikaci a databázi Oracle spustit na jednom PC, vyšší účinnosti je dosaženo v případě spuštění každé části na jiných zařízeních propojených pomocí sítě. V tomto rozděleném zpracování databázového systému je nutné užití více než jednoho procesoru umístěného v různých databázových systémech z důvodu zpracování individuálních úloh. [6]

2.1.3 Oracle Database 10g

Verze 10g byla první navrženou pro tzv. grid computing, kdy probíhá zpracování úloh na více počítačích. Byla nejvíce flexibilním a nákladově efektivním způsobem pro správu podnikových informací. Tím bylo zajištěno snížení nákladů na správu a zároveň bylo dosaženo nejvyšší kvality služeb. Tato verze byla v té době první podporovanou pro linuxové distribuce. Zároveň se v ní vyskytuje automatická správa databáze a konzole Database Control, které sama umí označit špatně napsaný aplikační kód, navrhnout lepší a automaticky vyladit databázi pro dosažení co nejvyššího výkonu.

Klíčovým prvkem pro grid computing byla technologie databázového clusteringu Oracle Real Application Clusters, s jejíž pomocí lze bez úprav provozovat komerční podnikové aplikace a průběžně do clusteru přidávat další servery a úložné systémy.

Mezi další novinky patří software Automatic Storage Management (ASM), který značně zjednodušuje správu a konfiguraci úložného systému databáze. To automatizuje ukládání databázových souborů a dat, čili uživateli uleví od nutnosti nákupu nákladných softwarů pro správu disků.

Samotná databáze ve verzi 10g byla navržena tak, aby byla schopna provozu i na malých serverech, clusterech i podnikových výpočetních sítích. [7]

2.1.4 Oracle Database 11g

Další verze Oracle Database přináší novou technologii Oracle Real Application Testing. Jako první databáze tedy pomáhá zákazníkovi testovat a řídit změny v jeho informačním prostředí.

Nově obsáhlá funkce Data Guard přináší možnost použití záložní databáze nejen pro ochranu při poruše databázového systému, ale i ke zvýšení výkonu prostředí. Umožňuje obnovu záložní databáze, kterou lze využít k zálohování, testování a implementaci upgradů.

Ve verzi 11g se značně rozšiřují možnosti v oblasti partitioningu a komprese dat. Velká řada manuálních operací v rámci členění dat na oddíly je zde zautomatizována a rozšířena o možnosti členění dat pomocí rozsahů, hashovacích funkcí či využití virtuálních sloupců. Nyní lze dosáhnout dvou či trojnásobného poměru datové komprese.

Oracle Total Recall je další přidanou funkcí, která umožňuje administrátorovi definovat tabulky, pro které se bude dlouhodobě ukládat historie o proběhlých úpravách. Vybraní uživatelé tedy mohou pracovat s informacemi v rámci určitého data či sledovat proběhlé změny.

Další změnou prošlo i vylepšení výkonu technologie XML DB, díky níž jsou uživatelé schopni ukládat a manipulovat s daty XML. Verze 11g začala podporovat binární XML, což rozšířilo možnosti volby ukládání XML dat dle požadavků konkrétní aplikace.

Velkou změnou prošlo i sdružení připojení a práce s mezipamětí. Oracle si zde upevňuje svoji vedoucí pozici v rámci výkonu a škálovatelnosti novou funkcí Query Result Cache, která s pomocí mezipaměti rapidně zvyšuje výkon a škálovatelnost celé aplikace, kdy opakovaně využívá výsledky často volaných databázových dotazů. Database Resident Connection Pooling oproti tomu zvyšuje možnost rozšíření webových systémů sdružením připojení jednovláknových aplikací [8].

2.1.5 Oracle Database 12c

Aktuální a nejnovější verze Oracle Database nese označení 12c. Oproti předchozím verzím, které byly označovány písmenem g (z angl. grid computing), nejnovější verze je označena písmenem c z důvodu příchodu nové technologie cloud. Přichází tedy jako nová generace databáze, která má splňovat požadavky cloudových řešení.

Verze 12c využívá nové multitenantní architektury. Každá databáze zapojená do této nové architektury se aplikaci jeví jako standardní databáze Oracle s pomocí nové funkce Oracle Multitenant. Existující aplikace tedy neprojdou žádnou změnou. Zároveň tato funkce umožňuje spravovat několik databází jako jeden celek. Tato architektura zajišťuje okamžitý přístup k databázi, popř. její klonování, což z ní dělá ideální platformu pro testování databází.

Další obsaženou funkcí je automatická optimalizace dat. Veškeré proběhlé aktivity databáze jsou monitorovány s pomocí tepelné mapy, čímž ulehčí definování strategií pro správu serverů a automaticky komprimuje a ukládá data na základě jejich aktivity a stáří.

Změnou v této nejnovější verzi prošly i některé datové typy, např. varchar2, NVarchar2 atp. Předchozí limit těchto typů byl 4 K. Ve verzi 12c byl však navýšen na 32, 767 bytů.

Další užitečnou funkcí je využití tzv. neviditelného sloupce. V předchozích verzích bylo nutné využití pohledů, aby byl sloupec skryt. V nově přichozí verzi ho stačí pouze označit jako neviditelný a nebude zobrazen do té doby, pokud nebude v samotném příkazu využíván.

Před 12c nebylo možné vytvořit více indexů na jednom sloupci, nebo jejich skupině. Pokud byl vytvořen index na sloupci 1 nebo sloupcích 1 a 2, nebylo možné vytvořit další index na stejném sloupci nebo sadě sloupců ve stejném pořadí. Ve verzi 12c lze vytvářet více indexů, musí být však stejného typu. Vždy lze však využívat jen jednoho indexu při dané operaci [9].

2.1.6 Memory cache

Používání in-memory databází se v posledních letech velice rozšířilo. Tyto databáze jsou rychlejší než databáze optimalizované pro práci s diskem, protože přístup na disk je pomalejší než přístup k paměti. Interní optimalizační algoritmy jsou zde jednodušší a spouštějí méně instrukcí pro CPU, také přístup k datům v paměti eliminuje čas vyhledávání při vykonávání dotazů.

Oracle In-Memory Database Cache (IMDB Cache) je novým produktem Oracle Database pro zachycení kritických podmnožin výsledků, ale i například celých tabulek do mezipaměti v aplikační vrstvě právě z důvodu snížení odezvy.

IMDB Cache je vestavěna do programu s pomocí využití Oracle TimesTen In-Memory databáze, která je zde z důvodu lepšího provozu pro aplikace, které slouží pro více uživatelů. Aplikace se připojí do cache databáze a dále k ní přistupuje pomocí standartního SQL. Synchronizace veškerých dat z cache do databáze je prováděna automaticky.

Oracle Database In-Memory poskytuje architekturu ve dvou formátech, která umožňuje současnou reprezentaci tabulek v paměti za pomoci tradičního formátu řádků a nového formátu sloupců. Oracle SQL Optimizer automaticky převede analytické dotazy na formát sloupců a OLTP dotazy na formát řádků. Formát sloupce není na disku pevný, tudíž neklade další náklady na ukládání či synchronizaci s úložištěm. Nejedná se tedy o úplné ukládání databáze, jak by nám mohl název naznačit, ale o ukládání sloupců, ke kterým se může databáze nebo její oddíly v mezipaměti připojit.

Mezipaměť IMDB je navržena tak, aby pokračovala v provozu i po problémech na databázovém serveru nebo ztrátě internetového připojení. Transakce po příkazu commit jsou uchovávány a po obnovení spojení vloženy do databáze Oracle. Stejně tak jsou uchovávány i odchozí transakce na zdrojových tabulkách [10, 11].

2.1.7 Zpracování dotazů

Zpracování SQL dotazů obsahuje bloky analýzy daného dotazu (ověření správnosti, parsování), optimalizace⁷, generování zdrojových řádků⁸ a samotné provedení dotazu nad databází podle vybraného exekučního plánu. V některých situacích lze určité fáze zpracování vynechat.

⁷ Optimalizace = výpočet využití operační paměti a systémových prostředků při použití různých exekučních plánů

⁸ Generátor zdrojového řádku vytváří strom zdrojového řádku, který obsahuje informace o uspořádání tabulek, na které je odkazován, metodu přístupu k daným tabulkám, metodu propojení s tabulkami, které daný příkaz ovlivňuje a datové operace – filtry, řazení či agregace

První fázi zpracování dotazu je analýza, též označována jako proces parsování. Ta zahrnuje oddělení částí příkazů SQL do datové struktury, se kterou jsou další části zpracování dotazu schopny pracovat. Celé parsování je řízeno aplikací, tedy samotná databáze nemůže určit počet výsledných částí.

Během parsování provádí databáze následující kontroly:

- lexikální analýza,
- kontrola syntaxe,
- kontrola sémantiky,
- a kontrolu sdílení pool.

V průběhu kontroly syntaxe musí aplikace zkontrolovat správnost SQL příkazu. Prvním krokem kontroly správnosti dotazu je lexikální analýza, kdy je vstupní řetězec znaků rozdělen tzv. lexémy (jednotky, které obsahují čísla, operátory, klíčová slova atp.). Tato analýza má také za úkol odstranění komentářů či bílých znaků. Tyto jednotlivé znaky jsou dále předány k syntaktické kontrole, která zjišťuje, zda posloupnost symbolů tvoří povolený výraz. Na základě této kontroly může daný příkaz selhat v případě nesprávného napsání výrazu (př. dotaz obsahuje FORM místo klíčového slova FROM). Následuje kontrola sémantiky, která již řeší význam celého dotazu, například zda objekty a sloupce v příkazu existují. Syntakticky korektní příkaz může selhat při sémantické kontrole v případě provádění dotazu nad neexistujícím objektem.

Další kontrola probíhá s pomocí sdíleného poolu, ve kterém se uchovávají příkazy zadané uživateli z předešlých činností. Na základě nalezení či nenalezení dotazu v poolu se rozhoduje, zdali se provede hard nebo soft parse. Shody se nalézají pomocí přidělených hash hodnot, který má každý již zpracovaný příkaz.

V případě nenalezení dotazu v poolu se přistupuje k hard parsování. V tomto případě musí aplikace vytvořit spustitelnou verzi aplikačního kódu a přistoupit k dalším fázím kontroly. V případě DDL dotazů je hard parse prováděn vždy.

V opačném případě, tedy daný příkaz byl v poolu nalezen, je přistupováno k soft parsu, kdy je použit již existující aplikační kód. Obecně je tento způsob upřednostňován, protože je možné vynechat kroky optimalizace a generování zdrojových řádků a pokračuje se přímo k provedení samotného dotazu.

Následující fází při vyhodnocení parsování jako hard se přistupuje k optimalizaci. Dochází zde k vytvoření exekučních plánů, které databázi určí, v jakém pořadí má databáze vyhodnotit výsledky zadaného příkazu. Rozhoduje se na základě skóre, které může uživatel ovlivnit pomocí nápověd pro vytváření daných plánů. Ze všech vygenerovaných exekučních plánů se vybere jeden optimální, který se odešle do generátoru zdrojového kódu a následně je připraven ke zpracování v databázi.

PLAN_TABLE_OUTPUT							
1	Plan hash value: 3028223791						
2							
3	-----						
4	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
5	-----						
6	0	SELECT STATEMENT		5	110	3 (34)	00:00:25
7	1	SORT ORDER BY		5	110	3 (34)	00:00:25
8	* 2	TABLE ACCESS FULL	TRPASLICI	5	110	2 (0)	00:00:17
9	-----						
10							
11	Predicate Information (identified by operation id):						
12	-----						
13							
14	2	filter("NAROZEN">'01.01.82')					

Obrázek 5 - Exekuční plán Oracle Database (vlastní zpracování)

Na předchozím obrázku je vyobrazen exekuční plán pro dotaz:

```
select * from a_o_snehurce.trpaslici
where narozen >'01.01.82'
order by jmeno ASC;
```

První provedenou částí dotazu byla filtrace dle sloupce narození, následovaná sběrným přístupem do dat v tabulce, které vyhovovaly danému filtru. Pokračovalo se setříděním podle sloupce jména a následně došlo k provedení příkazu SELECT. Exekuční plán byl tedy proveden od nejvíce vnořené části dotazu. Dále si můžeme povšimnout, že každá část byla ohodnocena cenou, tedy časem, který byl vynaložen procesorem na její provedení [12].

2.2 Databázový systém MICROSOFT SQL SERVER

2.2.1 Obecná charakteristika

Microsoft SQL Server je relační databázový systém vyvinutý společností Microsoft. Je to softwarový produkt s primární funkcí ukládání a načítání dat podle požadavků z jiných softwarových aplikací. Existuje nespočet vydaných verzí, zaměřených na různé cílové skupiny a pracovní úlohy od malých aplikací až po velké, které využívá velká masa uživatelů. Aktuální verzí je Microsoft SQL Server 2016, která byla vydána 1. června 2016 a je podporována je pouze na 64bitových procesorech. Dostupný je pro OS Windows a Linux v 11 světových jazycích. Podporuje programovací jazyky z .NET platformy – včetně Visual Basic .NET, Visual C# a C++, rozumí si např. i Javou. V posledních pár letech se pyšní titulem „Nejméně zranitelná databáze“ díky jeho vícevrstvému propracovanému systému ochrany. V poslední verzi Microsoft představil svůj nejmodernější ochranný systém s názvem Always Encrypted.

Tabulka 2 - Microsoft SQL Server v číslech (Microsoft 2016)

LIMIT	HODNOTA
Maximální velikost databáze	524 272 TB
Maximální velikost tabulky	524 272 TB
Maximální velikost řádku	8 060 b
Maximální počet řádků v tabulce	Limitováno velikostí úložiště
Maximální počet sloupců v tabulce	30 000
Maximální počet indexů v tabulce	Neomezeno

2.2.2 Architektura

Firma Microsoft pro svůj databázový systém využívá třívrstvé architektury. Hlavní části této architektury se dělí na relační modul, úložný modul a modul s SQL OS.

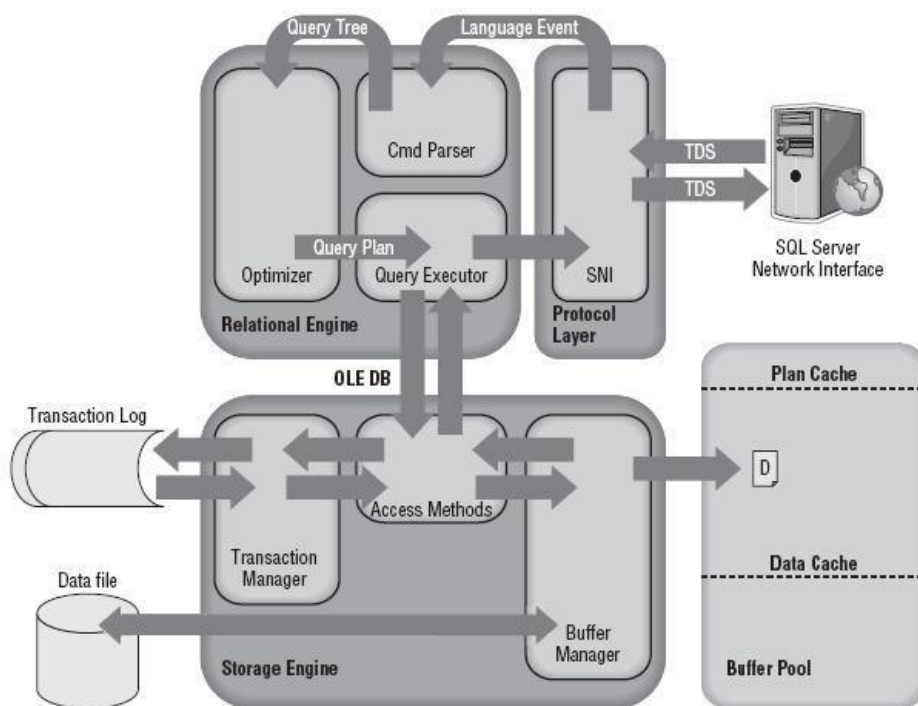
Relační část MS SQL zodpovídá za činnosti parsování, optimalizace a spuštění dotazů. Určuje tedy, co daný dotaz bude vykonávat a nejlepší způsob, jakým to provede. Na obrázku č. 6, v části Relational engine, jsou znázorněny jednotlivé části pro jednotlivé funkce – Cmd parser, který se stará o parsování daných dotazů a provádí lexikální, sémantické a syntaktické analýzy; Optimizer, který generuje exekuční plány pro dané dotazy a Query Executor, který s pomocí spojení s paměťovým modulem provádí dané dotazy.

Dále SQL Server obsahuje paměťový modul, tedy část, která spravuje datovou vrstvu celého databázového systému. Je odpovědný za ukládání a načítání dat do úložného systému. Na obrázku č. 6 v sekci Storage Engine jsou vyobrazeny jeho části – transaction manager, který spravuje transakce a buffer manager, který spravuje vyrovnávací paměť a všechny vstupně/výstupní operace nad fyzickými soubory. V rámci každé databáze v SQL serveru

existují dva typy souborů, které jsou vytvořeny na úrovni disku – datový soubor obsahující databázi (na obrázku č. 6 část Data File) a množina protokolů (na obrázku č. 6 část Transaction log). Datový soubor se fyzicky ukládá, na rozdíl od protokolů, které slouží pro ukládání transakcí prováděných v rámci databáze.

Další částí architektury SQL Serveru je jeho operační systém (SQL OS), který je uložen na hostitelském PC společně s SQL Serverem. Řídí veškeré činnosti na databázovém stroji. Jedná se o vysoce konfigurovatelný operační systém s výkonným aplikačním programovacím rozhraním, který umožňuje souběžnou činnost více úloh. Poskytuje různé služby operačního systému, jako je podpora tzv. buffer poolu, který zajišťuje rychlou odezvu při získávání dat z databáze (od verze 2014 lze jeho výchozí velikost rozšířit např. na SSD), či detekci deadlocku s pomocí blokovacích a logických struktur. Jiné jeho služby zahrnují zpracování výjimek, hosting pro externí komponenty, jako je např. Common Language Runtime [13].

Služba SQL Server Service Broker poskytuje nativní podporu pro zasílání zpráv a vytváření aplikačních front v databázovém systému SQL Server. To usnadňuje vývojářům vytvářet sofistikované aplikace, které používají součásti Database Engine pro komunikaci mezi různými databázemi. Vývojáři mohou pomocí služby Broker snadno vytvářet distribuované a spolehlivé aplikace.



Obrázek 6 - Architektura MS SQL (Microsoft 2016)

2.2.3 Microsoft SQL Server 2012

Tato verze MS SQL byla dlouho očekávanou, jelikož mezi ní a jejím předchůdcem stály dlouhé 4 roky. Mezi nepřehledné množství nových funkcí jistě patří nové pojetí tzv. zrcadlení databáze. Sekundární kopie jsou nyní čitelné a mohou být použity pro zálohu celé databáze. Dále přichází podpora Windows Server Core – verze operačního systému Windows bez grafického uživatelského rozhraní, která využívá DOS PowerShell pro interakci s uživatelem. Ta přináší až o 50 % méně využívané paměti, místa na disku a je bezpečnější než plná instalace MS SQL.

Od verze 2012 je nyní možné indexovat sloupce. V této době byla tato funkce pro MS SQL zcela jedinečná. Jedná se o speciální typ indexu jen pro čtení, který je určen pro použití s dotazy datových skladů. V dřívějších dobách také nebylo možné poskytnout přístup např. rozvojovému týmu ke čtení a zápisu do každé databáze na sdíleném serveru bez použití neregistrovaných procedur, či zdlouhavé manuální práce. MS SQL tedy od verze 2012 nabízí možnost vytvoření role, která poskytuje práva pro čtení a zápis na každé databázi na serveru.

Uživatelé, kteří do této verze měli oblíben databázový systém od společnosti Oracle z důvodu podpory vytváření sekvenčních objektů – tzv. počítadel, které slouží např. ke zvýšení hodnot v tabulce na základě využití spouštěče (trigger), tuto funkci naleznou nyní i u této verze společnosti Microsoft. Důležité je zmínit i nově přidané ovladače pro některé Linuxové distribuce [14].

2.2.4 Microsoft SQL Server 2014

Nejdůležitější a největší změnou v této verzi je bezpochyby nová funkce In-Memory OLTP (v době svého vývoje označován názvem Hekaton). Přesunutím vybraných tabulek a uložených procedur do mezipaměti značně snížilo výpočetní nároky na celou databázi. Jedním z důležitých chybějících parametrů u optimalizovaných tabulek byla podpora cizích klíčů. V době vydání Microsoft sliboval až 20x vyšší výkon aplikace. Vydáním Hekatonu se posunul na úroveň společností Oracle a SAP, kteří In-Memory využívaly od dřívějších verzí.

Další změnou prošel i SQL Server. Od verze 2014 je možnost zvýšení celkového počtu procesorů až na 640 s využitím až 4 TB fyzické paměti. Při spuštění ve virtuálním stroji lze využít až 64 virtuálních procesorů s 1 TB paměti. Nově také podporuje úložiště SSD.

Do této verze bylo k vytvoření šifrovaných záloh potřeba softwaru třetích stran. Od nynějška MS SQL využívá šifrovacích algoritmů Advanced Encryption Standard (AES) 128, AES 192, AES 256 a Triple DES (3DES) [15].

2.2.5 Microsoft SQL Server 2016

Z důvodu rostoucího zájmu o levnější úložiště přichází s novou funkcí poskytování tzv. „Stretch Database.“ Princip této databáze spočívá v přerozdělení dat mezi fyzickým úložištěm a cloudovým úložištěm Azure SQL. Optimalizátor dotazů je nastaven tak, že obsahuje informace o tom, kde se daná část dat nachází a automaticky přerozdělí výkon mezi tyto dvě úložiště. Zpoždění v obdržení výsledků dotazů závisí pouze na rychlosti odezvy sítě. I přes fakt, že je využívání Azure SQL zpoplatněno, jsou náklady na jeho využití rapidně nižší než zakoupení dodatkového fyzického úložiště.

S novou verzí přibyla i podpora pro komunikaci s databází prostřednictvím JSON (JavaScript Object Notation). Do této doby bylo toto nutné provádět manuálně pomocí softwaru třetích stran, ovšem MS SQL 2016 nyní podporuje ukládání relačních dat ve formátu JSON a jeho následný export. Jednou z největších novinek v MS SQL 2016 je ovšem také výskyt příkazu ALTER TABLE. Do této doby změna tabulky nebyla možná, musela se celá smazat a vytvořit znovu s úpravami. Nově také přibyla možnost využití cizích klíčů u optimalizovaných tabulek. [16]

2.2.6 Memory cache

In-Memory OLTP, také znám jako „Hekaton“, byl představen již v SQL Server 2014. Tato výkonná technologie umožňuje využít velké množství paměti pro zvýšení výkonu a je optimalizována pro zpracování online transakcí. Zároveň je integrována do databázového stroje SQL Serveru. Obsahuje především dvě nové datové struktury – paměťově optimalizované tabulky a nativně kompilované uložené procedury.

Optimalizované tabulky ukládají data do paměti s pomocí více verzí dat z jednotlivých řádků. Celá tabulka je tedy uložena v paměti, včetně řádků s možností uložení jako trvalých nebo dočasných dat. Druhá kopie tabulky je uložena na pevném disku v souladu s dodržáním pravidla o trvanlivosti tabulky (pokud je tato možnost povolena). Tabulky jsou čteny z disku pouze v případě obnovy databáze.

In-Memory OLTP je integrován s relačním enginem SQL Serveru a je transparentně přístupný pomocí stejných rozhraní. Ve skutečnosti tedy samotný uživatel ani nepozná, že pracuje s optimalizovanými tabulkami místo dřívějšími diskovými. Avšak interní chování a možnosti

OLTP jsou značně rozdílné. Optimalizované tabulky mohou buď splňovat nebo nesplňovat trvanlivost. Výchozí hodnota je nastavena na trvanlivost – tyto tabulky splňují samozřejmě i zbývající vlastnosti ACID. Operační systém si sám vytváří soubory kontrolních bodů pro obnovu trvanlivých tabulek při pádu databázového systému [17].

2.2.7 Zpracování dotazů

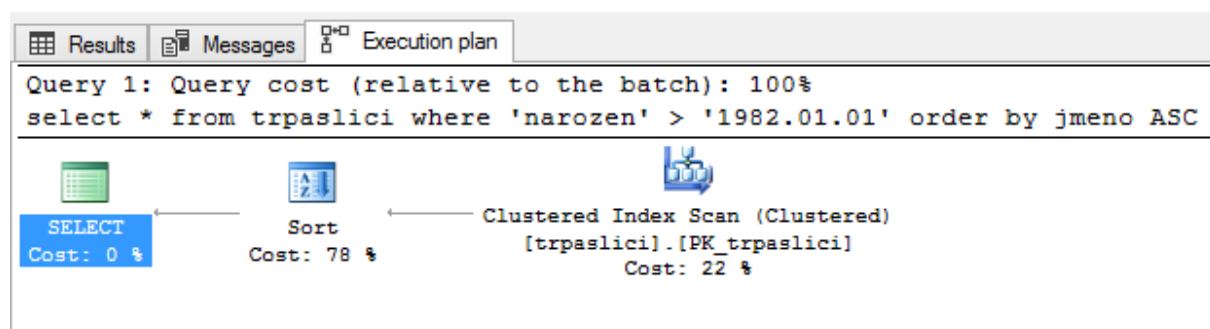
Zpracování dotazu u MS SQL se skládá z pěti částí.

V první části dochází k analýze celého SQL dotazu. SŘBD si celý dotaz rozdělí na jednotlivá slova, která jsou uložena do tzv. tokenů. Tímto se ujistí, zda má daný dotaz platné výrazy. Lze tedy odhalit, chyby v syntaxi a pravopisu.

Ve druhé fázi se odehrává ověření příkazu. Kontrola probíhá na základě systémového katalogu, tzn. zda se daná tabulka z dotazu nachází v databázi, zda v dané tabulce existují všechny sloupce a jestli tyto jednoznačně pojmenovány či zda má uživatel dostatečná oprávnění k provedení dotazu. Probíhá tedy kontrola sémantiky.

Třetí fáze generuje přístupový plán pro daný příkaz. Tento plán je binární reprezentací kroků, které jsou nutné pro provedení příkazu. Je to jakýsi ekvivalent SŘBD pro spustitelný kód.

V předposledním kroku zpracování dotazu se optimalizuje daný přístupový plán. Prozkoumává různé způsoby jeho provedení, tzn. zda může být daný index použit k rychlému vyhledávání, nebo má nejdříve dojít k vyhledání v tabulce A, poté ho připojit k tabulce B nebo se mají dané tabulky spojit a až pak přejít k procesu vyhledávání. Po vybrání optimálního plánu lze přistoupit ke kroku 5, tedy samotnému zpracování dotazu.



Obrázek 7 - Exekuční plán MS SQL (vlastní zpracování)

Na předchozím obrázku č. 7 máme exekuční plán MS SQL pro dotaz:

```
select * from trpaslici
where 'narozen' > '1982-01-01'
order by jmeno ASC;
```

Hned prvním rozdílem od předchozího exekučního plánu je jeho struktura. MS SQL vizualizuje plány jako strom, který čteme zleva doprava. Tímto způsobem je zároveň vyhodnoceno, jak probíhá. Na každém prvku můžeme analyzovat náklady na CPU – časové využití CPU, které je vynaloženo na zpracování dané sady výsledků, popř. počet vrácených řádků. Tyto atributy jsou nejdůležitější pro optimalizaci dotazů [18].

2.3 Databázový systém MYSQL

2.3.1 Obecná charakteristika

MySQL je relačně založený databázový systém. Původně byl vytvořen švédskou firmou MySQL AB, dnes je jeho vlastníkem společnost Oracle Corporation. Dostupný je jak pod bezplatnou licenci GPL včetně zdrojových kódů, tak pod placenou licenci. Z důvodu snadné implementace – podpory mnohých operačních systémů – a volné dostupnosti je v současné době jedním z nejčastěji používaných. Vyskytuje se jako základní software pro webové servery v kombinaci s GNU/Linux, Apache a PHP jako technologie LAMP. V začátcích vývoje byl optimalizován především pro rychlost čtení a zpracování dat na úkor některých pokročilých funkcí – využívá jednoduchých způsobů zálohy, dlouhá léta mu chyběla podpora pohledů, spouštěčů (triggers) a uložených procedur. Toto bylo z důvodu jeho dřívějšího užívání jako datového úložiště, namísto databázového systému. V dnešní době je využíván na mnoha webových stránkách jako jsou např. Google, Facebook, Twitter atp. Aktuální verze je MySQL 5.7.

Tabulka 3 - MySQL v číslech (MySQL 2016)

LIMIT	HODNOTA
Maximální velikost databáze	Neomezeno
Maximální velikost tabulky	MyIsam: 256 TB, Innodb: 64 TB
Maximální velikost řádku	64 KB
Maximální počet řádků v tabulce	Záleží na typu úložiště a velikosti řádků
Maximální počet sloupců v tabulce	4096
Maximální počet indexů v tabulce	64

2.3.2 Architektura

Architektura MySQL je stejně jako u předchozích databázových systémů založena na vícevrstvé architektuře. Liší se od nich však využitím svých vlastních úložných engineů.

Aplikační vrstva obsahuje nástroje pro manipulaci s připojením, ověřováním uživatele a bezpečností. Nazývá se connection pool. Zajišťuje tedy vlastní připojení daného uživatele k serveru, ověření jeho identity s pomocí uživatelského jména a hesla a po úspěšném přihlášení také zkontroluje příslušná práva daného uživatele pro manipulaci s daty. Dále má na starost

práci s vlákny. Po připojení každý uživatel obdrží jedno vlákno procesu, díky kterému jsou zpracovávány všechny jeho dotazy. Každá MySQL databáze má předem definovaný počet vláken, která jsou přiřazována klientům. Systém přiřazování se zdá být méně náročný než vytváření nových vláken, proto tato vlákna nejsou po odpojení smazána, ale ponechána v paměti, kde čekají na další spojení.

Serverová vrstva se stará o veškeré logické funkcionality. Často je tato vrstva označována jako „mozek“ celého databázového systému. Poskytuje široké spektrum služeb a utilit pro správu a údržbu celého databázového systému, např. provedení zálohy a obnovy dat, tvorbu kopií dat, clusteru atp. Dále obsahuje celkové rozhraní pro manipulační jazyk SQL a tvorbu procedur, pohledů atd. Zároveň obsahuje i parser pro vnitřní strukturu prováděných dotazů či prostor pro uložení cache.

Třetí vrstva už slouží pro skladování samotných dat. MySQL má velké množství variací úložišť, kdy je na uživateli, jaký typ je vhodný právě pro něj. Tímto se velice liší od konkurenčních produktů. Všechna úložiště jsou chápána jako zásuvné moduly a mohou být použity i jen např. pro jednu tabulku. Mezi typy těchto úložišť patří MyISAM, InnoDB, Blackhole, CSV a mnohé další.

MyISAM patří mezi nejstarší typy datového úložiště v MySQL. Podporuje vytváření fulltextových indexů, ukládání prostorových dat a jeho velikost je omezena na 256 TB. Je vhodný pro použití v případě požadavku na rychlost databáze. Často se však přechází k jiným typům úložiště z důvodu toho, že tento typ nepodporuje cizí klíče a transakční zpracování dotazů, tedy nezaručuje konzistenci dat. Všechny tabulky v tomto formátu ukládání jsou rozděleny do 3 částí - .MYD, ve které se nachází veškerá data z tabulky, .MYI, která ukládá veškeré indexy a .FRM, což je soubor typický pro MySQL server a obsahuje veškeré definice tabulky.

InnoDB je jedním z nejčastěji využívaných typů pro ukládání. Na rozdíl od předchozího typu již podporuje transakční zpracování dotazů, je tedy ACID kompatibilní a jsou na něm umožněny operace typu COMMIT a ROLLBACK. Dále je možné uzamknout záznamy v tabulce v rámci 1 řádku, nikoliv v rámci celé tabulky jako u předchozího typu [19].

2.3.3 Verze 5.5

Hlavním rysem této verze je bezpochyby přechod na InnoDB jako výchozí úložiště. Od nynějška tedy mohou uživatelé plně využívat ACID kompatibilní transakce, cizí klíče a obnovování dat v rámci výpadku. Dále byla vylepšena výkonnost databázového systému za

pomoci využití multi-serverových systémových replikací. Dochází tím ke zlepšení spolehlivosti převzetí dané služby při selhání serveru. Nově jsou také podporovány XML soubory a lze z nich přímo vytvořit tabulku. Podporovány jsou 3 formáty. V této verzi se také objevují nově přidané příkazy – SIGNAL a RESIGNAL, které umožňují implementaci výjimek v rámci uložených procedur, funkcí, událostí a spouštěčů. Příkaz SIGNAL umožní předání kódu volané chyby, hodnotu SQLSTATE a samotnou zprávu o chybě. RESIGNAL slouží k vyvolání výjimky SIGNAL po provedení různých úprav v databázi [20].

2.3.4 Verze 5.6

Jednou z nejdůležitějších úprav, kterou verze 5.6 prošla, byla v rámci zabezpečení. To obsahuje novou metodu pro ukládání ověřovacích dat, silnější šifrování pro uživatelská hesla, v unixových systémech dokonce podporu vytvoření náhodného hesla. Zároveň se heslo při zadávání již nezobrazuje. Dále proběhly úpravy i některých datových úložišť, např. InnoDB, které od nynější verze začalo podporovat fulltextové indexy, může tedy plně nahradit MyISAM [21].

2.3.5 Verze 5.7

Z důvodu častých stížností na pomalou odezvu databáze prošla verze 5.7 rapidním vylepšením výkonu - při testování dosáhla až dvojnásobného výkonu oproti předchozí verzi. Rozšířením prošlo i monitorování využívání paměti či funkce Explain Data, díky které je dnes snadnější pochopit činnosti optimalizátoru a dle něj i další ladění výkonu aplikace ku spokojenosti uživatele. Přidána byla funkce pro vícezdrojovou replikace – nyní lze dosáhnout toho, že jediný server ve funkci slave může čerpat data z více master serverů [22].

2.3.6 Memory cache

Memory cache v rámci MySQL ukládá text SELECT příkazů spolu s jejich výsledky, které byly uživateli poskytnuty. V případě, že později bude nad databází proveden stejný dotaz již server obdrží informace z cache a nemusí příkaz provádět znovu. Tato funkce je sdílena mezi různými připojeními, není tedy závislá pouze na jednoho uživatele. Nejvíce využitelná je tato funkce v prostředí, kde tabulky v databázi neprochází tak často nějakou změnou a server obdrhuje převážně stejné typy příkazů. To je typické pro webové servery, které generují dynamické webové stránky na základě obsahu databáze. V případě jakékoliv změny v tabulce jsou všechny přiřazené uložené hodnoty z cache smazány, tudíž nevrací pořád stejná data.

InnoDB ukládá do vyrovnávací paměti hodnoty tabulek a indexů v době, kdy je k nim přistupováno. Pro efektivitu operace čtení nad velkým objemem dat je paměť rozdělena na

stránky, které mohou obsahovat více řádků. Pro efektivnější správu je paměť implementována jako propojený seznam daných stránek. Data, která jsou jen zřídka používána jsou z mezipaměti mazána pomocí varianty algoritmu LRU [23].

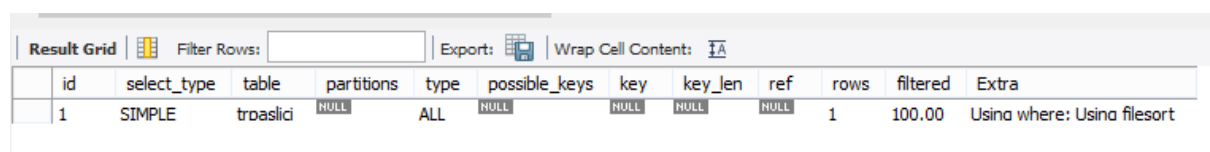
2.3.7 Zpracování dotazů

Nejprve je požadovaný dotaz odeslán klientem na server. Server samotný projde svou cache s uloženými dotazy, v případě nalezení shody klientovi odešle uložený výsledek. V případě neúspěchu je dotaz odeslán do dalších fází zpracování.

Při procesu parsování MySQL rozloží daný dotaz na více malých částí, tzv. tokenů pro vhodné použití v další fázi zpracování dotazu. Vytvoří z nich tzv. parsový strom. Součástí tohoto procesu je také ověření (analýza), zda je daný dotaz syntakticky správný a zda obsahuje základní informace. Zjistí tedy daný typ (zda se jedná o SELECT, INSERT, UPDATE atp.), které tabulky jsou v daném dotazu zahrnuty, zda jsou použity aliasy, co obsahuje klauzule where a zda jsou daná klíčová slova ve správném pořadí. Následně se kontrolují příslušná oprávnění.

Nyní je již parsovaný strom platný a připravený pro optimalizaci, aby se mohl změnit v exekuční plán. Dotaz může být proveden mnoha různými způsoby, které mají vždy stejný výsledek. Úlohou optimalizátoru je nalézt nejlepší volbu. MySQL využívá optimalizátor založený na nákladnosti dotazu a volí nejlevnější. Jednotkou pro náklad dotazu jsou náhodná data o velikost 4 kb.

Nyní se přistoupí k samotnému provedení dotazu na základě vytvořeného exekučního plánu.



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	trpaslici	NULL	ALL	NULL	NULL	NULL	NULL	1	100.00	Using where; Using filesort

Obrázek 8 - Exekuční plán MySQL (vlastní zpracování)

Na předchozím obrázku č. 8 je vyobrazen exekuční plán z MySQL na dotaz:

```
select * from trpaslici
where 'narozen' > '1982-01-01'
order by jmeno ASC;
```

Sloupec ID udává pořadové číslo selectu v dotazu. V dalším sloupci select_type se zobrazuje typ daného selectu – v tomto případě simple, protože není využito vnořených dotazů ani union. Následuje sloupec table, ve kterém je název tabulky, ke které se daný dotaz vztahuje. Jelikož není tato tabulka rozdělena na části, v sloupci partitions je hodnota NULL. Posledním důležitým

parametrem je poslední sloupec, který shrnuje další využití příkazy v SELECT – přidání podmínky where a řazení výsledků vzestupně [24].

2.4 Databázový systém POSTGRESQL

2.4.1 Obecná charakteristika

PostgreSQL je výkonný open source objektově-relační databázový systém. Má za sebou víc než 15 let aktivního vývoje a osvědčenou architekturu. Dobrou reputaci si získal svou spolehlivostí a integritou dat. Jeho spuštění je možné na všech operačních systémech, včetně Linuxových distribucí, MacOS a Windows. Je nejen ACID kompatibilní, ale zároveň plně podporuje cizí klíče, spojení tabulek, tvorbu pohledů, spouštěčů (triggers) a uložených procedur, a to v několika světových jazycích. Podporuje většinu z SQL datových typů, tj. INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL a TIMESTAMP. Dále podporuje ukládání binárně velkých objektů, včetně obrázků, zvuků nebo videí. V základu má programovací rozhraní pro C/C++, Javu, .NET, Perl, Python a mnoho jiných.

Tabulka 4 - PostgreSQL v číslech (PostgreSQL Global Development Group 2016)

LIMIT	HODNOTA
Maximální velikost databáze	Neomezeno
Maximální velikost tabulky	32 TB
Maximální velikost řádku	1.6 TB
Maximální počet řádků v tabulce	Neomezeno
Maximální počet sloupců v tabulce	250–1600 (závisí na typu sloupce)
Maximální počet indexů v tabulce	Neomezeno

2.4.2 Architektura

PostgreSQL využívá jako předchozí databázové systémy třívrstvou architekturu. Každá relace PostgreSQL se skládá ze 3 částí a to:

- Dozorčí proces démona (tzv. postmaster),
- Uživatelská frontend aplikace (např. psql program), a
- Jeden nebo více backendových databázových serverů.

Pro propojení jednotlivých částí se využívají dostupné knihovny. V případě žádosti aplikace o přístup do databáze se zavolá příslušná knihovna, která uživatelskou žádost přepoše přes síť postmasterovi. Ten už se postará o připojení aplikace k databázovému serveru. Od této chvíle veškerá komunikace mezi aplikací a serverem probíhá bez jakéhokoliv zásahu postmastera.

Libpq knihovna patří mezi nejznámější a s její pomocí může jedna aplikace vytvořit mnoho spojení se serverem.

Tato architektura je sestavena tak, že postmaster a backend server běží na jednom stroji současně a tvoří celý databázový server, zatímco aplikace může běžet na jakémkoliv jiném koncovém zařízení [25].

2.4.3 Verze 9.4

S verzí 9.4 přišel nový typ dat JSONB, kdy uživatel nemusí volit mezi relačními a nerelačními datovými úložišti. S touto službou přichází i podpora rychlého vyhledávání. Dále bylo dodáno nové rozhraní API pro čtení, filtrování a manipulaci s replikačním datovým proudem, které podporuje vytváření více jádrových clusterů. Další vylepšení v replikačním systému jsou replikační sloty a přidání časových zpoždění, která zlepšují správu a samotné využití replikačních serverů.

Z výkonnostního hlediska proběhly úpravou GIN indexy, což činí PostgreSQL až o 50 % menším a až 3x rychlejším, přibyl příkaz `pg_rewarm` pro rychlé načtení mezipaměti po restartu a dále přibyla i podpora pro Linux Huge Pages – servery s velkou pamětí. Využití GIN indexů je preferovaným typem indexového vyhledávání, kdy vyhledáváním více slov nalezneme první shodu a pomocí indexu odstraníme řádky, které postrádají další slova. Alternativou ke GIN indexům jsou GiST indexy, u kterých mohou nastat falešné shody a je nutné procházet celé řádky z důvodu eliminace neodpovídajících výsledků [26].

2.4.4 Verze 9.5

S verzí 9.5 přišla na svět nová politika zabezpečení tabulky, a to v rámci řádků. Od této doby tedy programátor může označit řádky v tabulce, které uživatel nemá používat, jako neviditelné. V tomto případě programátor využívá nastavení výchozí politiky pro dané tabulky – určí uživatele bez systémových práv, kteří daná data mohou zobrazit v rámci uplatnění dotazu nad danou tabulkou. Další využitelnou funkcí je tzv. grouping set. Jeho příkladem může být jednoduchý select příkaz nad tabulkou zaměstnanců, který vrátí sloupce se jménem, pozicí, pracovním oddělením a pohlavím zaměstnance. Pokud by uživatelem požadovaný výstup byl ten, který bude seskupen pomocí oddělení, pozice a pohlaví, příkaz `group by` by byl tedy rozšířen o výraz *grouping sets* s požadovaným filtrem. Výstupem tedy bude přehled oddělení, pozic, pohlaví a přibude poslední sloupec s celkovým počtem zaměstnanců vyhovujícím danému filtru. Dále může být např. příkaz `insert` rozšířen o klauzuli *on conflict do update/ignore*, která se aplikuje v rámci vkládání duplicitních hodnot a řeší vzniklé problémy [27].

2.4.5 Verze 9.6

Od této verze Postgre přidává podporu pro paralelizování některých operací dotazu, což umožní využití několika, popř. všech jader serveru pro urychlení výsledků dotazu. To zahrnuje paralelní skenování tabulek, agregací a spojení (join). V závislosti na dostupnosti jader distributor slibuje až 32x větší zrychlení.

Funkce full textového vyhledávání PostgreSQL nyní podporuje tzv. phrase vyhledávání. To uživatelům umožní vyhledávat přesné fráze nebo slova s pomocí GIN indexů. V kombinaci s novými funkcemi pro úpravy vyhledávání je tedy jasnou volbou pro hybridní vyhledávání [28].

2.4.6 Memory cache

Bez ohledu na obsah využívá PostgreSQL k ukládání formát stránek o velikosti 8kb. Do cache může ukládat údaje z tabulek, indexy (také ve formátu stránek, stejná oblast paměti jako data z tabulek) a plány na spuštění dotazu, popř. jejich vyhodnocení po spuštění. Toto lze nalézt pomocí dotazu `pg_prepared_statements`. Tyto plány se dají těžko analyzovat, ale důležité jsou pouze při složitějších a hodně rozsáhlých dotazech a zobrazit se dají pouze pro aktuální relaci. PostgreSQL má několik konfiguračních parametrů. Pro ukládání do mezipaměti je nejdůležitější konfigurace `shared_buffers` (ve zdrojovém kódu označováno jako `NBuffers`), která obsahuje všechny sdílené údaje v paměti. Zjednodušeně řečeno je to pole 8 kB bloků, kde každá stránka obsahuje metadata sebe sama, aby bylo jednodušší je rozlišit. Před jakoukoli kontrolou na disku si PostgreSQL nejdříve projde celý buffer, v případě nalezení shody se tak vyhne diskovým operacím. Mechanismy zapojené do ukládání dat do cache jsou řízeny algoritmem Clock Sweep, který je postaven tak, aby zvládal pracovní zátěž OLTP.

PostgreSQL je databázový systém založený na procesech, tzn. že každé spojení má svůj vlastní proces, který je výsledkem procesu root. Když je pro libovolnou stránku vyžadován přístup pomocí typického dotazu SQL, požaduje se přidělení cache. Stránka je z cache uvolněna, když se počet uživatelů rovná nule [29].

2.4.7 Zpracování dotazů

Zpracování dotazu u PostgreSQL probíhá velice podobně jako u předchozích databázových systémů.

Poté, co PostgreSQL server obdrží dotaz z klientské aplikace je jeho text předán analyzátoru. Ten skenuje příslušný dotaz a kontroluje syntaktické chyby. Je-li dotaz syntakticky správný,

analyzátor transformuje text do parsového stromu. Po dokončení analýzy je strom předán plánovači.

Plánovač je zodpovědný za nalezení všech možných plánů pro provedení dotazu. Plán může zahrnovat sekvenční skenování přes celou tabulku, popř. indexové skenování, pokud byly definovány užitečné indexy. Pokud dotaz obsahuje dvě a více tabulek, může plánovač navrhnout řadu různých metod pro připojení k tabulkám. Plány spuštění jsou vyvíjeny z hlediska operátorů dotazů. Každý operátor dotazu transformuje jednu nebo více vstupních sad do sady výsledků. Operátor Seq Scan například transformuje vstupní množinu (fyzickou tabulku) do sady výsledků a filtruje všechny řádky, které nesplňují omezení dotazu. Operátor řazení vytvoří výsledek nastavením přeskupení vstupní sady podle jednoho nebo více třídících klíčů.

Po vygenerování všech možných plánů spuštění optimalizátor vyhledá nejméně nákladný plán. Každému plánu jsou přiděleny odhadované náklady na provedení. Odhady nákladů se měří v jednotkách disku I / O. Každý operátor dotazu má jiný odhad na časové náklady na CPU. Příkladem mohou být náklady na sekvenční skenování celé tabulky. Tyto jsou vypočteny jako počet bloků 8 K v tabulce, včetně některých režijních nákladů na CPU.

Po výběru nejméně nákladného plánu spuštění začíná dotazovač na začátku plánu a požádá nejvyšší operátor, aby vytvořil sadu výsledků. Každý operátor transformuje svoji vstupní sadu na sadu výsledků, které jsou vráceny do klientské aplikace.

Data Output	Explain	Messages	Query History
	QUERY PLAN text		
1	Sort (cost=32.15..32.86 rows=283 width=32)		
2	Sort Key: jmeno		
3	-> Seq Scan on trpaslici (cost=0.00..20.63 rows=283 width=32)		
4	Filter: (narozen > '1982-01-01'::date)		

Obrázek 9 - Exekuční plán PostgreSQL (vlastní zpracování)

Na předchozím obrázku č. 9 je vyobrazen exekuční plán PostgreSQL pro příkaz:

```
select * from trpaslici
where 'narozen' > '1982-01-01'
order by jmeno ASC;
```

Tento plán ukazuje, jakým stylem bude tabulka skenována na základě daného dotazu. Nejkritičtější částí je opět udání nákladnosti spuštění dotazu [30].

2.5 SAP HANA

Poslední představený databázový systém je od společnosti SAP. Z důvodu toho, že nebude zařazen mezi testované bude pouze přiblížen z pohledu memory cache pro porovnání.

2.5.1 Memory cache

Úložná mezipaměť pro ukládání řádků v paměti ukládá transakčně aktivní data, poskytuje vysoce výkonný ukládací repositář pro tzv. horká data (tabulka či oddíly, angl. hot tables) a rozšiřuje možnosti tradičního stránkového ukládání. Pojem hot tables se označuje tabulka, která je paměťově rezidentní a je neustále skenována nebo aktualizována – horká data jsou tedy částí tabulky, většinou malé, které jsou transakčně aktivní. Vyrovnávací paměť IMRS⁹ je rozšířením prostoru pro databázové zařízení přiděleného do databáze a poskytuje další úložiště v paměti pro ukládání často přístupných řádků z databáze.

Ne všechny tabulky se nacházejí v této mezipaměti, převážně to bývá malý počet hotových tabulek schématu OLTP. Server vloží všechna nová data do vyrovnávací paměti, do které přistupuje a zároveň v ní aktualizuje. Pro transakční aplikace je typické, že jakmile server vloží datový záznam, je tento přístupný po krátkou dobu bezprostředně po příkazu insert. Po nějaké době, kdy není údaj zobrazen nebo aktualizován, přechází do stavu pasivních dat. Příležitostně se pasivní data dají znovu aktivovat – např. při účetnictví ve čtvrtletí. V tomto případě jsou znovu přesunuta do mezipaměti. Časem se jinak pasivní data stávají neaktivními a přesouvají se do méně účinného úložného a přístupového mechanismu.

Permanentní (disková) databáze s podporou IMRS je plně trvanlivá a chová se jako tradiční databáze podporující všechny vlastnosti ACID, která poskytuje plnou podporu transakčního zatížení, a přitom pro něj poskytuje lepší výkon. Ačkoli by měla IMRS databáze být v mezipaměti celá, skutečně v ní je jen malá část.

Pro existující aplikace existuje zanedbatelné množství úprav pro převod na IMRS, jako je vytvoření tabulky s povolenou IMRS službou či změna (ALTER) stávající tabulky tak, aby měla povolenou službu IMRS. Zvláštním rozdílem pro IMRS tabulky oproti předchozím databázovým systémům je ten, že v tomto případě není vyžadován primární klíč nebo unikátní index [31].

⁹ IMRS = In-memory row storage

3 METODY TESTOVÁNÍ

Databáze se nemusí vždy zabývat jen jedním typem dat, a proto může v databázových systémech často docházet k velkému množství chyb při implementaci a integraci, což negativně ovlivňuje výkon, spolehlivost a bezpečnost databázového systému. Z toho důvodu je důležité vše vždy otestovat, aby příslušný databázový systém splňoval vlastnosti ACID.

Jednou z nejkritičtějších vrstev z pohledu testů je ta, která přistupuje k samotným datům, protože řídí celý komunikační proces. Testování na této vrstvě se především zabývá kontrolou kvality, jak jí lze co nejlépe dosáhnout a celkovou konzistencí databázového systému. Špatně nebo nijak otestovaný databázový systém by mohl poškodit celou společnost, která ho využívá, např. společnost Google, která se zabývá ukládáním jednotlivých dat, by bez případné kontroly databázového systému a nefunkčnosti některých operací, jako je vkládání, odstraňování a aktualizace dat riskovala havárii celého svého databázového systému.

Veškerá testování probíhají v následujících krocích:

1. příprava prostředí,
2. spuštění testu,
3. kontrola výsledků,
4. ověření platnosti a
5. zpracování zprávy s výsledky.

Při testech se využívají převážně dotazy jazyka SQL, nejčastěji příkazy SELECT. Ostatní příkazy viz. kapitola 1.1.6. Veškeré dotazy jsou uzpůsobeny části, kterou testují.

V případě testování transakcí je důležitým faktorem dodržení vlastností ACID. K tomuto testování se využívají příkazy TCL (kapitola 1.1.6) v kombinaci s kontrolním výpisem SELECT, kdy se provádí kontrola změn v databázi, a tedy správnost provedené transakce.

Dále může být testováno databázové schéma, kdy dochází ke kontrole formální definice způsobu, jakým jsou data v databázi organizována. Nejdříve je tedy důležité určit požadavky, na kterých bude daná databáze pracovat, např. tvorba primárních klíčů musí proběhnout vždy před vložením ostatních dat, kompletní indexace cizích klíčů pro snadné vyhledávání, určení omezení na určité hodnoty, které nemohou být vloženy atp. Toto lze testovat různými způsoby, jako využitím příkazů SQL Query DESC <nazev_tabulky>, využitím příkazů pro ověření názvů jednotlivých polí a jejich hodnot či kontrola s pomocí externích programů, např. Schema Crawler.

Při nastání určité události nad určitou tabulkou jsou často využívány spouštěče (triggers), které probíhají při zpracování vkládacích, odstraňujících a editovacích příkazů. Společná metoda testování funkčnosti probíhá spuštěním SQL dotazu bez jeho využití, zapsáním výsledků a následným využitím samotného triggeru. Obojí je testováno během fáze tzv. Black-box a White-box metod, které jsou vysvětleny v následujících podkapitolách.

Uložené procedury jsou podobné uživatelsky nadefinovaným funkcím. Vyvolávají se příkazem EXECUTE PROCEDURE a jejich výstupem je obvykle sada výsledků. Ty jsou uloženy v databázovém systému a dostupné pro uživatelskou aplikaci. Pro jejich testování se opět využívají metody Black-box a White-box [32].

3.1 White-box

Metoda testování White-box se zabývá především interní strukturou databáze, tedy testováním na úrovni zdrojových kódů. Podrobná specifikace zde zůstává uživateli skryta. Zahrnuje testování databázových triggerů, pohledů, SQL dotazů atp. Dále validuje databázové tabulky, datové modely, databázová schémata, kontroluje pravidla referenční integrity a vybírá výchozí hodnoty tabulky pro kontrolu konzistence databáze.

Základní myšlenkou testování triggerů touto metodou je testovat databázi ještě před vložením samotných dat, při testování uložených procedur se využívá k jejich vyvolání a ověření výsledků.

Hlavní výhodou tohoto testování je zjištění chyb v samotném zdrojovém kódu, na jejichž základě lze odstranit interní chyby z databáze. Nekontroluje však samotné SQL příkazy a je nutná znalost programovacích technik. Z důvodu nekontrolování SQL příkazů vznikla technika založená na této s názvem WHite bOx Database Application Technique (WHODATE), kdy jsou příkazy SQL převedeny do formátu GPL a následně testována jejich sémantika [32].

3.2 Black-box

Testování metodou Black-box zahrnuje test rozhraní a integrace databáze, která zahrnuje mapování dat (včetně metadat), ověření příchozích dat, odchozích dat z dotazů. Dále využívá různé techniky, např. techniku ekvivalenčního dělení, která rozděluje vstupní data na oddíly ekvivalentních dat, ze kterých lze vytvořit zkušební případy, či techniku analýzy hraničních hodnot.

V kombinaci s White-Box metodou je možné sjednotit chování aplikace a databáze; můžeme vkládat, mazat a aktualizovat data způsobem, který vyvolá spuštění triggeru a dále kontrolovat

jeho funkčnost a výstupní data. Při testování procedur se také používá pro kontrolu jejich provedení a zhodnocení dat. Dále se tato metoda využívá např. při simulaci napadnutí celého databázového systému.

Velkou výhodou toho typu testování je její jednoduchost, nezávislost na aktuálním vývoji softwaru – lze ji aplikovat i při nedokončeném softwaru. Tím poskytuje programátorům lepší znalosti pro návrh databázového systému. Nevýhodou oproti tomu je, že není známo kolik z programu již bylo testováno a nelze nalézt všechny chyby [32].

4 TESTOVACÍ PROSTŘEDÍ

Celkový test bude zpracován na běžném notebooku, který přinese testovaným databázovým systémům značné hardwarové omezení.

Popis sestavy

Lenovo B590 – Type 6274

CPU: Intel Core i3-3110M 2,40 GHz

RAM: 4,00 GB

SSD: Samsung 850 EVO 250 GB

OS: Windows 8.1 Professional

K testování byly použity aktuální verze databázových systémů:

- Oracle Database 12c Enterprise Edition
- MS SQL 2016 Enterprise Edition
- MySQL 5.7.20
- PostgreSQL 9.6.6

Nastavení vybraných databázových systémů bylo ponecháno výchozí při instalaci. K úpravám docházelo pouze v případě Oracle Database pro povolení in-memory, která není ve výchozím nastavení povolena, a MySQL pro přenastavení omezeného přístupu k operační paměti, bez kterého by docházelo k problémům se zpracováním řádků s dlouhým textem.

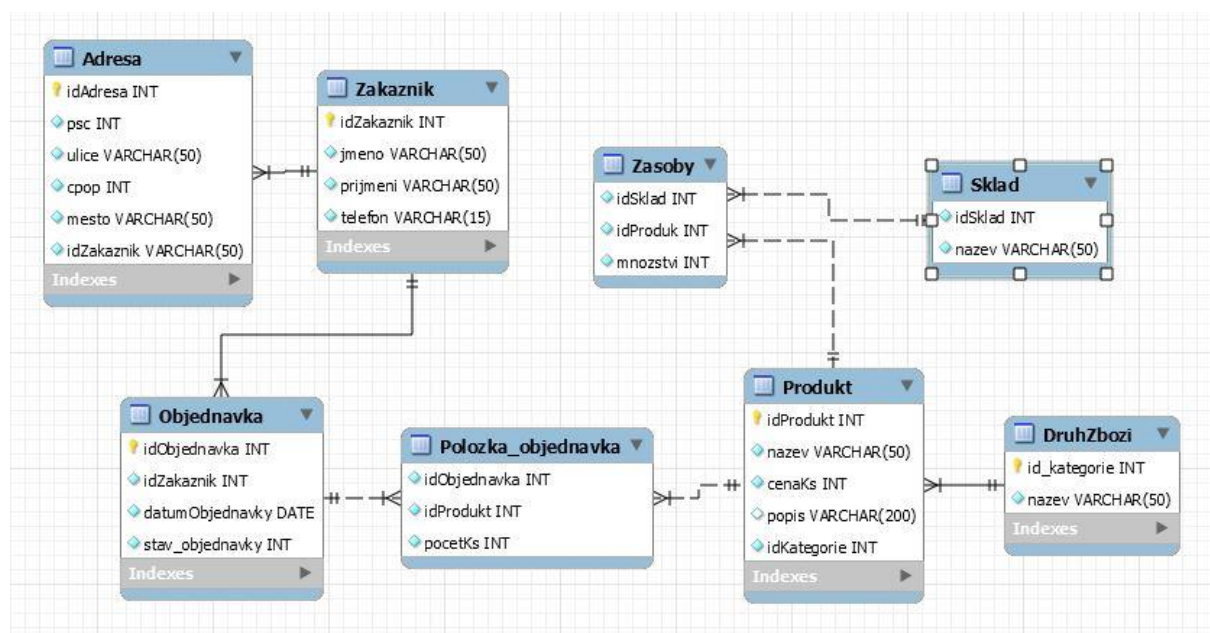
5 PŘÍPRAVA DATABÁZOVÉHO SYSTÉMU

Následující kapitola pojednává o přípravě databázových systémů na testování.

5.1 Databázové schéma

Databáze, na které bude probíhat testování, představuje malou pobočku firmy prodávající zboží. Pobočka si eviduje informace o svých zákaznících, kterým poskytuje přehled zboží, včetně detailních informací o něm, a to i v rámci jeho kategorie. Dále si eviduje informace o naskladněném zboží. V neposlední řadě dovoluje přijímat objednávky a kontrolovat jejich stav během jejich tvorby. Dále umožňuje vystavení faktury nebo přehled o tržbách.

Ukázkové schéma se skládá z 8 tabulek a je zobrazeno na obrázku č. 10. Toto schéma bylo vytvořeno s pomocí editoru databázových modelů MySQL Workbench 6.3 CE, a proto obsahuje proměnné, které jsou poskytovány MySQL databázovým systémem. Změny v rámci SQL jsou nastíněny v kapitole 5.3.



Obrázek 10 - Databázové schéma (vlastní zpracování)

5.2 Naplnění daty

U všech databázových systémů byly naplněny tabulky Adresa, DruhZbozi, Produkt, Sklad, Zakaznik a Zasoby stejnými, náhodně vygenerovanými daty, a to před začátkem testu. Tabulky Zakaznik a Adresa jsou naplněny 1500 záznamy, DruhZbozi a Sklad 10 záznamy a tabulky Produkt a Zasoby 2000 řádky. Testovací data jsou přílohou této bakalářské práce a byla vygenerována pomocí jednoduchého generátoru v jazyce Java.

5.3 Změny v SQL

Mezi jednotlivými databázovými systémy je značná odlišnost při zpracování SQL dotazů, ale je nutné počítat i s proprietárními vlastnostmi jazyka SQL pro jednotlivé databázové systémy. První, které bylo nutné během zpracování této práce řešit, jsou datové typy. Oracle Database jako jediný obsahuje číselné hodnoty typu NUMBER, zatímco ostatní databázové systémy typ INTEGER. Dalším zajímavým rozdílem je ukládání hodnot času – DATETIME. Pro ukládání těchto hodnot byl největší problém s MySQL. Zatímco ostatní databázové systémy dodržují SQL standard s drobnými úpravami, u MySQL se ukládá datum i čas bez zlomkových sekund. V tomto případě dochází mnohdy k chybnému ukládání nulových hodnot. Asi posledním větším problémem bylo nalezení vhodného ukládání typu BOOLEAN, jelikož jediným databázovým systémem, který tento typ podporuje je PostgreSQL, dále SQL Server obsahuje proměnnou BIT, která může nabývat hodnot 0,1 či NULL, což by bylo také vhodné řešení. Dokumentace Oracle doporučila ukládání hodnot 0,1 do datového typu NUMBER, který ovšem nepodporují další databázové systémy. Vše bylo vyřešeno jednotným ukládáním do typu INTEGER. Oracle Database tak v rámci svého vnitřního procesu provádí převod INTEGER na NUMBER.

Z důvodu využití jednodušších SQL příkazů se nevyskytly závažnější potíže v rámci syntaxe. To by ovšem mohlo nastat např. při výpisu prvních 3 řádků, které nejlépe vyhovují daným dotazům. U Oracle Database se toto řeší přidáním klíčového slovíčka ROWNUM do klauzule WHERE, u MySQL přidáním klíčového LIMIT za položku WHERE, kdežto v případě MS SQL se toto řeší klíčovým slovem TOP následovaným hned po SELECT. V případě úprav dotazů tedy došlo pouze v případě dotazu č. 11 (viz. následující kapitola č. 6), kdy MS SQL nepodporuje stejné způsoby zřetězení.

6 PŘÍPRAVA DOTAZŮ

Následující kapitola obsahuje příklady dotazů, které budou obsaženy v zátěžovém testu. Zadané číselné hodnoty v dotazech jsou v tomto případě použity pro demonstrační účely. Dotazy budou probíhat ve většině případů v rámci transakcí, aby nedocházelo k narušení databáze. Touto formou se bude dále zjišťovat, jak si daný server poradí v případě deadlocku, kdy dochází k tzv. uvážnutí databázového systému z důsledku požadování stejných dat dvěma transakcemi najednou.

1. Vytvoření objednávky

```
insert into objednavka values (?, ?, 'DD-MM-YYYY', 0);  
insert into polozkaObjednavka values (?, ?, ?);
```

2. Odpočet zboží na skladě

```
UPDATE zasoby SET mnozstvi = (mnozstvi - pocetZbozi) WHERE id_produkt = 1;
```

3. Odeslání objednávky

```
update objednavka set stavObjednavky = 1  
where idObjednavka = 1;
```

4. Výpis objednávky

```
SELECT objednavka.idObjednavka, polozkaObjednavka.idProdukt,  
polozkaObjednavka.pocetks, produkt.cenaKS, adresa.ulice, adresa.mesto,  
adresa.psc, zakaznik.jmeno, zakaznik.prijmeni FROM objednavka  
JOIN polozkaObjednavka ON objednavka.idObjednavka =  
polozkaObjednavka.idObjednavka  
JOIN zakaznik ON objednavka.idZakaznik = zakaznik.idZakaznik  
JOIN adresa ON zakaznik.idZakaznik = adresa.idZakaznik  
JOIN produkt ON polozkaObjednavka.idProdukt = produkt.idProdukt  
WHERE objednavka.idZakaznik = '2' AND objednavka.STAVOBJEDNAVKY = '1';
```

5. Výpis nabídky zboží

```
SELECT prod.idProdukt, prod.nazevProdukt, prod.cenaKs, zas.idSklad,  
zas.Mnozstvi, dr.Nazev from Produkt prod  
join zasoby zas on prod.IDPRODUKT = zas.IDPRODUKT  
join druhZbozi dr on prod.IDKATEGORIE = dr.IDKATEGORIE;
```

6. Doplnění skladu zboží pod 50 ks

```
UPDATE zasoby SET mnozstvi = (50 - mnozstvi + mnozstvi)  
WHERE MNOZSTVI < 50;
```

7. Nalezení nejvíce obsazeného skladu

```
SELECT sk.idSklad, sk.nazev FROM sklad sk  
join zasoby zas on sk.IDSKLAD = zas.idSklad  
where zas.mnozstvi = (select MAX(mnozstvi) from zasoby);
```

8. Vypočítání celkové tržby

```
select SUM(polObj.pocetKs * prod.cenaKS) AS VYDELEK
from polozkaObjednavka polObj
join produkt prod on prod.idProdukt = polObj.idProdukt
join objednavka obj on polObj.idObjednavka = obj.idObjednavka
where obj.stavObjednavky = 1;
```

9. Průměrná cena všech objednávek

```
select AVG(polObj.pocetKs * prod.cenaKS) AS PRUMER from polozkaObjednavka
polObj
join produkt prod on prod.idProdukt = polObj.idProdukt
join objednavka obj on polObj.idObjednavka = obj.idObjednavka
where polObj.idObjednavka = obj.idObjednavka;
```

10. Výpis zákazníků i s adresou a objednávkami

```
select zak.jmeno ||' '|| zak.prijmeni as ZAKAZNIK,
       adr.ulice ||' '|| adr.cpop ||', '|| adr.psc ||', '|| adr.mesto AS
adresa,
from adresa adr
join zakaznik zak on zak.idZakaznik = adr.IDZAKAZNIK
order by zak.jmeno ASC;
```

pro MS SQL byla provedena úprava zřetězení v první části dotazu

```
SELECT (zak.jmeno + ' ' + zak.prijmeni) AS Zakaznik,
       (adr.ulice + ' ' + adr.cpop + ', ' + adr.psc + ', ' + adr.mesto)
AS Adresa
```


7 ZÁTĚŽOVÝ TEST

Cílem zátěžového testu bude každý vybraný databázový server využít s větší zátěží. Celý proběhne v aplikaci¹⁰ napsané v jazyce Java a bude obsahovat transakční bloky dotazů, ve kterých budou probíhat operace s využitím DML příkazů, taktéž s využitím ORDER BY, protože řadící operace zaměstnávají server a velkou část HW nejvíce. Pro běh celého testu bylo použito 10 000 spouštěných operací, které byly rozpočítány mezi testovací části na základě jejich procentuálního ohodnocení. Tato procenta jsou pro všechny testované databázové systémy stejná. Po celou dobu testu se bude zaznamenávat počet všech uskutečněných dotazů, ze kterého bude za pomoci celkového uběhlého času vypočítán průměr dotazů/s – číslo, které nám určí propustnost dané databáze. Dále bylo na začátku testu sledováno, kolik času danému databázovému systému zabere import dat do tabulek.

K využití vlastní aplikace bylo přistoupeno z několika důvodů. Databázové systémy je možno sledovat s využitím konzolí vybraných databázových serverů. To by ovšem nezaručilo všem databázovým systémům absolutně stejné podmínky, dále by k tomuto způsobu bylo potřeba využívat dalších sledovacích nástrojů. Z prozkoumaného množství dostupného testovacího SW žádný nevyhovoval požadovaným kritériím, nebo s jeho využitím docházelo k problémům alespoň u jednoho testovaného databázového systému.

7.1 Popis testovacích částí

Jak bylo již naznačeno v úvodu této kapitoly, test bude rozložen do různých částí. Každá jednotlivá část bude obsahovat soubor několika dotazů a bude opakována na základě náhodného výběru vypočítaného na základě procentuálního ohodnocení daného bloku. Procentuální ohodnocení bylo rozděleno na základě vyhodnocení teoretické části – nejvyšší zastoupení mají bloky zpracovávající dotazy s nejčastějšími změnami, kdy jsou všechny databázové systémy nuceny provést daný dotaz znovu a nedochází zde tedy tak často k opakování stejných dotazů uložených v cache daného databázového systému.

1) Objednávka (15%)

Tento blok bude obsahovat největší kombinaci různých dotazů. Na začátku bude obsahovat příkazy s vytvořením záznamů v tabulkách Objednávka a PoložkaObjednávka. Z důvodu zablokování počtu zboží pro danou objednávku je nutné provést odpočet zboží na skladě pro dané produkty. Dále bude daná objednávka vyexpedována, tudíž dojde ke změně jejího stavu.

¹⁰ Aplikace je popsána v samostatném dokumentu, který je umístěn v příložených souborech k této práci

Poslední částí tohoto bloku bude vystavení faktury – výpis objednávky s jejími položkami a zákazníkem, který ji vytvořil (kapitola 6, dotazy 1–4).

2) Detailní výpis zboží (30%)

V tomto bloku nalezneme dotaz, který nám poskytne seznam veškerého dostupného zboží, včetně názvu, skladu i kategorie, které náleží. Z důvodu toho, že tento dotaz může sloužit i ke kontrole stavu naskladněného zboží, bude prováděn společně s dotazem, který naplní naskladněné množství u položek, které budou mít méně, než 50 kusů (kapitola 6, dotazy č. 5 a 6).

3) Nalezení skladu s nejvyšším počtem naskladněného zboží (10%)

V rámci využívání různých skladů pro uložení zboží je tomuto bloku věnován dotaz pro určení skladu, který je nejvíce zatížen z pohledu součtu naskladněných položek (kapitola 6, dotaz č. 7).

4) Řazený přehled zboží na základě atributu (30%)

V tomto případě proběhne výpis nabízeného zboží, kdy bude aplikace vybírat postupně řazení tohoto seznamu na základě parametrů názvu produktu, ceny či kategorie. Pro zrealizování tohoto bloku byl rozšířen dotaz č. 5 z kapitoly 6 o jeden řádek s přidáním daného parametru (`order by pr.nazev ASC`).

5) Výpočet tržeb (1%)

Obsahem tohoto bloku je výpočet celkové utržené sumy za všechny objednávky, které byly vyexpedovány, tzn. sloupec se stavem objednávky musí nabývat hodnoty 1 (kapitola 6, dotaz č. 8).

6) Průměrná cena objednávky (1%)

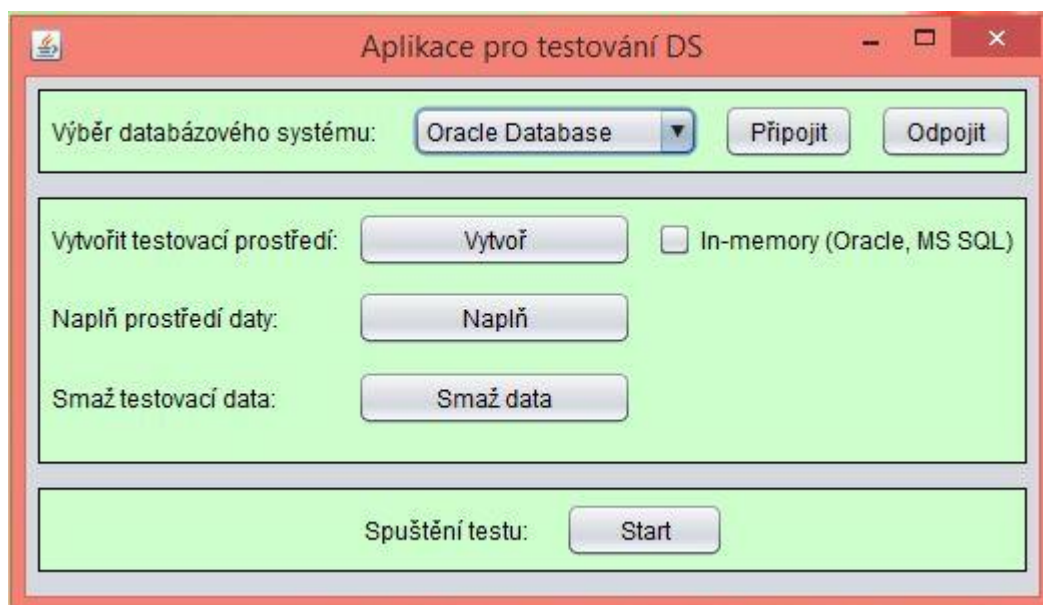
Úkolem tohoto bloku je vypočítání průměrné ceny objednávky z celkového počtu vyexpedovaných objednávek (kapitola 6, dotaz č. 9).

7) Výpis přehledu zákazníků (8%)

Poslední blok se bude zabývat detailním výpisem všech uživatelů, tedy včetně jejich adres (kapitola 6, dotaz 10).

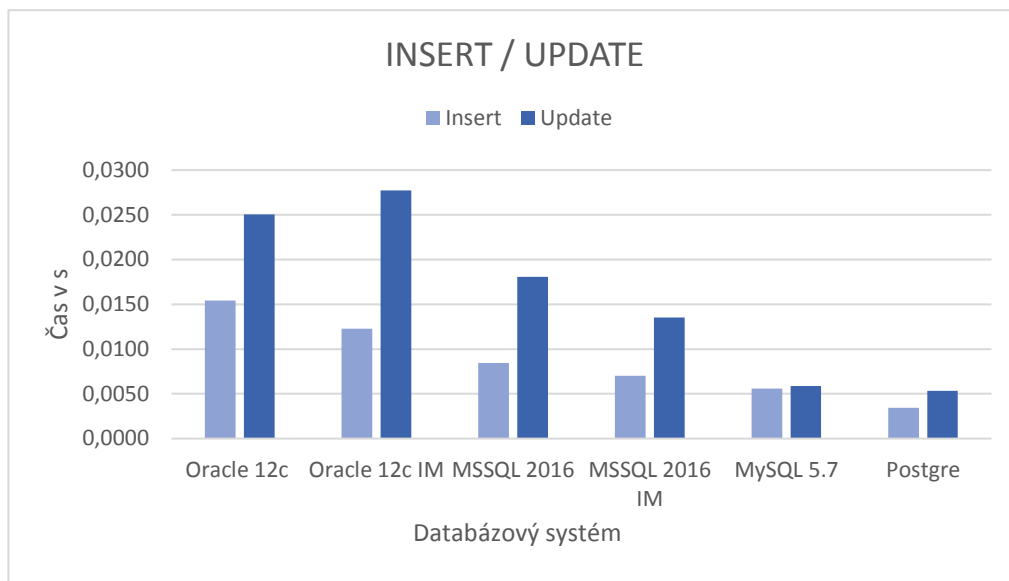
8 VYHODNOCENÍ

Jak již bylo popsáno v předchozích kapitolách popisující testování vybraných databázových systémů je hlavní test rozdělen do bloků vykonávajících příslušné dotazy, kdy součástí těchto bloků je i sledování časů potřebných k jejich vykonání. Celý test proběhl v již zmíněné aplikaci vytvořené přímo za účelem monitorování zpracování daných operací (obrázek č. 11).



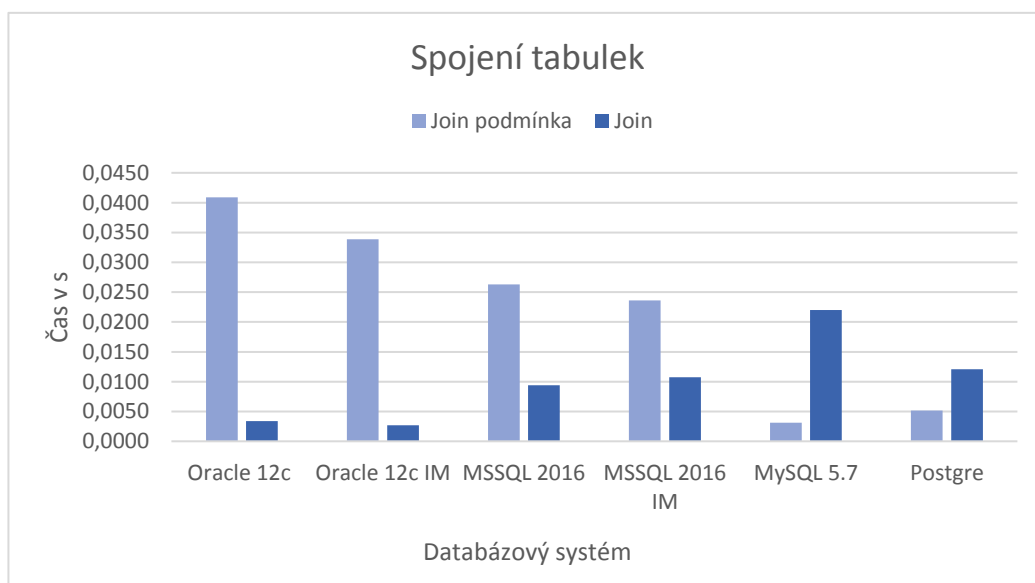
Obrázek 11 - Aplikace (vlastní zpracování)

První testovací blok obsluhuje vytváření objednávky a úpravu zásob na skladech. Na grafu č. 1 je vyobrazeno porovnání časů potřebných pro příkazy INSERT a UPDATE (kapitola 7.1, část 1). V tomto případě se jednalo o vkládání či úpravu jednoho řádku v tabulkách. Při pohledu na výsledek lze říci, že v případě využití memory cache u Oracle Database a MS SQL je vkládání řádků rychleji zpracováno než bez jejího využití. Dále je patrné rychlejší zpracování vkládání řádku u MS SQL než Oracle Database z důvodu využití optimalizovaných tabulek (kapitola 2.2.6) a rychlejšího přístupu k datům než u tabulek uložených na disku. Nejúspěšnějším databázovým systémem byl však jak v případě jednoduchých INSERT tak UPDATE operací nejúspěšnější PostgreSQL.



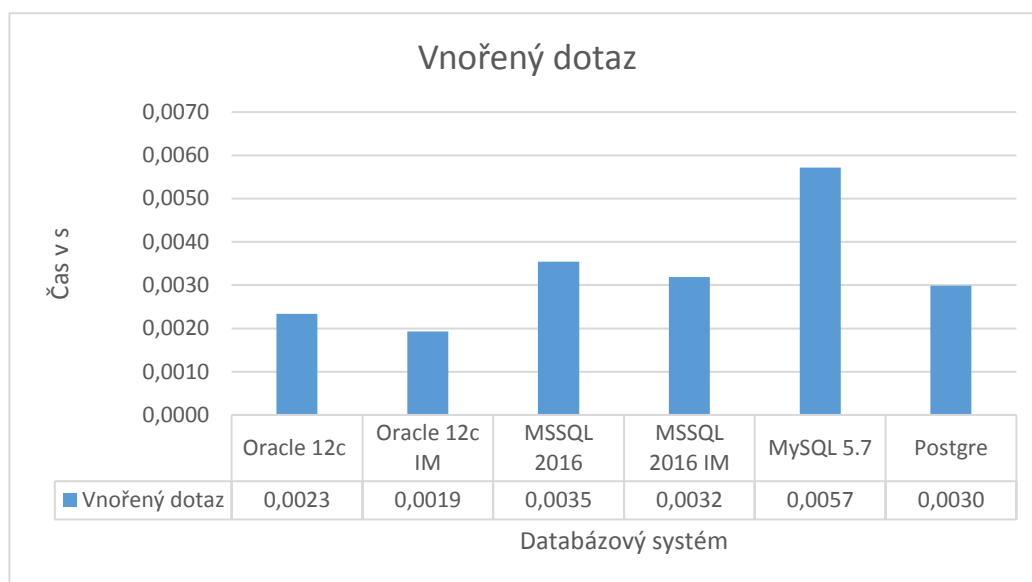
Graf 1 - Insert/Update (vlastní zpracování)

Následující bloky testu se zabývaly příkazy typu SELECT. První blok tohoto typu proběhl v kombinaci s připojováním potřebných tabulek potřebných k vykonání tohoto dotazu. V tomto případě docházelo jak k jednoduchému připojení tabulek na základě primárních a cizích klíčů daných tabulek, tak i ke spojení na základě jisté podmínky, které musel daný dotaz vyhovovat. Z grafu č. 2 je patrné, že databázové systémy Oracle Database a MS SQL měly větší problém se zpracováním JOIN s využitím podmínky bez výraznějšího rozdílu s použitím memory cache. Pro vyhodnocení této části byly použity dotazy bloků 1 a 2 z kapitoly 7.1. V případě JOIN bez využití podmínky pro vyhodnocení dotazu svou úspěšností značně převyšuje Oracle Database.



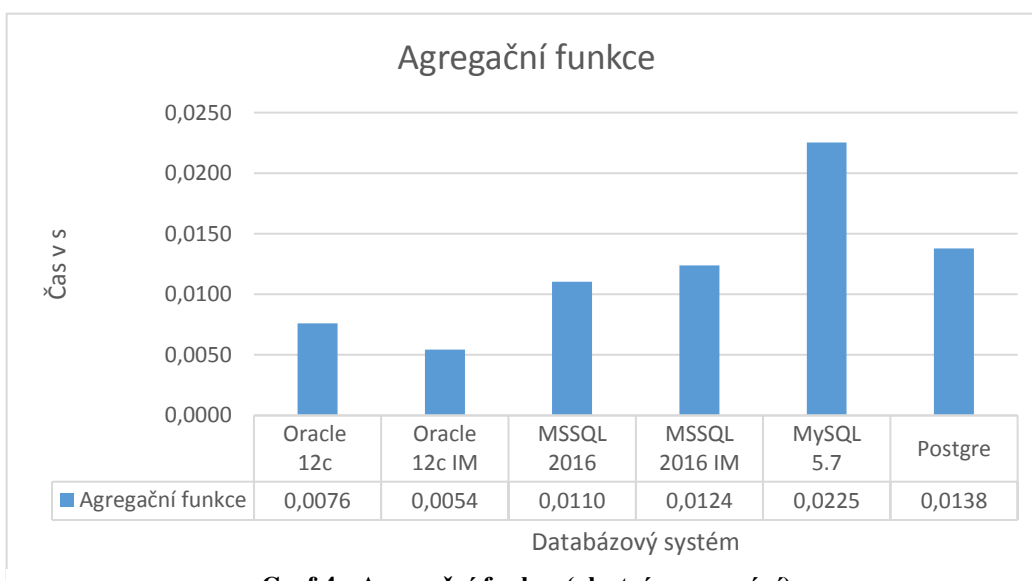
Graf 2 - Spojení tabulek (vlastní zpracování)

Další sledovanou částí při zpracování dotazů bylo též využití vnořeného SELECT dotazu (kapitola 7.1, část 3), kde byl vnořený dotaz využit spolu s agregační funkcí SUM. Tato část opět potvrdila předpoklad nejrychlejšího zpracování u databázového systému Oracle Database s ohledem na využití memory cache. Tohoto výsledku bylo dosaženo díky využití in-memory sloupce, popsaného v kapitole 2.1.6. Výsledek tohoto bloku je vyobrazen v grafu č. 3.



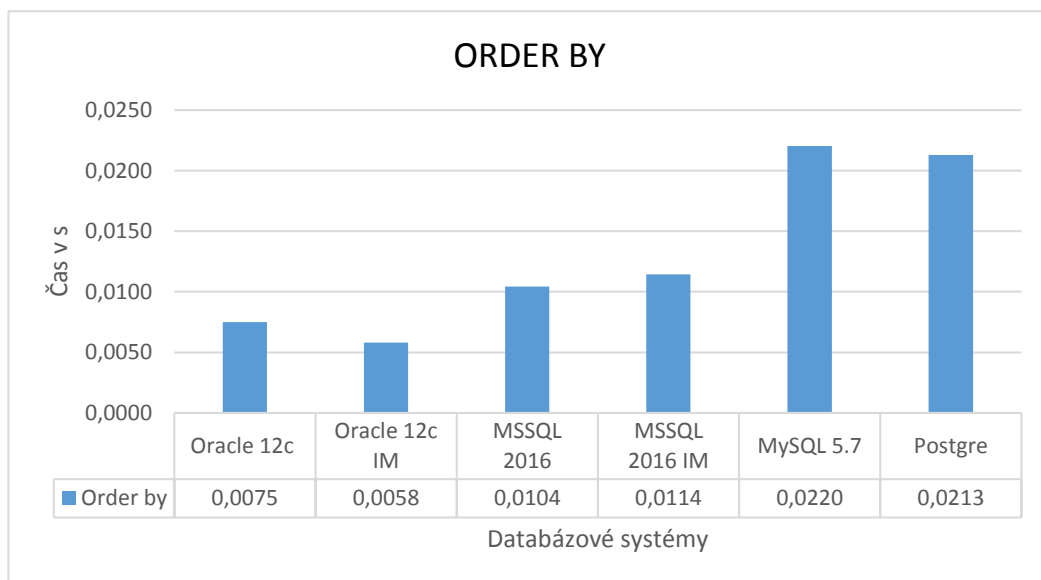
Graf 3 - Vnořený dotaz (vlastní zpracování)

V předchozím testovaném bloku byly zmíněny agregační funkce, které byly také testovány i mimo vnořený dotaz v samostatném SELECTu (kapitola 6, dotazy č. 8 a 9). I přes využití časově náročnější funkce se na prvním místě opět umístil Oracle Database a jeho rychlé zpracování dotazů typu SELECT s využitím memory cache. Výsledky tohoto bloku jsou demonstrovány na grafu č. 4.



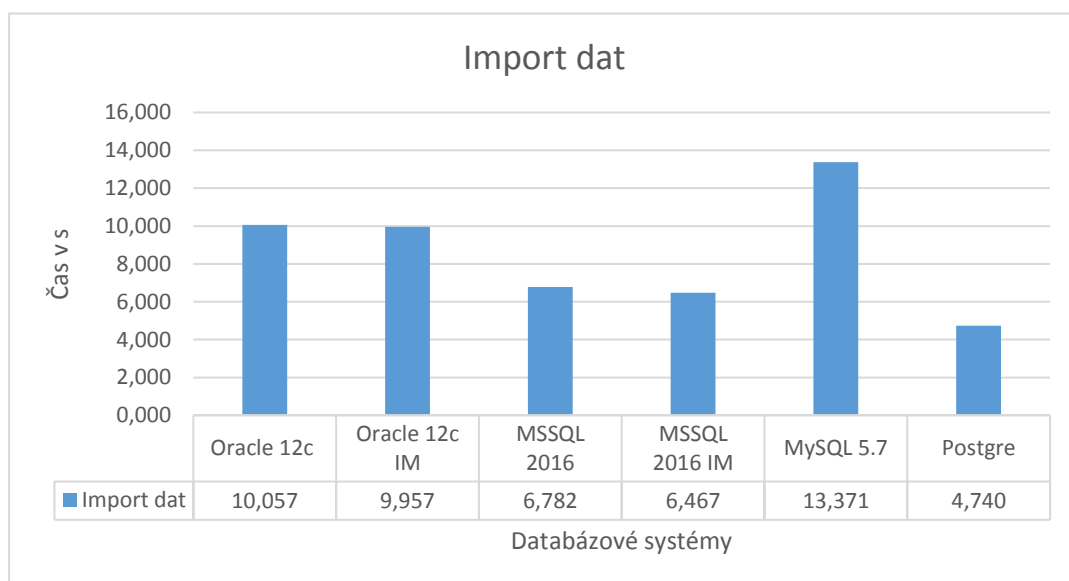
Graf 4 - Agregační funkce (vlastní zpracování)

Během popisu zátěžového testu v kapitole č. 7 došlo ke zmínění sledování časů potřebných k využití řadicí funkce ORDER BY, na které se dá dobře sledovat vytížení daného databázového systému. Jak je již z grafu č. 5 patrné, i v tomto případě se na prvním místě umístil Oracle Database s využitím in-memory sloupce.



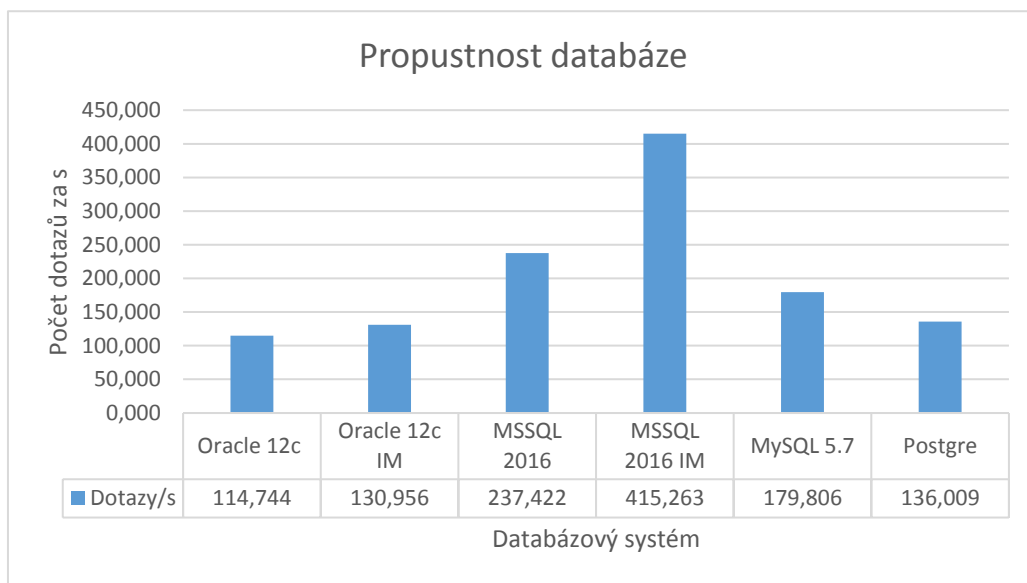
Graf 5 - Order by (vlastní zpracování)

V úvodní části tohoto vyhodnocení proběhlo k porovnání databázových systémů při jednoduchých INSERT a UPDATE operacích. Z důvodu zkrácení časového intervalu u monitorování jednotlivých bloků dotazů proběhlo naplnění jednotlivých databází odděleně. Na grafu č. 6 je vyobrazeno porovnání časů právě pro hromadné provádění INSERT operací u jednotlivých databázových systémů.



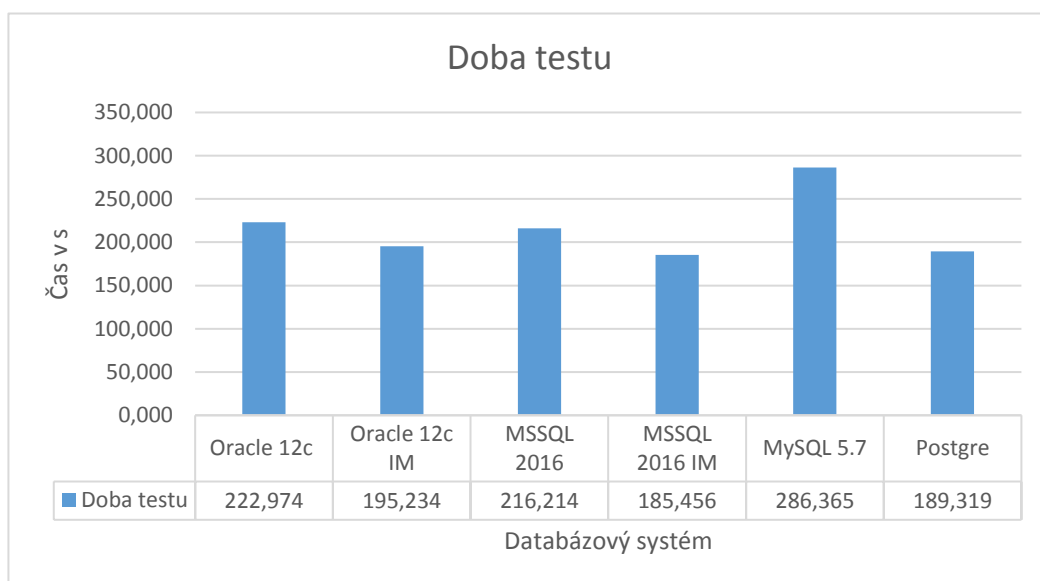
Graf 6 - Import dat (vlastní zpracování)

Při určení propustnosti databáze se využívá pro porovnání jednotlivých databázových systémů průměrného zpracování dotazů za sekundu. K získání tohoto čísla bylo nutné měřit celkový počet vykonaných dotazů v průběhu testu a vydělit ho celkovou dobou, po kterou daný test probíhal. Graf č. 7 obsahuje porovnání databázových systémů z hlediska počtu průměrně zpracovaných dotazů. V tomto případě se s velkým odstupem od ostatních databázových systémů na prvním místě nachází MS SQL s využitím memory cache. Na tomto příkladu se dá nejlépe demonstrovat rychlá komunikace s optimalizovanými tabulkami.



Graf 7 - Propustnost databáze (vlastní zpracování)

Poslední částí vyhodnocení je porovnání celkového času vykonání testu na grafu č. 8. V tomto případě žádný z databázových systémů až na MySQL nevybočuje z řady.



Graf 8 - Doba testu (vlastní zpracování)

ZÁVĚR

Hlavní cíle této práce spočívaly v představení vybraných databázových systémů (DBS), teoretickým popisem memory cache u každého z DBS, následované monitoringem, analýzou, zpracováním a komparací výsledků.

První dva vytyčené cíle jsou soustředěny na teoretickou část práce. Při zpracování této části bylo občas obtížné nalézt potřebné informace v jednotlivých dokumentacích databázových systémů. Představení jednotlivých databázových systémů má totožné či podobné členění, což čtenáři zjednodušuje přehlednost a snadnou orientaci v teoretické části. Z důvodu uspokojivého rozsahu teoretické části a následně úspěšnému pokrytí stejných kapitol o jednotlivých databázových systémech, lze tuto část označit jako vyhovující vytyčenému cíli.

Pro zpracování praktické části bylo, po delším průzkumu dostupného testovacího SW či náročnosti práce s jednotlivými konzolami a přidruženým SW pro monitorování, přistoupeno k vytvoření vlastní aplikace v jazyce Java. Výsledný projekt aplikace splňuje nezbytně nutné požadavky na monitorování jednotlivých databázových systémů. Je schopen spravovat připojení k vybranému databázovému systému vybráním z nabídky, vytvořit a následně i smazat testovací prostředí. V rámci instance Oracle Database se jednalo o vytvoření testovacího databázového schématu, u ostatních databázových systémů o vytvoření vlastní databáze a teprve následně došlo k implementaci daného schématu. Při spuštění testu umí aplikace po celou dobu trvání testu vypočítávat a průměrovat časy potřebné k vyhodnocení zadaného úkolu. Výstupy z této aplikace jsou ukládány do textového souboru s příponou txt.

Na základě uskutečněného testu a porovnání výsledků jednotlivých částí lze potvrdit určitá tvrzení z teoretické části. Mezi první jistě patří úspěšnost Oracle Database a jeho in-memory sloupce, který značně urychluje SELECT dotazy při vyhledávání v tabulkách. MySQL a její využití memory cache v rámci ukládání často používaných dotazů se nejlépe potvrdila v části spojování tabulek. Slibované zvýšení výkonu databázového systému při využití optimalizovaných tabulek u MS SQL bylo nejlépe patrné u výsledného čísla propustnosti databáze. V tomto směru byl MS SQL bezkonkurenčně nejlepší, kdy své konkurenty převyšoval až dvojnásobně. Důležité je také podotknout, že PostgreSQL, jakožto nejméně rozšířený z vybraných databázových zástupců, se může hravě porovnávat se svými kolegy.

Budoucí možné rozšíření dané aplikace by se dalo uplatnit ve směru využití více vláken zpracovávajících zátěžový test. Tento faktor by mohl ovlivnit výsledek propustnosti databázového systému.

ZDROJE

- [1] *Tutorials Point: Simply Easy Learning: DBMS - Overview* [online]. Hyderabad: Tutorials Point, 2017 [cit. 2017-08-22]. Dostupné z: https://www.tutorialspoint.com/dbms/dbms_overview.html
- [2] LACKO, Ľuboslav. *1001 tipů a triků pro SQL*. Brno: Computer Press, 2011. ISBN 978-80-251-3010-0.
- [3] *ACID properties* [online]. Birmingham: Wrox Press, 1998 [cit. 2017-11-13]. Dostupné z: <https://msdn.microsoft.com/en-us/library/aa480356.aspx>
- [4] *Tutorials Point: Simply Easy Learning: DBMS – Data Models* [online]. Hyderabad: Tutorials Point, 2017 [cit. 2017-08-22]. Dostupné z: https://www.tutorialspoint.com/dbms/dbms_data_models.html
- [5] ŠIMŮNEK, Milan. *SQL: kompletní kapesní průvodce*. Praha: Grada, 1999. ISBN 80-716-9692-7.
- [6] *Application Architecture. Oracle Help Center: Database Concepts* [online]. California: Oracle, 2017 [cit. 2017-08-22]. Dostupné z: https://docs.oracle.com/cd/B28359_01/server.111/b28318/dist_pro.htm#CNCPT1268
- [7] *Oracle Help Center: Database Concepts: Oracle Database 10g New Features* [online]. California: Oracle, 2017 [cit. 2017-08-22]. Dostupné z: https://docs.oracle.com/cd/B19306_01/server.102/b14214/chapter1.htm#NEWFTCH1
- [8] *Oracle Help Center: Database Concepts: Oracle Database 11g New Features*. [online]. California: Oracle, 2017 [cit. 2017-08-22]. Dostupné z: https://docs.oracle.com/cd/B28359_01/server.111/b28279/chapter1.htm#NEWFTCH1
- [9] *Oracle Help Center: Database Concepts: Oracle Database 12c New Features* [online]. California: Oracle, 2017 [cit. 2017-08-22]. Dostupné z: <https://docs.oracle.com/database/121/NEWFT/chapter12102.htm#NEWFT003>
- [10] *Oracle In-Memory Database Cache Overview* [online]. California: Oracle, 2017 [cit. 2017-11-13]. Dostupné z: <http://www.oracle.com/technetwork/products/timesten/overview/timesten-imdb-cache-101293.html>
- [11] *Oracle Technology Network: Oracle Database In-Memory* [online]. California: Oracle, 2017 [cit. 2017-11-20]. Dostupné z: <http://www.oracle.com/technetwork/database/in-memory/overview/index.html>
- [12] *Oracle Help Center: Database Concepts: SQL Processing* [online]. California: Oracle, 2017 [cit. 2017-08-22]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm#TGSQL175

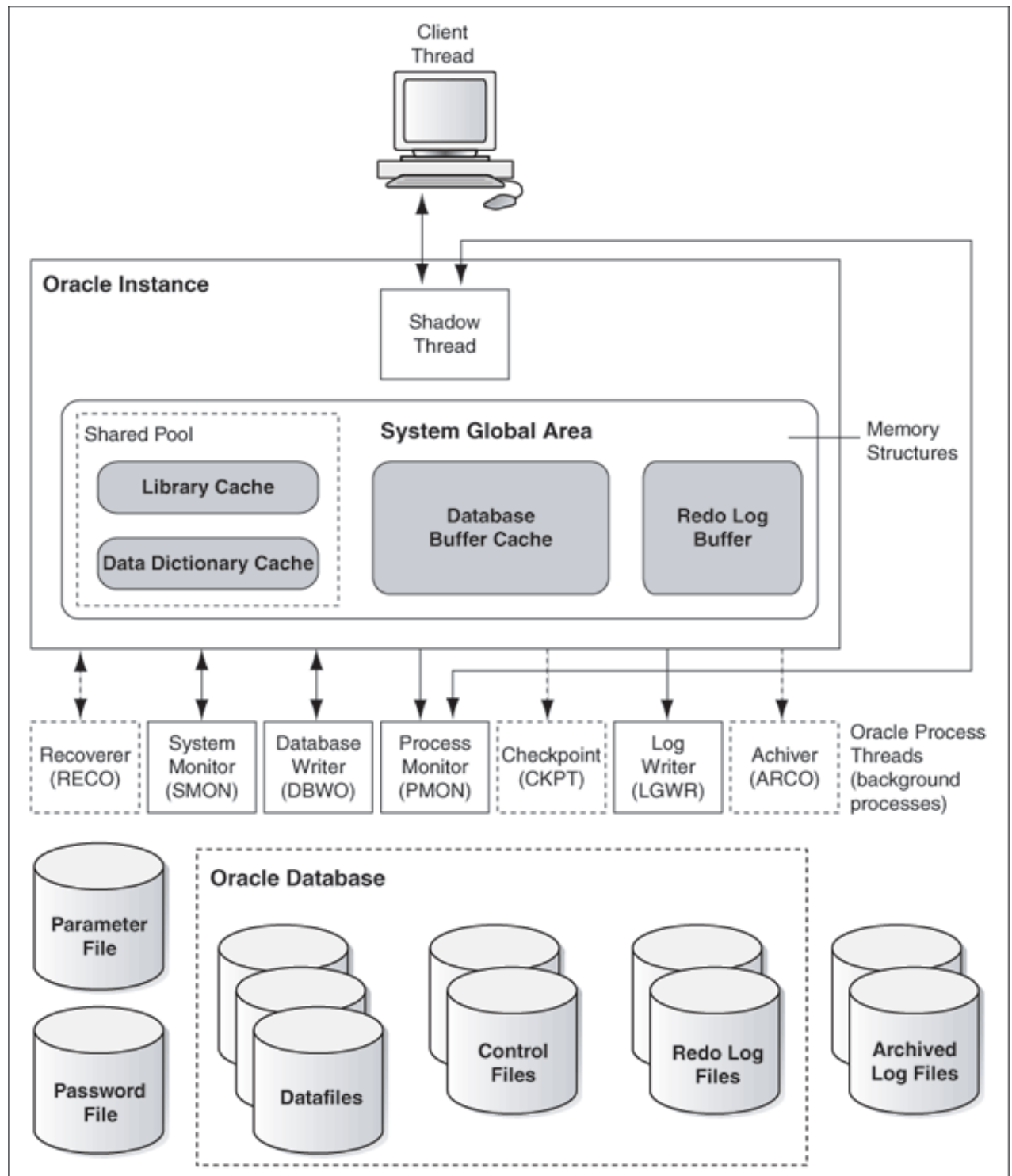
- [13] *SQL Authority: with Pinal Dave* [online]. Bengaluru: Pinal Dave, 2012 [cit. 2017-11-21].
Dostupné z: <https://blog.sqlauthority.com/2012/08/30/sql-server-beginning-sql-server-architecture-terminology/>
- [14] *Microsoft Tech Net: What's New in SQL Server 2012* [online]. Redmont: Microsoft, 2012 [cit. 2017-11-21]. Dostupné z: [https://technet.microsoft.com/en-s/library/bb500435\(v=sql.110\).aspx](https://technet.microsoft.com/en-s/library/bb500435(v=sql.110).aspx)
- [15] *Microsoft Developer Network: What's New in SQL Server 2014* [online]. Redmont: Microsoft, 2014 [cit. 2017-11-21]. Dostupné z: [https://msdn.microsoft.com/library/bb500435\(v=sql.120\).aspx](https://msdn.microsoft.com/library/bb500435(v=sql.120).aspx)
- [16] *Microsoft Docs: What's New in SQL Server 2016* [online]. Redmont: Microsoft, 2016 [cit. 2017-11-21]. Dostupné z: <https://docs.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2016>
- [17] *Microsoft Docs: SQL Server In-Memory OLTP Internals for SQL Server 2016* [online]. Redmont: Microsoft, 2016 [cit. 2017-11-22]. Dostupné z: <https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/sql-server-in-memory-oltp-internals-for-sql-server-2016>
- [18] *Microsoft Docs: Query Processing Architecture Guide* [online]. Redmont: Microsoft, 2016 [cit. 2017-11-22]. Dostupné z: <https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide>
- [19] *MySQL Server: InnoDB Architecture* [online]. Redmont: Oracle, 2017 [cit. 2017-11-21]. Dostupné z: <https://dev.mysql.com/doc/refman/5.7/en/innodb-concepts.html>
- [20] *MySQL Documentantion: What's new in MySQL 5.5* [online]. Redmont: Oracle, 2017 [cit. 2017-11-21]. Dostupné z: <https://dev.mysql.com/doc/refman/5.5/en/mysql-nutshell.html>
- [21] *MySQL Documentantion: What's new in MySQL 5.6* [online]. Redmont: Oracle, 2017 [cit. 2017-11-21]. Dostupné z: <https://dev.mysql.com/doc/refman/5.6/en/mysql-nutshell.html>
- [22] *MySQL Documentantion: What's new in MySQL 5.7* [online]. Redmont: Oracle, 2017 [cit. 2017-11-21]. Dostupné z: <https://dev.mysql.com/doc/refman/5.7/en/mysql-nutshell.html>
- [23] *MySQL Documentantion: Caching queries* [online]. Redmont: Oracle, 2017 [cit. 2017-11-21]. Dostupné z: <https://dev.mysql.com/doc/apis-php/en/apis-php-mysqldn-qc.quickstart.caching.html>
- [24] *Baron Schwartz's Blog: How MySQL really executes a query* [online]. Virginia: Baron Schwartz, 2009 [cit. 2017-11-22]. Dostupné z: <https://www.xaprb.com/blog/2009/04/01/how-mysql-really-executes-a-query/>
- [25] *PostgreSQL Documentation: Architectural Fundamentals* [online]. California: The PostgreSQL Global Development Group, 2009 [cit. 2017-11-22]. Dostupné z: <https://www.postgresql.org/docs/9.6/static/tutorial-arch.html>

- [26] *PostgreSQL Wiki: What's new in PostgreSQL 9.4* [online]. California: The PostgreSQL Web Team, 2016 [cit. 2017-11-22]. Dostupné z: https://wiki.postgresql.org/wiki/What%27s_new_in_PostgreSQL_9.4
- [27] *PostgreSQL Wiki: What's new in PostgreSQL 9.5* [online]. California: The PostgreSQL Web Team, 2016 [cit. 2017-11-22]. Dostupné z: https://wiki.postgresql.org/wiki/What's_new_in_PostgreSQL_9.5
- [28] *PostgreSQL Wiki: NewIn96* [online]. California: The PostgreSQL Web Team, 2016 [cit. 2017-11-22]. Dostupné z: https://wiki.postgresql.org/wiki/NewIn96#What.27s_New_in_PostgreSQL_9.6
- [29] *Understanding caching in Postgres - An in-depth guide* [online]. Chennai: Madusudanan B.N, 2016 [cit. 2017-11-29]. Dostupné z: <https://madusudanan.com/blog/understanding-postgres-caching-in-depth/>
- [30] *Etutorials.org: Understanding How PostgreSQL Executes a Query* [online]. eTutorials.org [cit. 2017-11-22]. Dostupné z: <http://etutorials.org/SQL/Postgresql/Part+I+General+PostgreSQL+Use/Chapter+4.+Performance/Understanding+How+PostgreSQL+Executes+a+Query/>
- [31] *SAP Help Portal: In-Memory Row Storage* [online]. Walldorf: SAP [cit. 2017-11-22]. Dostupné z: <https://help.sap.com/viewer/a1237e466dba417da6f0e5504cf9fb83/16.0.3.2/en-US/4621155144774163837984cbe3fe0656.html>
- [32] TUYA, Javier, M. José SUÁREZ-CABAL a Claudio DE LA RIVA. A practical guide to SQL white-box testing. *ACM SIGPLAN Notices* [online]. 2006, **41**(4), 36- [cit. 2017-11-22]. DOI: 10.1145/1147214.1147221. ISSN 03621340. Dostupné z: <http://portal.acm.org/citation.cfm?doid=1147214.1147221>

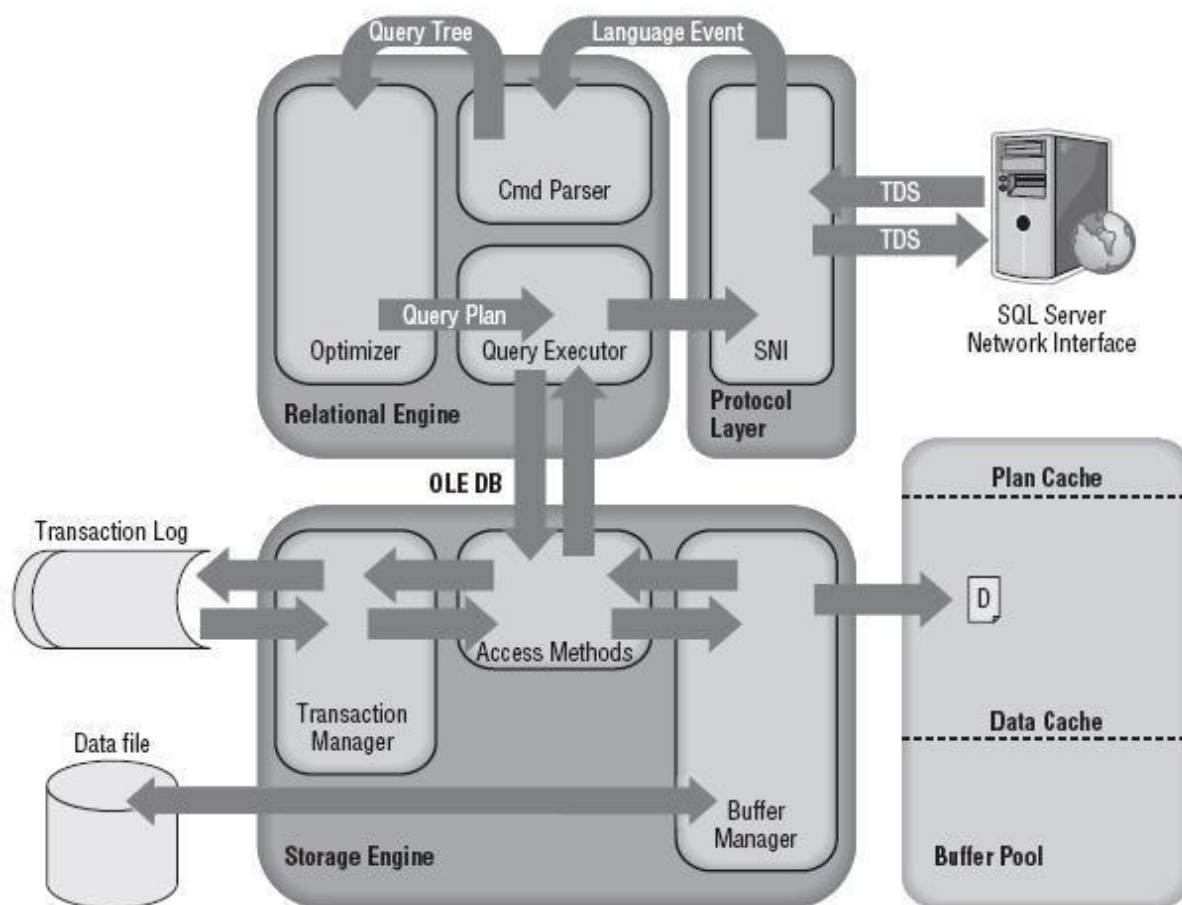
PŘÍLOHY

Příloha 1 - Architektura Oracle Database (Oracle Corporation 2016)	61
Příloha 2 - Architektura MS SQL (Microsoft 2016).....	62
Příloha 3 – Dokumentace testovací aplikace	63

Příloha 1 - Architektura Oracle Database (Oracle Corporation 2016)



Příloha 2 - Architektura MS SQL (Microsoft 2016)



UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Dokumentace aplikace

Lebedová Kateřina

Obsah

<u>Úvod</u>	65
<u>Zadání</u>	66
<u>Uživatelská dokumentace</u>	67
<u>Základní popis používané aplikace</u>	68
<u>Instalace</u>	68
<u>Přístupová oprávnění</u>	68
<u>Použití aplikace</u>	68
<u>Programová dokumentace</u>	69
<u>Datová část</u>	70
<u>Analýza</u>	70
<u>Indexy</u>	70
<u>Aplikace</u>	71
<u>Použité prostředí</u>	71
<u>Moduly</u>	71
<u>Orientace ve zdrojovém kódu</u>	78
<u>Závěr</u>	80

Úvod

Tato práce je přílohou k bakalářské práci na téma In-Memory Database Cache – analýza a zhodnocení. Obsahuje základní seznámení s aplikací pro testování databázových systémů.

Zadání

Mezi cíle bakalářské práce patří i vytvoření aplikace pro testování databázových systémů z pohledu sledování časů potřebných pro zpracování dotazů, které budou také výstupem této práce. Daná aplikace by měla obsahovat obsluhu připojení k danému databázovému systému, tvorbu testovacího prostředí, jeho smazání a provedení příslušného sledovacího testu v jazyce Java, obsluhující SQL dotazy.

Uživatelská dokumentace

Základní popis používané aplikace

Tato aplikace je vytvořena za účelem testování čtyř databázových systémů – Oracle Database 12c, Microsoft SQL Server 2016, MySQL 5.7 a PostgreSQL 9.6. Celé uživatelské prostředí bylo implementováno v jednoduchém designu, aby i méně zasvěcený uživatel pochopil, co která část vykonává.

Instalace

Spuštění aplikace je možné dvěma způsoby. V prvním případě je nutné mít nainstalováno prostředí pro spuštění Java aplikace přímo s přístupem ke zdrojovému kódu, např. prostředí NetBeans. V druhém případě, kdy dochází ke spuštění již spustitelného .jar souboru bez přístupu k aplikačnímu kódu uživateli plně postačí nainstalované Java Runtime Environment.

Přístupová oprávnění

Samotná aplikace neřeší přístupová oprávnění k databázovému systému. Je tedy nutné při připojení přihlašovat uživatele s oprávněním pro vytváření databáze.

Použití aplikace

Aplikace je rozdělena do tří částí. První část obsluhuje výběr požadovaného databázového systému, připojení k němu a následně i ukončení spojení. Druhá část obsluhuje naplnění databázového systému – tvorba testovacího databázového schématu, jeho naplnění testovacími daty a nakonec i jeho celé smazání z databázového systému. Třetí část má za úkol spuštění testu. Po spuštění testu už ho nelze přerušit jiným způsobem, než je ukončení spojení s daným databázovým systémem.

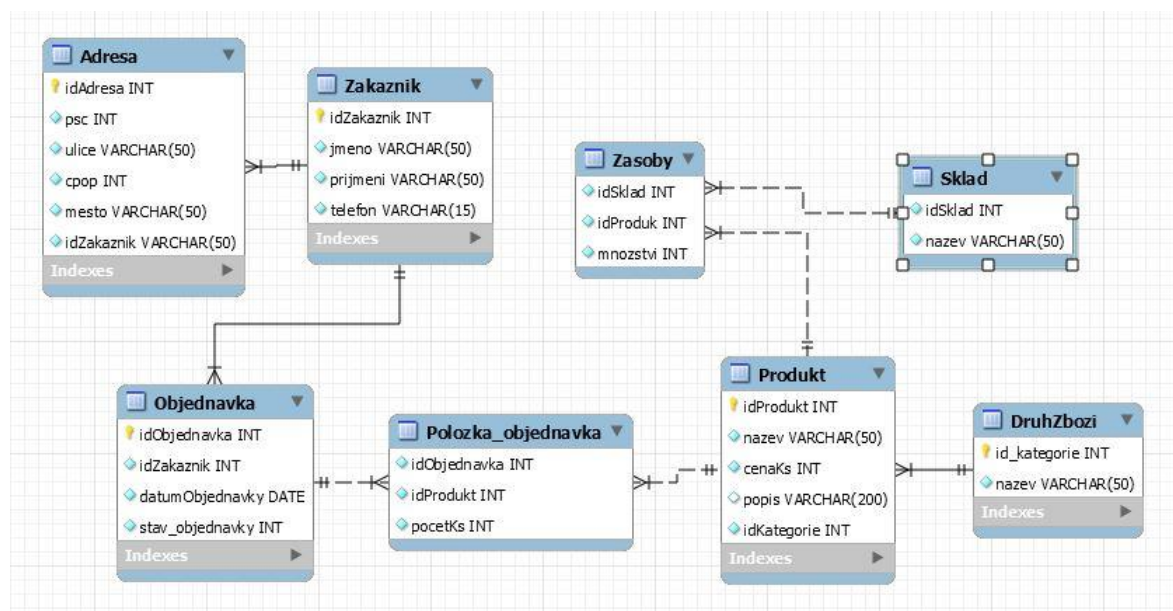
Programová dokumentace

Datová část

Analýza

Pro tvorbu testovacího prostředí v databázovém systému je použito schéma skládající se z osmi tabulek – Adresa, Zákazník, Zásoby, Sklad, Objednávka, PoložkaObjednávka, Produkt a DruhZboží. Dané tabulky obsahují převážně číselné a řetězcové datové typy pro ukládání požadovaných informací. Příložený ukázkový model z obrázku č. 1 byl vytvořen v programu MySQL Workbench 6.3, obsahuje tedy příslušné datové typy pro tento databázový systém. Veškeré vztahy mezi tabulkami jsou také demonstrační.

Tabulky Zákazník, Sklad a DruhZboží jsou samostatné tabulky, které je možno vytvořit bez závislosti na nějaký primární klíč. Adresa, Produkt a Objednávka již obsahují cizí klíče odkazující na primární klíče z tabulky Zákazník, popř. DruhZboží. Poslední dvě tabulky – Zásoby a PoložkaObjednávka slouží jako spojovací mezi příslušnými tabulkami a obsahují cizí klíče odkazující na dvě různé tabulky.



Obrázek 12 - Ukázkové schéma (vlastní zpracování)

Indexy

Celá aplikace nevyužívá indexů, které by však v mnohých případech urychlily vyhledávání v příslušných tabulkách. Použity jsou však v případě vytváření optimalizovaných tabulek pro SQL Server 2016, kdy jsou použity ve formě nonclustered¹¹ při vytváření primárních klíčů dané tabulky.

¹¹ Obsahují pouze hodnoty daného sloupce. Opakem jsou clustered indexy, který pokrývá všechny sloupce.

Aplikace

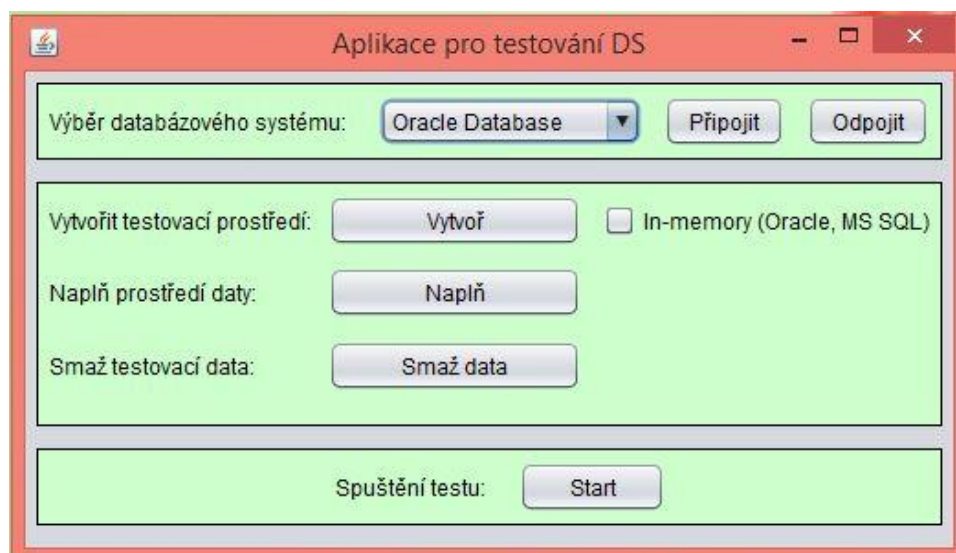
V následující části dokumentace se nachází popis jejích jednotlivých částí, tedy uživatelské části a základní orientace ve zdrojovém kódu.

Použité prostředí

Pro vytvoření aplikace bylo využito prostředí NetBeans IDE 8.2 s podporou JDK 1.8. Pro vytvoření spojení s databázovým systémem prostřednictvím aplikace bylo využito knihoven patřících k danému databázovému systému a JDBC (Java Database Connectivity).

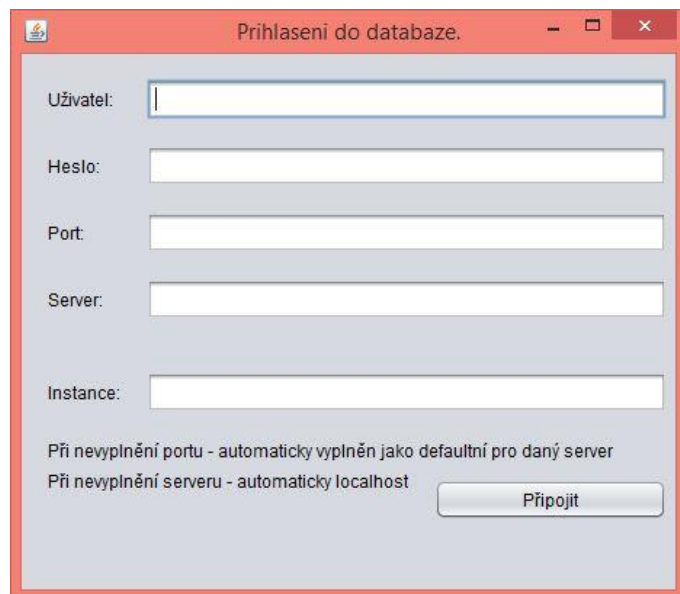
Moduly

První dialogové okno, které se zobrazí po spuštění dané aplikace a je zobrazeno na obrázku č. 2, je hlavní okno obsluhující celou aplikaci. V horní části uživatel nalezne obsluhu připojení k vybranému databázovému systému. Po rozkliknutí nabídky databázových systémů přistoupí k připojení k dané databázi.



Obrázek 13 - Aplikace (vlastní zpracování)

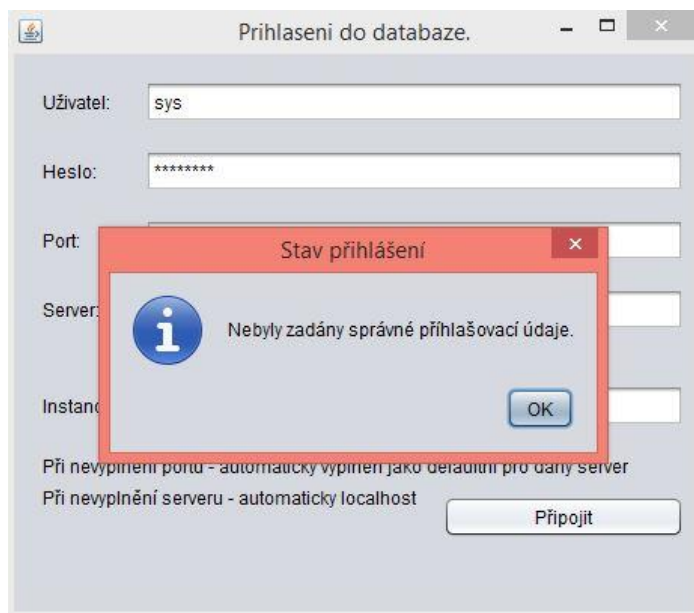
Po volbě daného databázového systému se stisknutí tlačítka Připojit aktivuje dialogové okno z obrázku č. 3, kde uživatel vyplní potřebné údaje k připojení k danému databázovému systému.



The screenshot shows a dialog box titled "Přihlasení do databaze." with a red title bar. It contains five input fields: "Uživatel:", "Heslo:", "Port:", "Server:", and "Instance:". Below these fields, there are two lines of text: "Při nevyplnění portu - automaticky vyplněn jako defaultní pro daný server" and "Při nevyplnění serveru - automaticky localhost". At the bottom right, there is a button labeled "Připojit".

Obrázek 14 - Přihlášení k DS (vlastní zpracování)

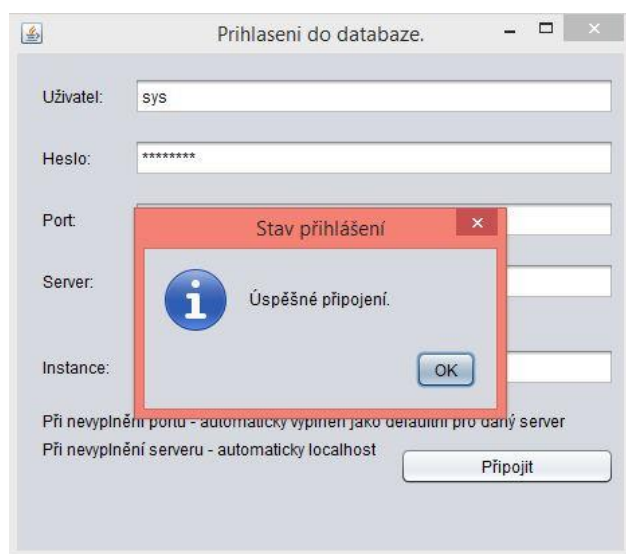
Při nevyplnění správných údajů pro připojení vyskočí dialogové okno oznamující uživateli, že něco není v pořádku (obrázek č. 4).



The screenshot shows the same "Přihlasení do databaze." dialog box, but with the "Uživatel:" field filled with "sys" and the "Heslo:" field filled with "*****". A smaller dialog box titled "Stav přihlášení" is overlaid on top. It has a red title bar and contains an information icon (i) and the text "Nebyly zadány správné přihlašovací údaje." Below this text is an "OK" button. The "Připojit" button is visible at the bottom right of the main dialog box.

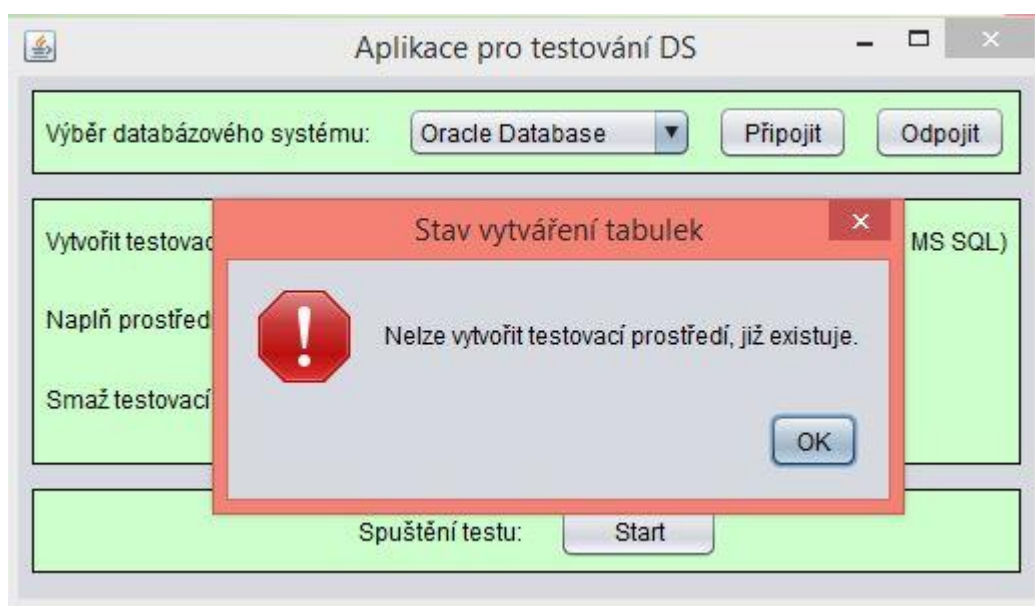
Obrázek 15 - Nesprávné přihlášení (vlastní zpracování)

V případě vyplnění správných přihlašovacích údajů se uživateli zobrazí zpráva o úspěšném připojení (obrázek č. 5) a okno s formulářem pro přihlašovací údaje se samo zavře. Uživateli se opět zobrazí hlavní okno aplikace z obrázku č. 1.

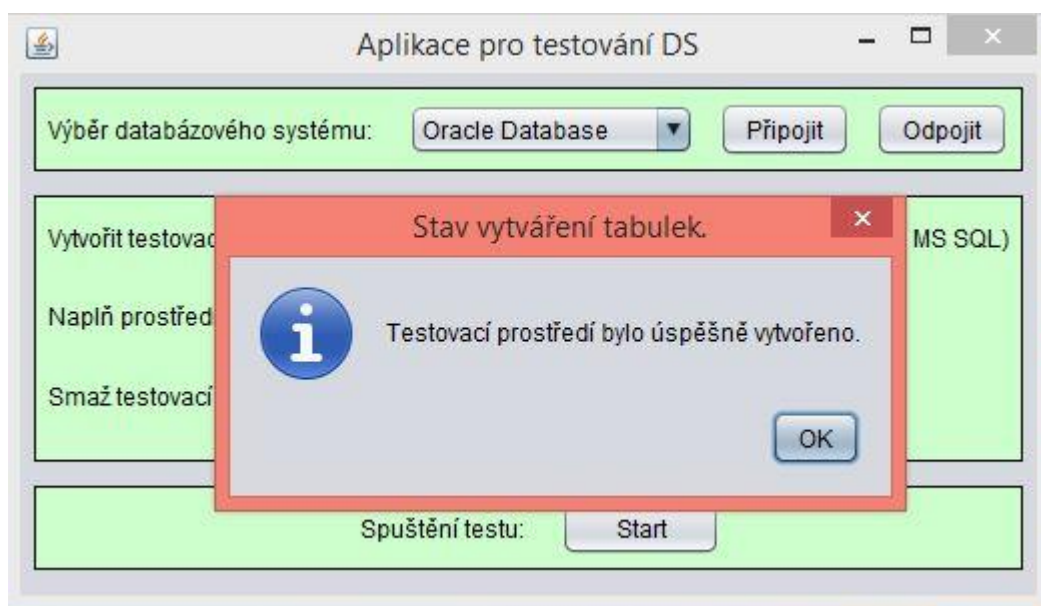


Obrázek 16 - Úspěšné přihlášení

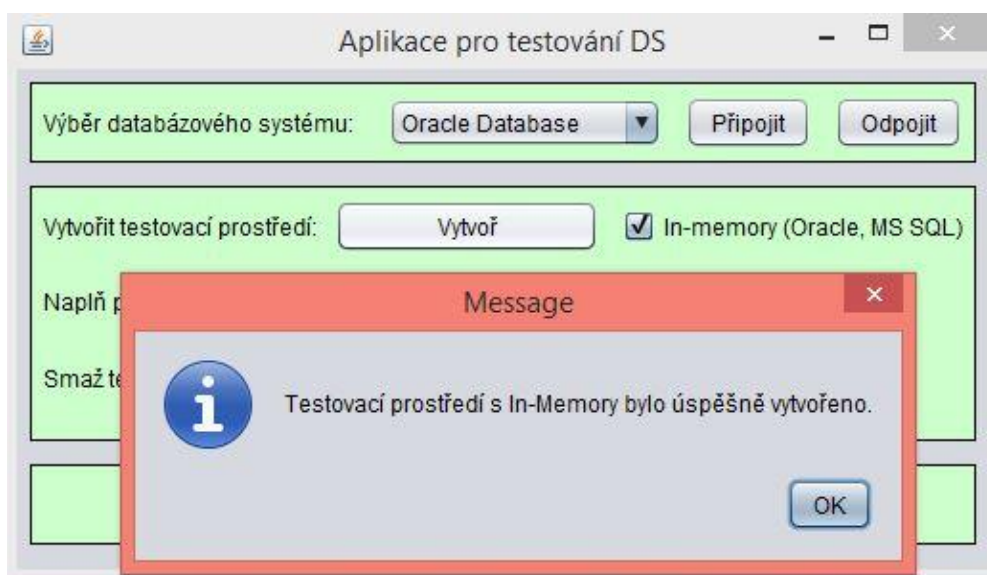
Po úspěšném přihlášení a přepnutí do hlavního okna se již přistupuje ke tvorbě testovacího prostředí tlačítkem Vytvoř. V případě připojení k databázovému systému Oracle Database nebo SQL Server 2016 si uživatel vybírá, zda vytvoří prostředí s využitím In-Memory optimalizovaného prostředí, či bez jeho využití zaškrtnutím/odškrtnutím políčka In-Memory (Oracle, MS SQL). Celá aplikace je ošetřena tak, aby nedocházelo k vytváření testovacího prostředí, které již existuje (duplicitní název tabulek/databáze). Při vzniku tohoto problému je uživateli zobrazeno informační okno upozorňující na tento stav (obrázek č. 6). Při úspěšném vytvoření vyskočí dialogové okno oznamující tento stav (obrázky č. 7 a 8).



Obrázek 17 - Chyba při vytváření existující prostředí (vlastní zpracování)

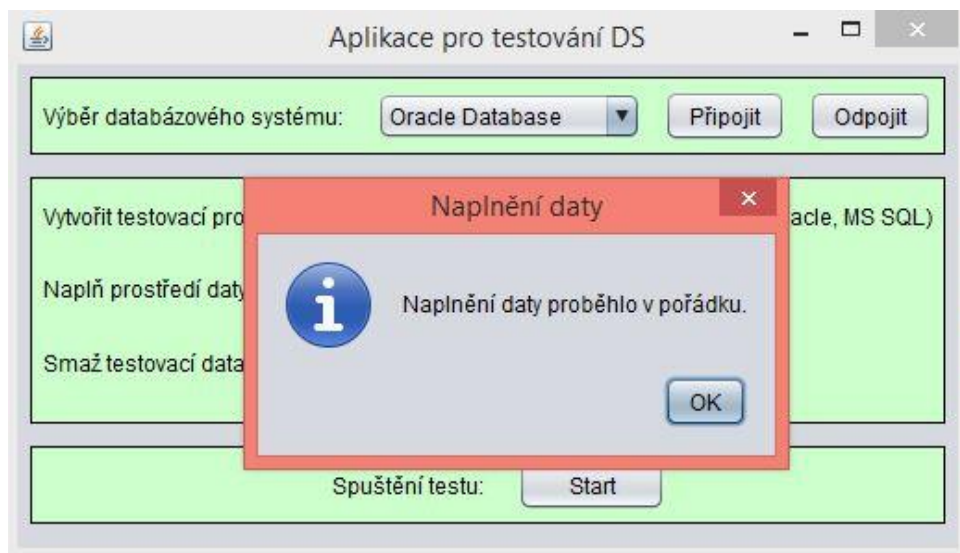


Obrázek 7 - Úspěšné vytvoření testovacího prostředí (vlastní zpracování)



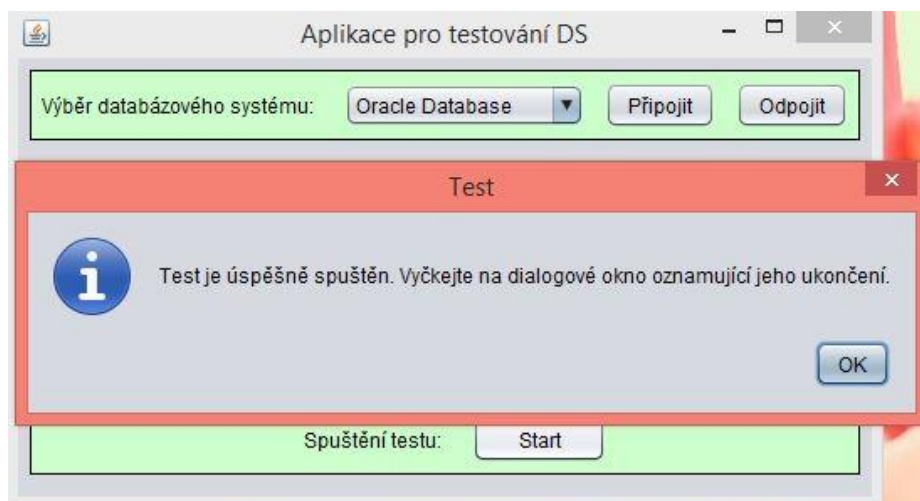
Obrázek 8 - Úspěšné vytvoření testovacího prostředí s In-Memory (vlastní zpracování)

Po vytvoření testovacího prostředí uživatel přestoupí k naplnění testovacího prostředí testovacími daty. Po úspěšném naplnění vyskočí dialogové informační okno (obrázek č. 9). Pro přehled času, který tato činnost zabrala a výsledek se objeví v textovém souboru ve složce, ze které je aplikace spouštěna.



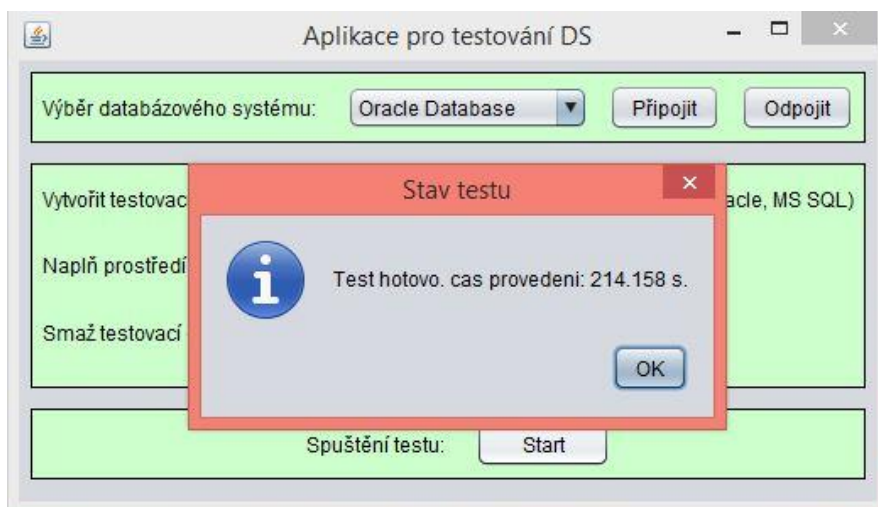
Obrázek 9 - Naplnění daty (vlastní zpracování)

Po vytvoření a naplnění testovacího prostředí už může uživatel přistoupit ke spuštění samotného testu kliknutím na tlačítko Start. Po spuštění testu se uživateli zobrazí informační okno oznamující tuto skutečnost (obrázek č. 10).



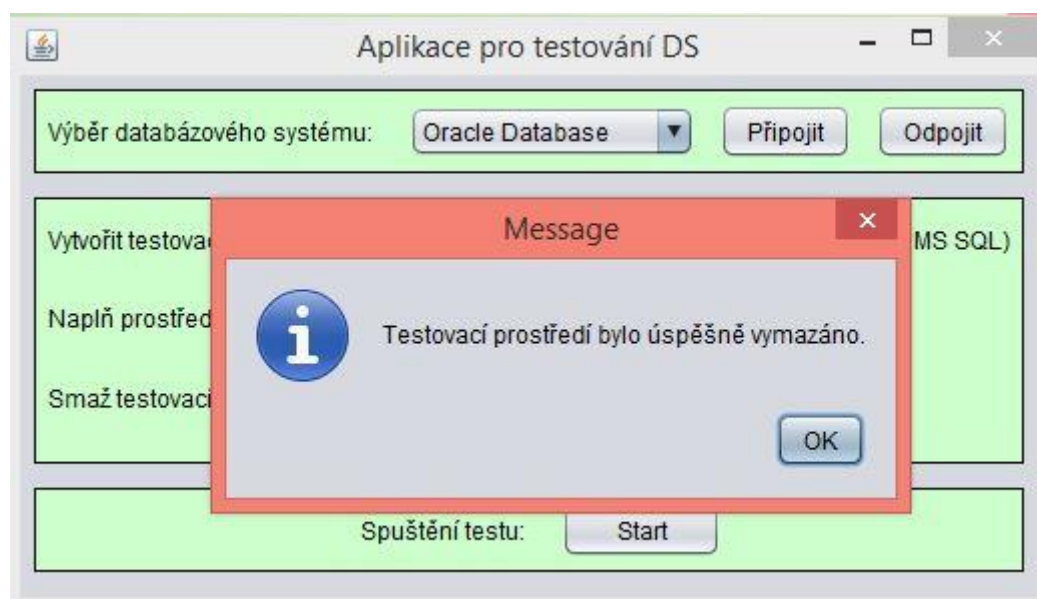
Obrázek 10 - Spuštění testu (vlastní zpracování)

Po skončení testu se zobrazí dialogové okno informující o ukončení testu společně s časem, který byl potřebný k jeho vykonání (obrázek č. 11).



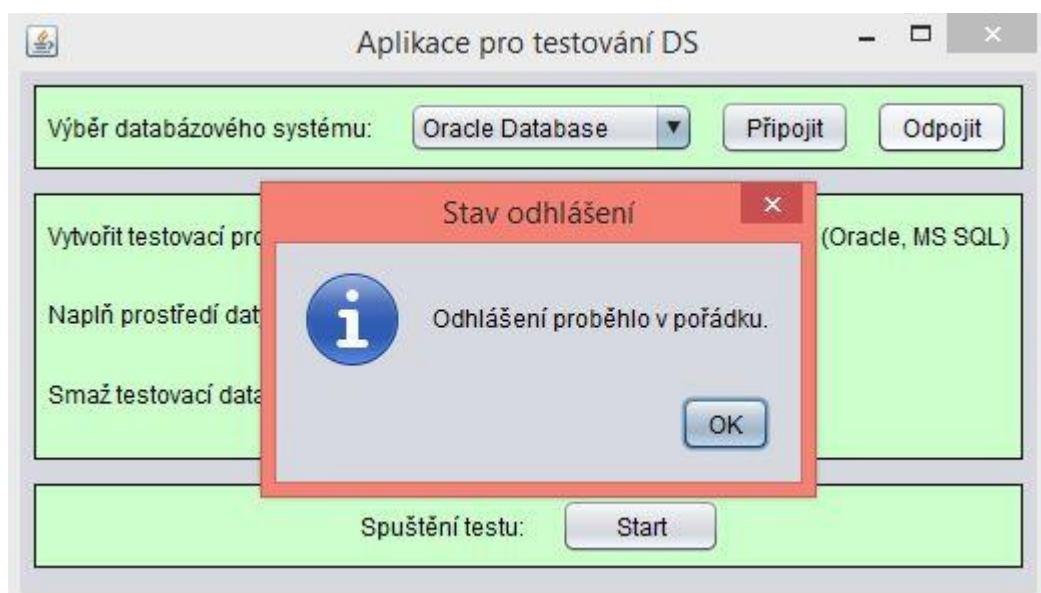
Obrázek 11 - Ukončení testu

Po ukončení testu se uživateli poskytuje nabídka pro smazání testovacího prostředí tlačítkem Smaž data, kterému následuje oznamovací okno o úspěšném provedení této činnosti (obrázek č. 12). Výsledek testu se zobrazí v textovém souboru nacházejícím se ve složce, ze které je daná aplikace spuštěna.



Obrázek 12 - Úspěšné smazání testovacího prostředí (vlastní zpracování)

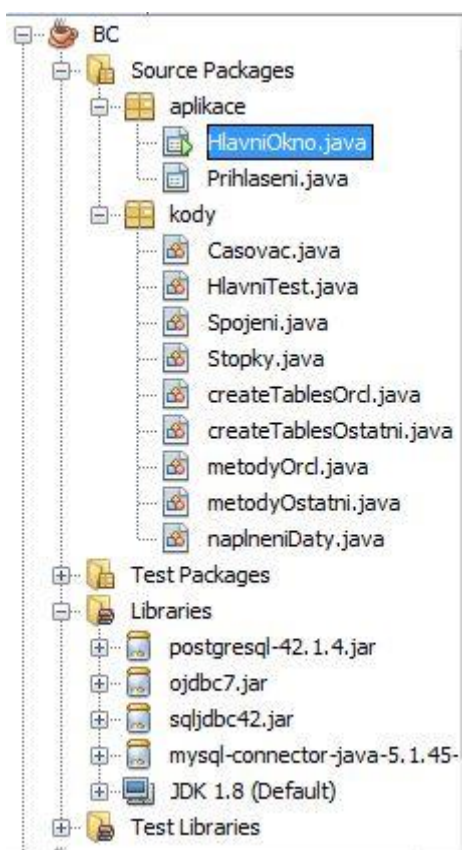
Poslední možností je odpojení od daného databázového systému tlačítkem odpojit. Tato operace je opět provázena dialogovým oknem o úspěšnosti této operace (obrázek č. 13).



Obrázek 13 - Úspěšné odhlášení (vlastní zpracování)

Orientace ve zdrojovém kódu

Celá aplikace je obsažena ve dvou složkách daného projektu. První složka obsahuje 2 balíčky, které oddělují aplikační část (balíček aplikace) od části s pomocnými třídami (balíček kody). Druhá složka – Libraries – obsahuje potřebné knihovny obsluhující připojení k danému databázovému systému. Celé toto rozdělení je vyobrazeno na obrázku č. 13.



Obrázek 13 - Rozdělení projektu (vlastní zpracování)

V balíčku aplikace se nachází dvě třídy – HlavniOkno a Prihlaseni. HlavniOkno slouží ke spuštění celé aplikace a obsahuje příslušné metody pro volání potřebných metod z tříd obsažených v části kódů. K němu je přiložen formulář Prihlaseni, který obstarává zpracování přihlašovacích údajů k danému databázovému systému.

Druhý balíček kody obsahuje třídy pro obsluhu daných operací. Třída Casovac obstarává zpracování naměřených časů, které jsou potřebné pro vykonání dané operace. HlavniTest obsahuje metodu obstarávající přepočet bloků mezi částí samotného testu a jejich spuštění. Třída Spusteni slouží jako pomocná pro předávání existující připojení mezi třídami. Stopky spolupracují s třídou Casovac, které odesílají daný čas zpracování příslušné operace, aby ho mohla přepočítat na potřebné výstupní hodnoty. Třídy CreateTablesOrcl a CreateTablesOstatni obsahují příslušné proměnné typu String obsahující již samotné SQL operace nutné k vytvoření daného testovacího prostředí či jeho smazání. Tyto proměnné jsou zpracovány v třídách metodyOrcl a metodyOstatni.

Poslední třídou celého projektu je `naplneniDaty`, která s pomocí přiložených souborů obsahující dotazy s hodnotami pro daný databázový systém vytvoří testovací data v databázi.

Příslušné metody ve zmíněných třídách jsou patřičně okomentovány v kódu, aby bylo jasné, kterou činnost provádějí.

Závěr

Požadovaným cílem bylo vytvoření jednoduché aplikace, která by obsluhovala zpracování časů při zátěžovém testu nad databázovým systémem. Vytvořená aplikace ovládá potřebné prvky nezbytné k zrealizování tohoto cíle. Obsahuje jednoduché uživatelské prostředí, což je velkou výhodou pro budoucího méně zasvěceného uživatele. Chybí ošetření některých chybných uživatelských vstupů, což však nezapříčinilo velké problémy při testování.

V budoucnu by se dále mohla aplikace rozvíjet z pohledu rozšíření nabídky testovaných databázových systémů, popř. vytvoření více spuštěných vláken pro daný test, kdy by se dalo sledovat chování databázového systému při více připojeních. Výsledky by se tedy více přiblížily reálnému zatížení.