Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Optimalizace výkonu databázových systémů

Jan Mikolášek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jan Mikolášek

Informatika

Název práce

Optimalizace výkonu databázových systémů

Název anglicky

Performance optimalization of database systems

Cíle práce

Diplomová práce je zaměřena na řešení výkonu relačních databází. Hlavním cílem práce je popsat postupy při ladění výkonu relačních databází s popisem oblastí, které nejvíce ovlivňují celkový výkon systémů relačních databází. Dílčími cíli této diplomové práce jsou následující oblasti:
- popis návrhu relační databáze s ohledem na následný výkon,
- rozbor a popis typických problému s výkonem relačních databází,
- popis nástrojů pro analýzu výkonu relačních databází,
- možnosti řešení ladění výkonu relační databáze,
- praktické ověření navržených postupů na existujících relačních databázových systémech Oracle, IBM DB2 a Microsoft SQL.

Metodika

Metodika řešené problematiky je založena studiu odborných podkladů pro návrh relačních databází, studium aspektů ovlivňujících výkon relační databáze z pohledu fyzického i logického rozložení. Dále následuje návrh vzorové relační databáze, na které budou aplikovány získané poznatky a validovány s očekávanými výstupy v praktické části. Na základě těchto poznatků budou popsána doporučení pro návrh relační databáze s ohledem na výkon.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

Relační databáze, výkon, databázová tabulka, relační model, databázový index, databázové schéma.

Doporučené zdroje informací

Ciro Fiorillo. Oracle Database 11 g R2 Performance Tuning Cookbook. Packt Publishing, 2012. 542s. ISBN: 978-1-84968-260-2

CHAUDHURI, Surajit; NARASAYYA, Vivek. Self-tuning database systems: a decade of progress. In: Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 2007. p. 3-14.

Oracle Database Performance Tuning Guide. [online]. < https://docs.oracle.com/cd/E11882 _ 01/server.112/e41573/toc.htm>

Pokorný J., Valenta M. Databázové systémy. ČVUT Praha, Česká technika – vydavatelství ČVUT, 2013. 274 s. ISBN: 978-80-01-05212-9

Sam Lightstone, Toby Teorey, Tom Nadeau. Physical Database Design. Morgan Kaufmann Publishers, 2007. 448s. ISBN-13: 978-0-12-369389-1

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Jan Tyrychtr, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 21. 10. 2016

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 10. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 21. 03. 2017

Čestné prohlášení

Prohlašuji, že svou diplomovou práci ,,Optimalizace výkonu databázových systémů" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 28. 3. 2017 _____

Poděkování

Rád(a) bych touto cestou poděkoval vedoucímu diplomové práce Ing. Janu Tyrychtrovi, Ph.D. za ochotu a odborné vedení při vypracování této diplomové práce.

Optimalizace výkonu databázových systémů

Souhrn

Hlavním cílem této práce je navrhnout vhodné prostředky pro ladění výkonu relačních databází. Relační databáze je základem takřka každého informačního systému, a její výkon ovlivňuje celkovou výkonnost tohoto systému. Vlastní výkonnost relační databáze je dána už při samotném návrhu fyzické i logické architektury databáze a následně také pravidelným ověřováním správnosti návrhu a reakcí na měnící se požadavky. Cílem této práce je stanovit správný postup při návrhu relační databáze, určit klíčové oblasti, na které je potřeba se zaměřit při návrhu a průběžném monitoringu databázového systému. Tyto poznatky budou následně prakticky ověřeny na fiktivní databázi s využitím databázových systémů od společností IBM, Oracle a Microsoft. V práci je nejdříve stanovena metodika, podle které bude postupováno k dosažení hlavního cíle této práce. V následující části jsou popsány teoretické základy jednotlivých komponent relační databáze a také vybraných částí, které mají vliv na výkon relační databáze. V praktické části bude vytvořena fiktivní databáze a bude zkoumáno její chování v různých situacích dle stanovené metodiky. V závěru práce budou využity výsledky praktické části ke stanovení správného procesu návrhu relační databáze poskytující optimální výkon při různých způsobech implementace.

Klíčová slova: výkonnost, relace, databáze, tabulka, databázový index, hledání, třídění, komprese, optimální cesta, dotaz, časová složitost

Performance optimization of database systems

Summary

The main goal of this thesis is to identify the appropriate means for performance tuning of relational databases. A relational database is the foundation of almost any information system, and hence its performance affects the overall performance of the system. The performance is based on physical and logical draft of database architecture and also a regular verification of the design and reaction to changing requirements. The aim of this study is to determine the correct procedure for designing relational databases, to identify key areas that need to be addressed in the design. These areas will then be practically verified on a fictitious database using database systems from IBM, Oracle and Microsoft. At the beginning methodology for this work will be specified to achieve the main objective of this thesis. The following section will describe the theoretical foundations of the individual components of the relational database and the parts that have the impact on relational database performance. There will be a fictitious database which will be used in practical part to examine behavior in different situation. In conclusion, the results will be used to determine the proper process of relational database design provides optimal performance in a variety of ways of implementation.

Keywords: performance, relation, database, table, database index, search, sorting, compression, optimal path, query, time complexity

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Databáze tvoří základ většiny komplexních informačních systémů (IS). Databáze poskytují aplikacím data, se kterými tyto aplikace pracují a uživatelům poskytují požadované informace. Databáze může být součástí řešení pro malý e-shop, stejně tak i součástí systému řízení vztahu se zákazníky velké společnosti. V současné době existuje více typů databází, kdy k nejznámějším stále patří relační databáze, jejíž základy položil v roce 1970 Ted Codd, který přišel právě s relačním datovým modelem. Mezi další typy databází patří například objektová databáze nebo v současné době se rozvíjející noSQL databáze. Tato práce je věnována pouze relačním databázím, a to z důvodu, že tento typ databáze je stále nejvyužívanějším a nejrozšířenějším typem databáze, i když už existují oblasti, pro které by mohl být jiný typ databáze vhodnější.

Tato práce si klade za cíl popsat postup jak návrhu vlastní struktury relační databáze, tak i databázového systému s ohledem na celkový výkon v různých variantách využití. Jak již bylo napsáno, většina komplexních IS se neobejde bez databáze, a proto výkon databázového systému ovlivňuje výkon celého informačního systému. Relační databáze již mají svou dlouhou historii, v rámci které se měnily i požadavky na výkon, a to hlavně s ohledem na jejich využití. V dřívějších dobách velkých sálových počítačů byla data z databází využívána zpravidla jedním systémem, který data zpracoval a následně předal výsledek. Nebylo nutné, a většinou hlavně ani možné, zpracovávat více úloh paralelně. S rozvojem počítačových sítí se zvyšovaly i požadavky na výkon databázového systému, neboť bylo možné databáze poskytnout více uživatelům, kteří ji využívali ke své pracovní činnosti a požadovali tedy rychle dostupné informace. V současné době rozvoje internetu jsou již požadavky na výkon nejen databázových systémů maximální a informace je potřeba poskytovat takřka v reálném čase. Kvalitu jakéhokoliv informačního systému může zásadně snížit skutečnost, kdy je práce s ním zdlouhavá a na každý požadavek je nutné čekat. Uživatelé od aplikací očekávají nízkou dobu odezvy a dá se říct, že už v případě, kdy tento čas přesáhne více jak jednu vteřinu na každý požadavek, uživatel může takovouto aplikaci, kterou používá ke své celodenní práci, požadovat za nevyhovující. Uživatel při práci s takovouto aplikací posílá denně desítky až stovky požadavků a postupným načítáním prodlevy v odezvě aplikace může být výsledný promarněný čas značný, a to nejen pro samotného uživatele, ale také i pro jeho zaměstnavatele.

Z pohledu používání relační databáze se dá říci, že existují dva druhy implementace databázového systému. První a častější způsob využití je provozní databáze, kde jsou víceméně rovnoměrně rozprostřeny požadavky na čtení, zápis nebo modifikaci dat. Druhým příkladem může být tzv. datový sklad, kdy je databáze primárně využívána k tvorbě reportu nad daty a majoritní podíl požadavků se týká čtecích operací pro potřeby analýzy dat. V obou případech bude potřeba k návrhu databáze přistoupit odlišným způsobem, aby byly maximálně splněny požadavky na výkon. Je tedy jasné, že základem pro správný návrh databázového řešení je nutné v první řadě dobře porozumět požadavkům na reálné využití. A následně je pro každý jednotlivý případ užití nutné zvolit odlišný způsob návrhu databázového systému.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této práce je popsat a ověřit postupy pro ladění výkonu relační databáze ve zvolených oblastech. K dosažení tohoto cíle vedou následující dílčí kroky:

- rozbor a popis typických oblastí ovlivňujících výsledný výkon relační databáze,
- popis nástrojů správce databáze pro analýzu výkonu,
- návrh relační databáze a jejich struktur s ohledem na požadavky pro dosažení optimálního výkonu,
- praktické ověření teoretických informací v praxi za použití nejrozšířenějších systémů pro správu relačních databází, kterými jsou Oracle, IBM DB2 a Microsoft SQL server.

2.2 Metodika

Metodika tvorby této práce je v úvodu založena na studiu odborné literatury k získání informací o možnostech současných databázových systémů pro optimalizaci výkonu. Tyto nabyté znalosti budou následně použity a zkoumány v praktické části.

V úvodu praktické části bude nejprve vytvořeno prostředí, které bude použito pro následné testování vybraných oblastí pro optimalizaci výkonu relační databáze. Prostředí bude připravováno v postupných krocích. Nejprve bude navržena fiktivní relační databáze, která bude splňovat požadavky pro následně prováděná měření. Dalším krokem bude implementace všech databázových systémů, na kterých bude implementována navržená databáze a budou prováděna měření. Vzhledem k potřebě co nejstálejších podmínek měření bude připraven další oddělený databázový systém, který bude sloužit jako základna pro data navržené databáze. Odtud budou před zahájením měření distribuována data na testované databázové systémy. Pro potřeby generování testovacích dat bude vyvinuta jednoduchá java konzolová aplikace, která bude generovat náhodná data do databáze. Vzhledem k podstatě fungování relačních databázových systémů je možné získat relevantní výsledky až v případě

většího objemu dat, a bude tedy potřeba vygenerovat alespoň jednotky milionů záznamů do tabulek databáze.

Pro potřeby vlastních testů bude vyvinuta další jednoduchá java konzolová aplikace, která bude generovat požadovanou zátěž na databázový systém. Klíčovou vlastností této aplikace bude možnost spouštět dotazy více vlákny a tím simulovat reálnou práci více koncových uživatelů databázového systému. Aplikace bude na vstupu mimo počtu požadovaných vláken spouštějící dotazy očekávat i typ databázového systému, typ prováděného testu a počet cyklů v testu. Tato aplikace také zajišťuje důležitou potřebu pro měření a tím je identický způsob zasíláním dotazů do všech použitých databázových systémů. Výsledek tedy nebude ovlivněn měnící se režií různých aplikací. Připojení k jinému databázovému systému bude realizováno pouhou výměnou ovladače bez nutnosti modifikace vlastního kódu aplikace či kompletní změně použité aplikace. Aplikace bude instalována na stejný fyzický stroj jako databázové systémy a odtud bude i spouštěna. Tím bude vyloučeno co nejvíce externích vlivů na výsledky měření, například síťové latence. Z tohoto důvodu bude při vývoji testovací aplikace brán ohled na minimalizaci využívání zdrojů počítače, kde je test spuštěn. Pro každý dotaz v rámci konkrétního měření bude uložen čas zpracování dotazu. Následně bude vypočten průměr času pro provedení dotazu v součtu pro všechna vlákna a cykly a tento bude brán za výsledný čas pro konkrétní typ měření a databázového systému. Pro větší zpřesnění výsledků bude každé měření provedeno vícekrát a výsledný čas bude určen jako medián hodnot všech provedených měření konkrétního typu testu. Následně budou všechny hodnoty porovnány s očekávanými výsledky mezi použitými databázovými systémy. Z dosažených výsledků bude odvozeno doporučení pro realizaci relační databáze při různých způsobech užití.

3 Teoretická východiska

3.1 Databáze

Vlastní databázový systém by se dal rozdělit na dva logické celky, a to vlastní databáze a softwarové vybavení systém řízení báze dat (SŘBD). Název systém řízení báze dat vznikl překladem anglického výrazu DataBase Management System (DBMS) do češtiny. Podle Bouchera je SŘBD program, který umožňuje uživatelům uložit, zobrazit a modifikovat data bez nutné znalosti o jejich fyzickém umístění nebo organizaci jejich uložení (BOUCHER & Ali, 2006).

Výkon databázového systému se stejně jako u ostatních IS dá navyšovat pomocí škálování. Takovýto způsob je spíše volen v případech, kdy jsou na databázový systém kladeny vysoké nároky co do počtu paralelně zpracovávaných transakcí (uživatelských dotazů). Zde je možné využít některého z dostupných klastrových řešení a navýšit počet fyzických serverů, které požadavky na databázový systém zpracovávají. Nicméně v případě nevhodně zvoleného logického nebo fyzického modelu relační databáze nemusí navýšení počtu fyzických serverů pomoci vyřešit problém s celkovým výkonem.

V nejvyšším patře dělení implementace relačních databází existují dva typy, provozní databáze a datový sklad. V případě provozní databáze se jedná se o klasickou relační databázi většinou poskytující data pro uživatelské aplikace. Dle Nielsena provozní databáze, neboli online transaction processing (OLTP), shromažďuje transakční data, která jsou nutná pro každodenní činnosti společnosti a jsou pro ni unikátní (Nielsen, White, & Parui, 2009). V databázi jsou většinou udržována data nutná pro práci uživatelů, historická data jsou po čase přenášena do druhého typu implementace relační databáze. V některých případech se dá ještě hovořit o dělení na aktivní a pasivní data. Aktivní data jsou ta, která jsou využívána v majoritě případu, zatímco pasivní data již nejsou tak často požadována, ale je potřeba je v databázi udržovat z důvodu nutnosti globálního pohledu na data, například k získání informací o historii vztahu s aktuálním zákazníkem.

Druhý typ, datový sklad, funguje jako takové odkladiště nepotřebných dat pro každodenní práci uživatelů, ale důležitých pro vedení společnosti z historických důvodů. Dle Inmonna je datový sklad definován jako soubor neměnných dat orientovaných na konkrétní

oblast. Jsou zde udržovaná jak historická, tak aktuální data integrovaná z mnoha zdrojů pro potřeby rozhodování vedením společnosti na základě získaných informací (Inmonn, 2002). Dle Nagabhushana je datový sklad definován jako informační systém s následujícími vlastnostmi (Nagabhushana, 2006):

- databáze designovaná pro analytické činnosti využívající data z mnoha zdrojů,
- je používán relativně malým počtem uživatelů s dlouhými interakcemi,
- je náročná na čtení dat,
- obsah je pravidelně modifikován, nejvíce pomocí nových dat,
- obsahuje aktuální i historická data pro poskytnutí pohledu vývoje informací z historického hlediska,
- obsahuje málo velkých tabulek.

3.2 Pojem relace v relační databázi

Relační model zavedl koncem šedesátých let pracovník společnosti IBM Edgar Frank Codd. Relačním modelem je nazýván proto, že je velice podobný matematické struktuře zvané relace. Relační databáze je potom dána množinou relací (Pokorný & Valenta, 2013). V databázovém světě se pojem relace převádí na tabulku, kdy se prvkem relace rozumí řádek v tabulce. Tento řádek tabulky určuje vztah mezi daty. Například, existuje-li tabulka zákazníků definována sloupci Číslo zákazníka, Jméno, Příjmení, Telefonní číslo, pak záznam v řádku tabulky 007, Jan, Novák, 603123456 určuje vztah mezi zákazníkem Jan Novák s jeho unikátním identifikátorem 007 a telefonním číslem 603123456. Při návrhu databáze definujeme tabulky neboli relace a hovoříme o relačním modelu dat. Relační model dat naplňuje následující ideje charakteristické pro relační databáze (Pokorný & Valenta, 2013):

- důsledně se oddělují data, která jsou chápana jako relace, od jejich implementace,
- při manipulacích s daty se nezajímáme o přístupové mechanismy k datům obsažených v relacích,
- pro manipulaci s daty jsou k dispozici dva silné prostředky, relační kalkul a algebra, které slouží jako základ uživatelských relačních jazyků,

- pro omezení redundance dat v relační databázi jsou k dispozici pojmy umožňující normalizovat relace, tedy navrhovat potřebné relační databázové struktury podle přesně definovaných kritérií.

V relačním modelu se datové struktury modelují pomocí matematických relací popsaných schématem relace. Schéma relace je dáno jménem relace a množinou atributů relace (A). Atribut relace je dvojice A:D, kde A je jméno atributu a D doména reprezentující množinu hodnot (Pokorný & Valenta, 2013). Atribut, v některých publikacích nazývaný také jako pole, je sloupec v tabulce databáze. Podle Codda je relace v matematickém smyslu definována následovně. Existují-li množiny S1, S2, ….., Sn (nemusí být nutně odlišné), R je relace na těchto n množinách, pokud se jedná o množinu n-tic, kdy každá z nich má první prvek z množiny S1, druhý z S2 atd. Prvek Sj nazýváme j-tou doménou relace R (Codd, 1970).Uvažujeme-li schéma relace R (A1:D1,…,An:Dn), pak číslo n nazýváme arita relace R. Relace R* nad schématem R je podmnožinou kartézského součinu D1 ×….. × Dn. Prvky relace se nazývají n-tice a mají tvar (a1,……,an). Každá komponenta je atomická, a tedy dále nedělitelná (Pokorný & Valenta, 2013).

Pro databázové aplikace je užitečné omezit množinu dat v tabulce (relaci) na informace obsahující pouze smysluplná data. K tomuto účelu se v relačních databázích používá takzvaného integritního omezení. Definováním jednoho, či více integritních omezení, je určeno, jaká data je možné do relační databázové tabulky vložit. Integritní omezení je samozřejmě aplikováno i v případě modifikace nebo odstranění dat. Integritní omezení se definuje na atributu nebo atributech relace tím, že se specifikuje vlastnost dat, které může konkrétní atribut obsahovat. Mezi nejdůležitější integritní omezení patří primární klíč. Pomocí je určena skutečnost, že data v atributu (atributech) jsou v celé relaci jedinečná, tedy neexistuje duplicitní záznam. Volba tohoto klíče se provádí výběrem klíče z množiny kandidátních klíčů, což je množina atributů relace obsahující pouze unikátní záznamy. Příkladem primárního klíče z výše zmíněného příkladu relace může být číslo zákazníka. Integritní omezení primárního klíče může existovat pro každou relaci maximálně jedno.

Dalším důležitým integritním omezením je referenční integrita, která představuje binární vztah dvou množin atributů. Existují-li dvě relační schémata R(A) a S(B) s primárními klíči KR a KS a XR, je atribut z množiny A. Pak referenční integrita je dána dvojicí (XR, KS) a je splněna na relacích R* a S*, jestliže pro každou n-tici u ∈ R* existuje

n-tice v ∈ S* tak, že u[XR] = v[KS] (Pokorný & Valenta, 2013). Tomuto integritnímu omezení se říká cizí klíč.

3.2.1 Relační algebra

V relačním modelu se pro práci s daty používá nástroj zvaný relační algebra. Operátory relační algebry se aplikují na relace a výsledkem vyhodnocení operací jsou opět relace. Relační algebra je založena na pěti základních operacích (Pokorný & Valenta, 2013):

- kartézský součin (×)
- sjednocení (∪)
- rozdíl (-)
- selekce
- projekce

Kartézský součin je množinová operace tvořící základ pro důležité operace spojení. Vstup této operace tvoří dvě relace a výstupem je opět relace mající všechny atributy z obou relací a řádky tvoří všechny kombinace řádku z obou relací.

Operace sjednocení poskytuje nástroj pro sloučení dvou relací při splnění podmínky stejné arity a stejného schématu obou relací. Výsledkem této operace je pak relace s n-ticemi o počtu odpovídající součtu n-tic obou relací.

Operace rozdílu R1 – R2 poskytuje nástroj pro získání relace obsahující n-tice z relace R1, které nejsou součástí relace R2.

Selekce je zadána pomocí logické podmínky, která je aplikována na relaci R. Výsledkem je relace R, která obsahuje pouze n-tice splňující danou podmínku. Podmínka je obvykle booleovský výraz jednoduchých podmínek ve tvaru X Θ Y a X Θ c, kde Θ je symbol porovnávacího predikátu jako například < > = (Pokorný & Valenta, 2013).

Projekce relace R(A) na množinu atributů D ⊆ A značíme R[D], je relace se schématem obsahující atributy D (Pokorný & Valenta, 2013). Jinými slovy řečeno, výsledkem je relace obsahují všechny n-tice relace obsahující pouze určené atributy.

Operace spojení využívá výše zmíněné operace kartézského součinu, selekce a projekce. Pomocí kartézského součinu získáme veškeré kombinace spojovaných relací, pomocí selekce je určen vztah, podle kterého má být spojení vytvořeno a podmínku, kterou

musí splňovat řádky na výstupu. Pomocí projekce jsou určeny atributy z obou relací, které se mají objevit na výstupu. Nejpoužívanější spojení je přirozené spojení, to spojuje dvě relace přes jejich největší společnou množinu atributů. Existují-li dvě relace se schématy R(A) a S(B), schéma relace R * S má atributy A ∪ B a n-tice (R * S)* definované následujícím způsobem. Jsou-li R* a S* instance R, respektive S, pak výsledná relace je dána množinou {u | u[A] ∈ R* ∧ u[B] ∈ S*} (Pokorný & Valenta, 2013). Existuje ještě druhá varianta spojení, kterou je vnější spojení, a to buď levé, pravé nebo plné. V tomto případě se do výsledku dostanou i n-tice, které nemají odpovídající protějšek v druhé relaci.

3.2.2 Normalizace

V průběhu procesu návrhu relační databáze je jedním z kroků takzvaná normalizace, což je proces, za pomoci kterého dostaneme relační model do normální formy. Normalizace je proces organizace dat v databázi. Normální formy definované v teorii relačních databází reprezentují podklad pro design záznamů v databázi (Kent, 1983). Existuje celkem pět úrovní normálních forem, které na sebe navazují a pro splnění vyšší normální formy je vždy potřeba dodržet podmínky všech předchozích normálních forem. Hlavním úkolem procesu normalizace je navrhnout relační model tak, aby obsahoval minimum redundantních dat. V praxi se většinou provádí návrh relačního modelu tak, aby splňoval podmínky až do třetí normální formy. Vyšší normální formy mnohdy nejsou vhodné z pohledu výkonu relační databáze, a proto se někdy stává, že se provádí tzv. denormalizace, což je opačný postup, který zvyšuje na jedné straně redundanci dat, ale na straně druhé také výkon. Následuje popis normálních forem dle Stephense (Stephens, 2009):

- 1. Normální forma
    - Každý sloupec v tabulce má unikátní název.
    - Nezáleží na pořadí řádků a sloupců v tabulce.
    - Každý sloupec je jednoho datového typu.
    - Dvě řádky tabulky neobsahují identická data.
    - Každý sloupec musí obsahovat jednu hodnotu.
    - Sloupce v tabulce nemohou obsahovat opakující se skupiny hodnot.

- 2. Normální forma
    - Tabulka je v první normální formě
    - Všechny neklíčové hodnoty ve sloupcích tabulky jsou závislé na celém hlavním klíči tabulky, primárním klíči. Znamená to také to, že pokud existuje primární klíč, který obsahuje více atributů, musí být všechny ostatní atributy přímo závislé na všech atributech klíče.
- 3. Normální forma
    - Tabulka je v druhé normální formě.
    - V tabulce neexistuje tranzitivní závislost, tedy taková, kdy jsou na sobě vzájemně závislé neklíčové hodnoty ve sloupcích tabulky.
- Boyce-Codd Normální forma. Jedná se o variaci třetí normální formy
    - Tabulka je ve třetí normální formě.
    - Každý determinant je kandidátní klíč. Determinant je v tomto případě sloupec v tabulce, který částečně určuje hodnotu v jiném sloupci tabulky. Kandidátní klíč je minimální množina sloupců, které jednoznačně určují záznam v tabulce. Jinými slovy, tabulka je v BCNF, pokud v ní neexistují překrývající se kandidátní klíče.
- 4. Normální forma
    - Tabulka je v BCNF formě.
    - Tabulka neobsahuje žádné nesouvisející vícehodnotové závislosti. Jinými slovy všechny hodnoty sloupců tabulky popisují pouze jednu skutečnost/souvislost.
- 5. Normální forma
    - Tabulka je ve čtvrté normální formě.
    - Tabulka neobsahuje žádné spolu související vícehodnotové závislosti.

3.3 Časová složitost

Zajímáme-li se o výkon jakéhokoliv počítačového programu, databázové systémy nevyjímaje, většinou nás zajímá čas, který je nutný pro zpracování konkrétní úlohy. Čas zpracování úlohy je závislý na zvoleném algoritmu a velikosti vstupních dat. Pro možnost

porovnání různých algoritmů byl zaveden pojem asymptotická složitost, značená pomocí řeckého písmene O (Omikron). Pro označení algoritmu a velikosti vstupních dat je použita notace O(f(N)), kde f značí použitý algoritmus a N velikost vstupních dat. Výsledkem asymptotické složitosti je maximální počet operací, které je potřeba provést. V databázovém světě existuje několik typických operací, které mají podle Niemanna následující časovou složitost (Niemann, 2008):

- O(1) – vyhledání v hash tabulce v konstantním čase,
- O(log(n)) – vyhledání ve vyváženém stromu,
- O(n) – vyhledání v nesetříděném poli,
- O(n*log(n)) – časová náročnost nejlepšího třídícího algoritmu,
- $O(n^2)$ – časová složitost špatného třídícího algoritmu.

V následující tabulce jsou uvedeny časové složitosti jednotlivých algoritmů v závislosti na velikosti vstupních dat. Z této tabulky plyne, že použitý algoritmus je z časového hlediska důležitý až pro velká vstupní data. Při malé velikosti vstupních dat, kdy jsou dnešní počítače schopny provést stovky milionů operací za vteřinu, je rychlost odezvy z pohledu uživatele takřka okamžitá. Z pohledu výkonu databázových systémů není jediným hardwarovým prvkem, který ovlivní výkon, pouze procesor, ale hlavně vstupně výstupní zařízení, jako jsou úložiště dat v podobě pevných disků či operační paměti počítače. Následující tabulka zdůrazňuje, jak je důležitá volba správného algoritmu s ohledem na velikost vstupních dat.

Algoritmus (počet operací) / Velikost dat (milionů)	1	10	100	1 000
O(log(n))	20	23	27	30
O(n)	1 000 000	10 000 000	100 000 000	1 000 000 000
O(n*log(n))	19 931 569	232 534 967	2 657 542 476	29 897 352 854
$O(n^2)$	1 000 000[1)]	100 000 000[1)]	10 000 000 000[1)]	1 000 000 000 000[1)]

Tabulka 1: Asymptotická složitost algoritmů (vlastní zpracování)

[1)] Miliony operací

3.4 Teoretické základy pro třídění, hledání a spojování dat

Z pohledu databázového systému se s daty pracuje jako s relacemi, nad kterými se provádějí různé operace. Pro ně je potřeba zvolit vhodný algoritmus tak, aby jeho zpracování bylo co nejefektivnější. Z pohledu práce s daty databáze jsou prováděny tři nejdůležitější operace, jako jsou vyhledání, třídění a spojování. Existuje mnoho postupů (algoritmů), jak je provádět co nejefektivněji. Je také důležité, jakým způsobem, neboli v jaké struktuře, jsou data uložena. V této kapitole jsou popsány teoretické základy těchto struktur a algoritmů používané ve světě databázových systémů.

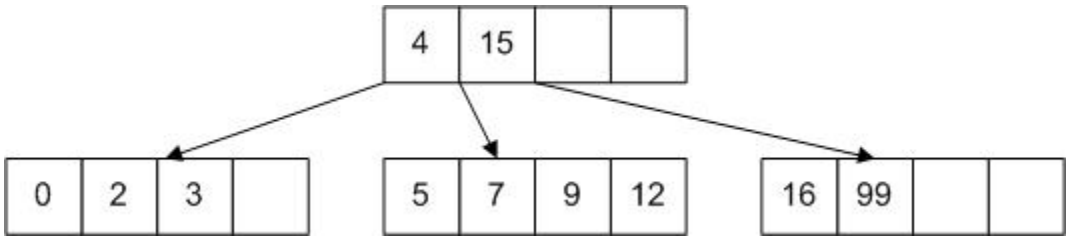
3.4.1 Binární vyhledávací strom

V případě databází je pro vyhledávání v datech často využívaná struktura nazývaná databázový index. Tato struktura je založena na binárním vyhledávacím stromu, z něhož vychází B-strom, drobná modifikace nazývaná B+ strom je pak základem struktury uložení dat v databázovém indexu. Definice B-stromu řádu n podle Bayera je následující (Bayer, 2002):

- každá cesta od kořene k jakémukoliv listu má stejnou vzdálenost h, nazývanou výška stromu T,

- každý vrchol, kromě kořene a listů, má minimálně k + 1 potomků, kořen je list nebo má minimálně dva syny,
- každý vrchol má minimálně 2k + 1 potomků.

Na následujícím obrázku je vidět jednoduchý B-strom řádu 5. Jsou zde vidět důležité vlastnosti stromu, tedy jeho stejná výška pro všechny listy a také uchování hodnot v setříděné podobě. Proto je tento strom označován také jako vyvážený.



Obrázek 1: B-strom - vlastní zpracování podle (Bayer, 2002)

Obecně se v rámci databázového systému řeší tři operace se strukturami udržujícími data. Jsou to operace hledání, vložení a smazání. Způsob, jakým jsou tyto operace prováděny, se liší od struktur, ve kterých jsou data uchovávána. Jak je patrno již z obrázku, v případě B-stromu, tedy databázového indexu, je operace vyhledávání velice efektivní. Nicméně další operace, jako jsou vložení nebo smazání, už budou o něco složitější, neboť je nutné po jejich dokončení splnit podmínku vyváženého stromu. Tady by byla vhodnější například struktura pole. Další důležitou operací je úprava dat. Tu je možné si představit jako po sobě jdoucí operace smazání a vložení prvku. V následující části jsou popsány jednotlivé operace vyhledání, vložení a smazání prvku v B-stromu (Cormen, Leiserson, Rivest, & Stein, 2009)

3.4.1.1 Vyhledání prvku v B-stromu

Vyhledání prvku v B-stromu je poměrně jednoduché. Začne se v kořenu stromu a postupně se porovnává hledaná hodnota s hodnotami ve vrcholu. Během porovnání mohou nastat následující situace.

- Hledaná hodnota je nalezena přímo ve vrcholu, potom hledání úspěšně končí s nalezenou shodou.

- V průběhu porovnávání je ve vrcholu nalezena hodnota, která je větší než hledaná. Pokud je vrchol list, končí hledání neúspěšně, pokud není, pokračuje hledání v levém podstromě.
- Hledání došlo až na konec vrcholu a hodnota nebyla nalezena. Pokud je vrchol list, končí hledání neúspěchem, pokud není, pokračuje hledáním v pravém podstromě.

3.4.1.2 Vložení prvku do B-stromu

Vložení nového prvku probíhá tak, že se hodnota vkládá až do listu. Nesmí ale dojít k porušení maximálního počtu hodnot ve vrcholu. Nejprve je tedy nalezen list, kam se má nový prvek uložit, pokud je zde volné místo, prvek se do listu uloží a operace je úspěšně dokončena. Pokud ale již volné místo není, vrchol je rozdělen na dva tak, že mezi hodnotami je nalezen medián. Hodnota mediánu je umístěna do rodiče původního vrcholu, pod něj jsou následně navázány dva vzniklé vrcholy. Zařazením mediánu do rodičovského vrcholu může opět dojít k problému s maximálním počtem hodnot. Tato situace je řešena stejným způsobem. Důležité je, že po těchto operacích je stále udržena struktura B-stromu.

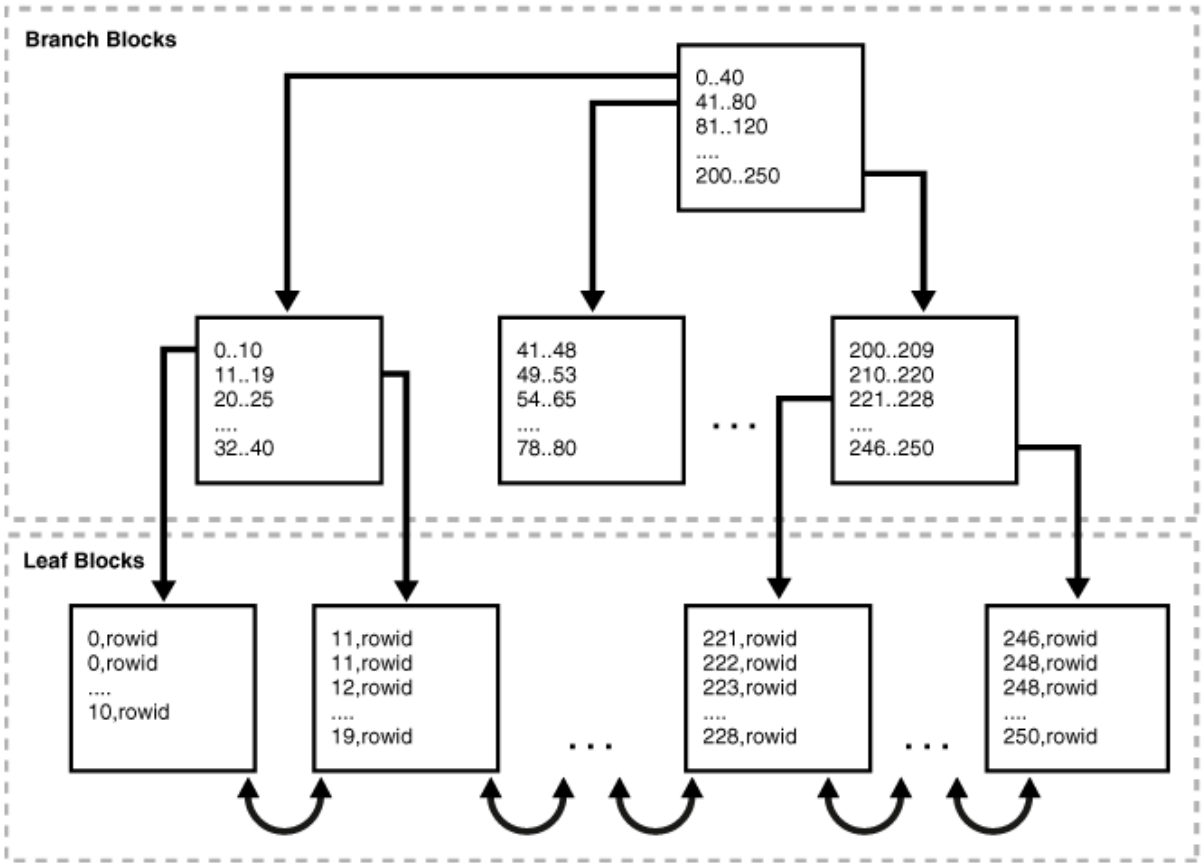
3.4.1.3 Smazání prvku z B-stromu

Poslední operací je smazání prvku. Prvek musí být nejprve nalezen ve stromě. Můžou nastat dvě situace, prvek je nalezen ve vrcholu nebo v listu. Pokud je nalezen v listu, prvek je jednoduše smazán. Může také nastat situace, kdy je porušena struktura B-stromu, tedy, že vrchol obsahuje méně hodnot, než je povoleno. Pokud k této situaci dojde, je potřeba B-strom přeorganizovat tak, aby splňoval podmínky B-stromu. Toto je dosaženo tím, že je nalezen sousední list s minimálním počtem hodnot. Tyto dva listy jsou spojeny, a ještě je k nim přidána hodnota z rodiče, která existovala mezi těmito dvěma spojenými listy. Takto vzniklý list je zařazen pod rodiče tak, aby splňoval podmínky B-stromu. Samozřejmě může nastat situace, kdy modifikovaný rodičovský vrchol porušil podmínku o minimálním počtu hodnot. Tato situace se řeší stejně do té doby, než je dosaženo korektní struktury B-stromu. Pokud není nalezen sousední list s minimálním počtem hodnot, je potřeba vybrat jednoho z nich. Z něj je vybrána nejkrajnější hodnota, ta je přesunuta do rodičovského vrcholu a hodnota z rodičovského vrcholu je přesunuta do listu, kde byla porušena podmínka o minimálním počtu hodnot. Mazání prvku z vnitřního vrcholu se provádí tak, že je nalezena

nejbližší hodnota ze syna, ta je zařazena místo smazané hodnoty. Tento způsob je vlastně převeden na mazání prvku z listu.

Časová složitost jednotlivých operací je dána výškou stromu h = O(log n), kdy n je počet hodnot uložených ve stromě. Časová složitost hledání v každém vrcholu je O(t). Celková složitost pro všechny operace je $O(t \log_t n)$ (Cormen, Leiserson, Rivest, & Stein, 2009).

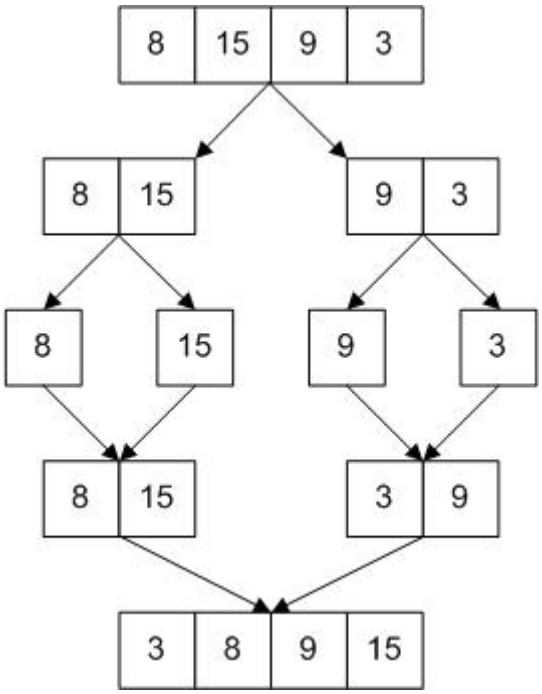
Databázový index převzal strukturu a pravidla B-stromu a s lehkými modifikacemi se využívá jako struktura pro uložení dat. Takto modifikovaný B-strom se nazývá B+strom. Zmíněné modifikace spočívají v tom, že vlastní data jsou uložena pouze v listech, může se tedy stát, že stejná hodnota se nachází jak ve vnitřním vrcholu, tak i v listu. Druhá důležitá modifikace je v tom, že listy stromu jsou mezi sebou spojeny tak, že každý list má vazbu na přímo vedle sousedící list. Tato modifikace je důležitá při hledání v nějakém rozsahu, kdy není potřeba se vracet zpět v hierarchii stromu nahoru, tzv. „Range index scan“. Stačí nalézt nejmenší hodnotu z dotazu v listech a pak postupně přecházet pomocí zmíněné vazby na další list stromu až do maximální velikosti hodnoty v dotazu. V listech B+stromu jsou uloženy hodnoty spolu s unikátním číslem řádky v tabulce, podle které je možné přímo identifikovat záznam v tabulce databáze, respektive její fyzické umístění. Na následujícím obrázku je naznačena konkrétní implementace databázového indexu pomocí struktury B+stromu v prostředí databázového systému společnosti Oracle.



Obrázek 2: Struktura databázového indexu B+strom, zdroj: (Oracle, 2016)

3.4.2 Merge sort

Merge sort je jednoduchý algoritmus většinově využívaný v databázových systémech pro setřídění dat. K setřídění používá datovou strukturu pole. Samozřejmě existuje spousta dalších třídících algoritmů, ale tento typ je vzhledem ke své rychlosti nejpoužívanější. Vychází z jednoduché logiky, že setřídit jednoprvkové pole není složité. Tento algoritmus tedy postupnými kroky rozdělí pole na n jednoprvkových polí. Následně jsou porovnány vždy dvě sousední pole tak, že se postupně porovnávají prvky z každého pole a tím vznikne nové setříděné pole. Takto se postupuje do doby, až je zpět složeno celé pole, které už je setříděné (Sherrod, 2007). Časová složitost se skládá ze dvou hlavních částí, zaprvé z rozložení celého pole na jednoprvková pole a následné porovnávání vzniklých polí. Z tohoto plyne, že celková časová složitost tohoto algoritmu je O(n*log(n)) (Sherrod, 2007). Na následujícím obrázku je vidět postup setřídění malého pole pomocí algoritmu merge sort.



Obrázek 3: Merge sort (vlastní zpracování podle (Sherrod, 2007))

3.4.3 Spojování

V relačních databázových systémech dochází velmi často při práci s daty k situacím, kdy je potřeba spojit dvě relace. Z následujícího popisu plyne, že spojování relací je nejnáročnější operace, a proto bude mít zřejmě největší vliv na výkon při zpracování uživatelského požadavku na vrácení určité množiny informací. V databázích se používají typicky tři různé způsoby spojení dvou relací. Jedná se o spojení pomocí nested loop (česky by se dalo přeložit jako vnořená smyčka), merge join a hash join. Každý z těchto typů spojení má výhodu v jiném typu dotazu a při zpracování uživatelských požadavků na data jsou vyhodnocovány všechny typy spojení, aby byl následně vybrán ten nejoptimálnější pro konkrétní případ. V každém typu spojení se pracuje vždy se dvěma relacemi, z nichž jedna je nazývána jako inner, česky vnitřní a outter jako vnější.

Vnořená smyčka

Jak již z názvu tohoto typu spojení vyplývá, jedná se o určitou smyčku, kdy jsou data postupně čtena z vnější relace a jsou porovnávána s daty z vnitřní relace. Pokud jsou si rovny, jsou tyto záznamy zařazeny do výsledné relace (Rahayu, Leung, & Taniar, 2008). V tomto případě je nutné přečíst záznamy z vnitřní relace tolikrát, kolik je záznamů ve vnější

relaci. Časová složitost tohoto spojení je tedy O(n * m) (Rahayu, Leung, & Taniar, 2008), kdy n je počet záznamů vnější relace a m počet záznamů vnitřní relace.

Merge join

V tomto případě se při spojování pracuje se setříděnými relacemi. Prvním krokem při spojení dvou relací je setřídění podle klíče, pomocí kterého jsou spojovány. Následuje spojení obou relací (Rahayu, Leung, & Taniar, 2008). Tento způsob spojení má výhodu v tom, že není nutné pro každý načtený záznam z vnější relace skenovat celou vnitřní relaci. Díky setříděnému seznamu je možné určit pozici, odkud je potřeba skenovat vnitřní relaci a nehledá se v oblasti, která již byla prohledávána. Opět pokud jsou nalezeny totožné hodnoty, jsou tyto záznamy zařazeny do výsledné relace. Časová náročnost tohoto spojení je O(n + m) v případě, kdy byly obě relace již setříděné. Pokud nebyly, je nutné je nejdříve setřídit, a to zvyšuje časovou náročnost na o O(n*log(n) + m*log(m)) (Rahayu, Leung, & Taniar, 2008).

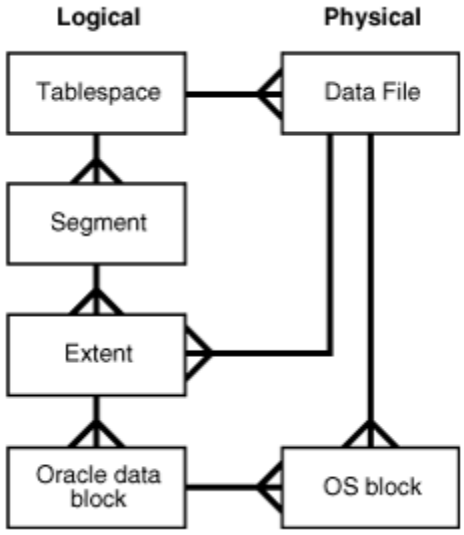
Hash join

V třetím případě je nejprve potřeba zvolit správnou hashovací funkci. Takováto hashovací funkce znamená, že po spočtení hash pro jednotlivé záznamy jsou výsledkem unikátní hodnoty nebo hodnoty s minimálním počtem opakování. Hodnoty hash jsou počítány z klíče, podle kterého byly obě relace spojeny. Na každý záznam ve vnitřní relaci je spočítána hash a výsledky jsou uloženy do hashovací tabulky. Následně jsou čtena data z vnější relace, je opět spočítána hash a ta je porovnána s hodnotami v hashovací tabulce (Rahayu, Leung, & Taniar, 2008). Pokud je hodnota nalezena, pak jsou záznamy, nad kterými byla spočítána shodná hash, zařazeny do výsledné relace. V případě, že je zvolena dobrá hashovací funkce, je časová složitost tohoto spojení O(n + m) (Rahayu, Leung, & Taniar, 2008).

3.5 Fyzické ukládání dat databáze

Vzhledem k tomu, že veškerá data databáze musí být uložena do trvalého úložiště, je potřeba znát, jakým způsobem jsou data ukládána, neboť i tento faktor má určitý vliv na výkon databázového systému. Každý databázový výrobce používá pro pojmenování jednotlivých komponent úložiště dat databáze jiné pojmy, nicméně princip je vesměs stejný. Například společnost Oracle ve svém databázovém systému používá pro ukládání

jednotlivých záznamů v databázové tabulce „data block“ (Oracle, 2015), zatímco společnost IBM používá ve svém databázovém systému označení „data pages“. Pro popis systému ukládání dat byl využit materiál společnosti Oracle (Oracle, 2015). Každý databázový systém ukládá svá data do datových bloků. V případě, že je potřeba přečíst data z databáze, je nutné přečíst celý datový blok bez ohledu na to, jestli jsou všechna data z bloku potřeba. Velikost datového bloku je tedy důležitá pro správný návrh databázového systému. Menší velikost datového bloku je vhodná pro malé řádky s náhodným přístupem. Možné velikosti datového bloku jsou pro databázový systém Oracle 12.1 2kB, 4kB, 8kB, 16kB a 32kB (Oracle, 2016). Tato velikost se definuje při tvorbě tablespace, tedy fyzického úložiště dat databáze. Datové bloky definované v databázovém systému nemají žádnou vazbu na bloky souborového systému použitého operačního systému. Na následujícím obrázku je vidět vazba mezi logickou strukturou úložiště v databázovém systému na rozložení dat na souborový systém.



Obrázek 4: Logická struktura úložiště versus fyzická, zdroj: (Oracle, 2015)

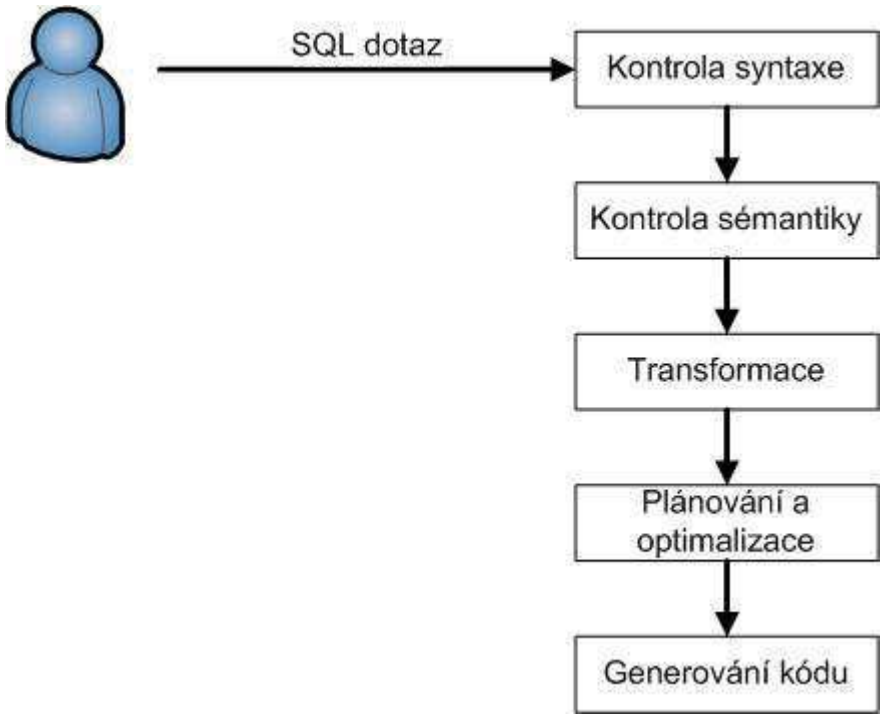
Jak je vidět z obrázku výše, datové bloky jsou alokovány pomocí takzvaných extentů (počet je možné zvolit při tvorbě tablespace) a ty jsou následně přiřazeny do segmentu. Každý objekt v databázi (tabulka, index atd.) má pro sebe alokovaný právě jeden segment, který je určen pouze pro data konkrétního objektu.

Při návrhu databáze je důležité vědět, s jakými daty (hlavně z pohledu velikosti záznamu) a jak se bude pracovat. Z dostupné literatury jednotlivých výrobců se dozvíme, že

pro OLTP aplikace se hodí datové bloky s menší velikostí, zatímco pro OLAP aplikace, kde se předpokládá práce s velkým množstvím dat, se hodí větší datové bloky (Oracle, 2015). Jednotlivé záznamy jsou ukládány do data bloků a teprve ve chvíli, kdy již není dostatek volného místa v aktuálním datovém bloku, použije se další. Tyto bloky jsou pak čteny při práci s daty databáze. V případě provozních aplikací se předpokládá, že se vrací většinou menší počet záznamů a není potřeba číst tolik datových bloků, kdežto v případě aplikací datových skladů se počítá s prací nad celým obsahem databáze. Proto tedy výše zmíněné doporučení. Důležité také je při návrhu databáze pamatovat na to, aby nedocházelo k situacím, kdy se záznam (řádek v tabulce) nevejde celý do jednoho datového bloku a musí být rozdělen do více datových bloků. Proto je důležité vědět, jaká data budou do databáze ukládána a podle toho také zvolit velikost datového bloku. Posledním důležitým aspektem při volbě velikosti datového bloku je to, že datový blok nemůže být využit až do maximální výše své kapacity, neboť každý datový blok obsahuje hlavičku s informacemi důležitými pro jeho správu. Tato hlavička může zabírat místo od 84 bytů až po 107 (Oracle, 2015).

3.6 Kroky zpracování SQL dotazu

Proces zpracování SQL dotazu prochází několika kroky od kontroly správnosti až po spuštění kódu nad databází, který zajistí vrácení výsledku dotazu zpět uživateli (Pokorný & Valenta, 2013). Na následujícím obrázku je znázorněn celý proces v postupných krocích. Některé kroky je možné za splnění určitých podmínek vynechat.



Obrázek 5: Proces zpracování SQL dotazu (Pokorný & Valenta, 2013)

V prvním kroku je provedena kontrola syntaxe, tedy jestli není chyba v SQL příkazech. Pokud tato validace projde, je provedena kontrola sémantiky. Ta má za úkol provést kontrolu existence všech objektů, které se v dotazu nacházejí. Následující krok je důležitý, neboť je v něm mimo jiné provedena kontrola na schodu s dříve provedenými SQL dotazy v rezervované paměti databáze. Pokud je takovýto dotaz nalezen, je možné přeskočit krok optimalizace a využít informací z dříve provedených dotazů. Tyto informace obsahují mimo jiné dříve vyhodnocenou přístupovou cestu pro zpracování dotazu. Tím odpadá mnohdy časově náročný krok v procesu zpracování SQL dotazu a tím je tvorba přístupového plánu a optimalizace. Celý proces hledání nejoptimálnější cesty je popsán v následující kapitole Vyhodnocení optimální cesty pro zpracování SQL dotazu.

3.7 Vyhodnocení optimální cesty pro zpracování SQL dotazu

V průběhu zpracování SQL dotazu je jedním z nejdůležitějších kroků výběr vhodného přístupového plánu pro konkrétní dotaz. Výsledkem tohoto vyhodnocení je určení cesty (dotazovacího plánu), kterou databázový systém použije pro získání dat z databáze. Na správné volbě cesty závisí rychlost, jakou je uživateli nebo aplikaci vrácen výsledek dotazu.

O vyhodnocení se stará zvláštní komponenta databázového systému, která se nazývá „Query optimizer“ (česky volně přeloženo jako vyhodnocovač dotazů). Tato komponenta není přímo přístupná koncovému uživateli, ale je automaticky spouštěna databázovým systémem při každém dotazu do databáze. Jedinou možností, jakou uživatel může ovlivnit funkci vyhodnocovače dotazů, je pomocí tzv. hintů, což je jednoduše řečeno vnucení způsobu zpracování dotazu uživatelem a obejití automatického vyhodnocení databázovým systémem. Použití hintů se doporučuje až v situacích, kdy evidentně databázový systém nevyhodnotí správně cestu zpracování dotazu. K vyhodnocení dotazu a stanovení dotazovacího plánu nemusí docházet vždy pro každý dotaz. Existuje totiž dedikovaná část paměti pro udržování těchto dotazovacích plánů a databázový systém může pro identické dotazy využít informací z této paměti a tím zkrátit čas zpracování. Důležitou podmínkou pro správné fungování vyhodnocovače dotazů je aktuální verze databázových statistik, na základě kterých jsou právě vyhodnocovány nejoptimálnější varianty zpracování dotazu. Proto je jednou z důležitých podmínek pro poskytování výkonného databázového systému udržování aktuálních databázových statistik.

Obecný postup vyhodnocení dotazovacího plánu byl popsán týmem odborníků z IBM Research Division již v roce 1973 v dokumentu „Access Path Selection in a Relational Datase Management System“ (Selinger, Astrahan, Chamberlin, Lorie, & Price, 1973).

Dotaz do databáze je po kontrole databázovým systémem na syntaktickou správnost dotazu předán vyhodnocovači dotazů pro stanovení dotazovacího plánu. Prvním úkolem této komponenty je získání informací o všech tabulkách (relacích) a sloupcích tabulek uvedených v dotazu, které budou později použity k vytvoření dotazovacího plánu. Tyto informace jsou získány jednak ze systémového katalogu databázového systému, tak hlavně z databázových statistik. Nejlépe vyhodnocený dotazovací plán je následně uložen v speciálním jazyku „Access Specification Language“ a pomocí generátoru kódu transformován do strojového jazyka a předán k exekuci (Selinger, Astrahan, Chamberlin, Lorie, & Price, 1973). V průběhu exekuce jsou data čtena (skenována) na základě dotazovacího plánu, a to buď přímo z datových struktur (segment scan) nebo struktur databázového indexu (index scan). V průběhu čtení jsou data vyhodnocována na základě podmínek uvedených v dotazu, tzv. predikátů. V případě, že načtená data neodpovídají podmínce v dotazu, jsou tato zahozena a nejsou dále předávána a tím nezatěžují celý systém. Data v databázi jsou ukládána do

datových bloků neboli stránek (pages) a jedním z určujících faktorů vhodného dotazovacího plánu je právě počet těchto stránek, které musí být načteny. Je potřeba říct, že v případě čtení dat pomocí segment scanu je vždy potřeba načíst celou stránku, nehledě na to, zda jsou data do výsledku potřeba nebo ne.

Úkolem vyhodnocovače dotazů je stanovení všech možných cest pro získání dat jako výsledek dotazu a z nich vybrat tu nejoptimálnější. Ke stanovení složitosti vyhodnocení dotazu se stanovuje takzvaná cena dotazu, která je dána vzorcem $cost = page\ fetches + W * RSI\ calls$ (Selinger, Astrahan, Chamberlin, Lorie, & Price, 1973), neboli součet všech načtených stránek a počet vrácených výsledků (řádků z tabulek) odpovídajících podmínkám dotazu. Písmeno W představuje vyrovnávací faktor mezi I/O operacemi čtení z disku a CPU. Pro rozhodnutí o optimálním dotazovacím plánu využívá vyhodnocovač dotazů následujících informací získaných ze statistik. Pro každou relaci (Selinger, Astrahan, Chamberlin, Lorie, & Price, 1973) sledujeme:

- kardinalitu, neboli počet unikátních hodnot relace (NCARD),
- počet stránek v segmentu, které obsahují data pro konkrétní relaci (TCARD),
- poměr stránek v segmentu použitých pro konkrétní relace vůči počtu neprázdných stránek v daném segmentu (P).

Pro každý databázový index jsou pak zjišťovány následující informace:

- počet unikátních klíčů v indexu (ICARD),
- počet stránek indexu (NINDX).

Pomocí těchto údajů vyhodnocovač dotazů stanoví tzv. selektivitu F pro každý predikát v dotazu. Tuto hodnotu je možné si představit jako poměr vrácených řádků tabulky vůči celkovému počtu záznamů pro konkrétní podmínku danou predikátem. Tato hodnota se tedy nachází v intervalu (0;1>. Pro jednoduchý dotaz skládající se z jedné tabulky (relace) je dotazovací plán stanoven na základě těchto selektivit faktorů spolu se statistikami všech možných přístupových cest k datům a indexům. Pro takovýto dotaz je nejoptimálnější (nejlevnější) cesta zvolena pomocí porovnání cen všech dostupných cest, tedy vyhodnocení přístupu pomocí všech indexů nad tabulkou v dotazu a skenu všech datových stránek segmentu tabulky. V následující tabulce jsou uvedeny ceny dotazů pro různé situace.

Situace	Cena dotazu (počet stránek)
Unique index	1 + 1 + W
Clustered index obsahující jeden nebo více sloupců z WHERE podmínky dotazu	F[1)] * (NINDX + TCARD) + W*RSICARD[2)]
Non-clustered index obsahující jeden nebo více sloupců z WHERE podmínky dotazu	F[1)] *(NINDX + NCARD) + W*RSICARD[2)]
Clustered index neobsahující jeden nebo více sloupců z WHERE podmínky dotazu	(NINDX + TCARD) + W*RSICARD[2)]
Non-clustered index neobsahující jeden nebo více sloupců z WHERE podmínky dotazu	(NINDX + NCARD) + W*RSICARD[2)]
Segment scan	TCARD/P[2)] + W*RSICARD[1)]

Tabulka 2: Cena dotazu pro různé situace skenování (Selinger, Astrahan, Chamberlin, Lorie, & Price, 1973)

[1)] Celková selektivita pro WHERE podmínku

[2)] Očekávaný počet RSI volání

Dotazy do databáze mohou být samozřejmě složitější a to zejména z hlediska počtu tabulek (relací), které jsou v dotazu zahrnuty. Stanovení optimální cesty v tomto případě je již trochu složitější, ale vychází z výše popsaných základů. Takovýto dotaz je většinou určen klíčovým slovem JOIN (česky spojení) ve WHERE podmínce. Parametrem klíčového slova JOIN je takzvaný join predicate (predikát spojení), který určuje vazbu mezi dvěma relacemi. V případě spojování dvou relací vystupuje každá z nich v různé roli. První je označena jako OUTTER (vnější) relace a druhá jako INNER (vnitřní). V průběhu zpracování dotazu jsou pak nejdřív načteny výsledky z vnější relace a v dalším kroku jsou k nim hledány postupně odpovídající záznamy z vnitřní relace. Prvním úkolem při spojování je potřeba zvolit způsob, jakým dvě relace spojit. Možné způsoby spojení byly popsány v kapitole Teoretické základy pro třídění, hledání a spojování dat. Dalším úkolem při vyhodnocení spojení více relací je určení pořadí, v jakém dojde ke spojení. Je totiž důležité vzít do úvahy, že i když je výsledek dotazu přes více relací nezávislý na pořadí spojování relací, výsledná cena pro různá pořadí spojení se může dramaticky lišit. Počet způsobů, jak je možné zvolit pořadí spojení relací, je roven faktoriálu počtu relací obsažených ve WHERE podmínce. Na

vícenásobné spojení je možno nahlížet jako na postupné spojování relací k sobě, tedy ke vzniklé relaci (můžeme ji nazvat jako dočasná kompozitní relace vzniklá ze dvou relací uvedených ve WHERE podmínce) je připojena další relace definovaná v dotazu. Ke spojení více relací je tedy možné využít stejný princip spojení jako na dvě relace. Prvním krokem v určení pořadí spojování relací je snaha o nalezení relací, které mají společný klíč pro spojení uvedený v JOIN klauzuli. Je jasné, že spojení pomocí kartézského součinu má nejvyšší cenu a je použito až v případě, že jiná možnost spojení mezi relacemi není možná. Pro nalezení nejoptimálnějšího způsobu spojování relací je využito heuristických metod a existuje mnoho způsobů, jak k němu dojít. Poté, co je nalezeno optimální pořadí spojení relací, je postupně stavěn strom možných cest pro přístup k datům konkrétního dotazu. V postupných iteracích, dle stanoveného pořadí spojování, jsou vždy k nejoptimálnějším cestám dočasných relací postupně přidávány další relace dle stanoveného pořadí, až je vystaven celý strom všech možných přístupových cest, ze kterých je následně vybrána ta nejoptimálnější. Nejoptimálnější cesta je také vyhodnocována na konci každé iterace a do další iterace je předána pouze vybraná cesta. Na konci každé iterace je potřeba vyhodnotit tu nejoptimálnější cestu pro všechny možné kombinace spojených relací a setřídění. Je také důležité zmínit, že podmínkou spojení relace s kompozitní relací není dokončení spojení kompozitní relace. Dokončení spojení dočasné relace je nutné pouze v případě, že pro další spojení je nutné setřídění této dočasné relace.

Moderní databázové systémy poskytují nástroje, které umožní správci si zobrazit přístupový plán, který byl zvolen pro zpracování dotazu. Je možné si tyto informace zobrazit jak pro historické dotazy, tak i pro konkrétní dotaz, který je možné poskytnout jako vstup tomuto nástroji. Výstupem pak jsou informace jako počet I/O operací, doba zpracování každé části přístupové cesty, zvolený způsob spojování relací, počet řádků na výstupu každé části přístupové cesty atd. Na základě těchto informací může administrátor usoudit, zda je dotaz zpracováván optimálně nebo existuje ještě cesta, jak jej zefektivnit.

3.8 Databázové statistiky

Z pohledu výkonu databázového systému se jedná o jednu z klíčových částí databáze. Databázové statistiky poskytují informace o povaze dat uložených v databázových tabulkách či databázových indexech. Dle Powella jsou statistiky vypočtené nebo odhadnuté velikosti

o rozložení dat v tabulkách a indexech (Powell, 2011). Pro to, aby se v průběhu vyhodnocování dotazu mohl vyhodnocovač přístupového plánu správně rozhodnout a zvolit optimální přístupový plán, potřebuje informace o povaze dat, která jsou zahrnuta ve vyhledávacím požadavku. Tyto informace získá z databázového katalogu, kde jsou uložena statistická data o povaze záznamů. Zde je možné zjistit informace, jakými jsou počet řádek/stránek využívaných konkrétní tabulkou. Pro každý sloupec v tabulce jeho kardinalitu, průměrnou délku hodnot, maximální a minimální hodnotu a počet prázdných hodnot ve sloupci. Pro databázové indexy lze získat informace o kardinalitě, počtu listů indexového stromu, výšku indexového stromu a počty unikátních hodnot v indexu postupně pro různé sloupce v indexu.

Dalším typem statistik jsou histogramy. Data v jednotlivých sloupcích tabulky nebudou většinou rovnoměrně rozdělena, to znamená, že se například hodnota A ve sloupci tabulky vyskytuje v pěti procentech z celkového počtu záznamů, kdežto hodnota B v padesáti procentech. Na základě těchto informací je pak vyhodnocovač schopen volit různé přístupové cesty pro získání dat. Takováto analýza dat zabere určitý čas v závislosti na objemu dat, proto je potřeba sběr statistik pouštět v době méně vytíženého databázového systému. Důležité je, aby statistiky maximálně odpovídaly aktuální skladbě dat. Proto je také vhodné přepočet statistik pouštět v případě masivnější modifikace dat. Moderní databázové systémy již umožňují sběr statistik automaticky.

3.9 Statický SQL versus dynamický

Z pohledu zpracování přístupového plánu je rozdíl v případě využití statického SQL dotazu od dynamického. Rozdíl mezi těmito dvěma typy dotazů je v tom, že v případě statického SQL je známa jeho struktura včetně hodnot v podmínce, kdežto u dynamické verze je možnost SQL dotaz za běhu programu měnit v závislosti na interakci s uživatelem. V případě dynamického dotazu jsou použité takzvané pojmenované parametry, které jsou v dotazu uvedeny pomocí znaku otazníku. Jedná se pouze o zástupný znak, který bude za běhu programu nahrazen skutečnou hodnotou získanou z předchozích událostí. Následující jednoduchý statický SQL dotaz do tabulky zákazníků „SELECT firstname, surname FROM customer WHERE custId=123” může být nahrazen následujícím dynamickým SQL dotazem ve tvaru „SELECT firstname, surname from customer WHERE custId= ?”. Zástupný znak

otazníku může být následně nahrazen jakoukoliv hodnotou získanou z dosavadního běhu programu. Dynamický typ dotazu nejenže dodává programům větší flexibilitu, řeší bezpečnostní problémy typu SQL injection, ale hlavně může ovlivňovat výkon při zpracování dotazu. V případě statického SQL dotazu tento vždy unikátní a databázový systém při každém, ač strukturou identickém, dotazu znovu vyhodnocuje optimální přístupovou cestu, nehledě na to, jedná-li se o dotaz identický pouze s odlišnými hodnotami parametrů v podmínce. V případě dynamického dotazu moderní databázové systémy obsahují vyrovnávací paměť pro dynamické SQL dotazy, kde se uchovává dotaz společně s již vyhodnocenou přístupovou cestou. Odpadá tedy nutnost opětovného vyhodnocování přístupové cesty pro každý dotaz. Využití dynamických SQL může tedy přinést zlepšení ve zpracování dotazů, ale může mít i opačný efekt. V případě, že se data v databázi často mění, může se i měnit vhodnost různých přístupových cest. Je tedy nutné brát v potaz i toto úskalí a paměť s informacemi o uložených přístupových cestách pro dynamické dotazy pravidelně čistit.

3.10 Komprese

Databázové systémy udržují obrovské množství dat. K jejich uložení jsou stále využívané pevné disky jako úložiště s obrovskou kapacitou, kde je možné data trvale udržovat. V případě jakékoliv operace nad daty z databáze je potřeba k datům v úložišti přistoupit. To, jak rychle je možno k datům přistoupit, je dáno počtem I/O operací, které dané zařízení poskytuje, takzvanými IOPs (počet vstupně výstupních operací za sekundu). I když v současné době dostupné SSD disky poskytují mnohonásobně vyšší výkon, stále je část úložiště jednou z hlavních komponent ovlivňujících celkový výkon databázového systému. Jednou možností, jak tento problém částečně řešit, je komprese dat. Dle LoForta datová komprese přináší následující výhody (LoForte, Wort, & Jorgensen, 2012):

- lepší využití I/O operací, neboť více dat je čteno a zapisováno pomocí jedné datové stránky,
- lepší využití vyrovnávací paměti, neboť se do ní vejde více uživatelských dat,
- do datové stránky se vejde více uživatelských dat a tím je možné z jedné stránky získat více uživatelských dat.

V případě sekvenčního čtení komprimovaných dat v databázi se snižuje počet potřebných načtených bloků pro stejný objem dat z úložiště. Tímto dochází i ke snížení počtu I/O operací, které vede ke snížení času nutného k načtení dat. Na druhé straně je ale potřeba data při uložení komprimovat a při načítání dekomprimovat, což zvyšuje nároky na procesor databázového serveru a komprese nemusí být vhodná pro všechny případy. Vždy je tedy potřeba uvážit váhu zvýšení výkonu při přístupu k datům vůči váze zvýšení požadavku na výkon, zatížení procesoru.

3.11 Vyrovnávací paměť

Z předchozích kapitol je zřejmé, že velký vliv na celkový výkon databázového systému má načítání dat z externích úložišť, a tak možnost, jak eliminovat tento problém, je využití vyrovnávací paměti, která poskytuje mnohem větší výkon pro I/O operace. Je tedy snaha co nejvíce dat udržet v paměti serveru, než data neustále číst z externího zařízení, které je mnohonásobně pomalejší. V případě, kdy je potřeba přistoupit k datům, jsou požadovaná data nejdříve načtena do paměti a až následně se s nimi pracuje přímo v paměti. Zde se také objevuje pojem „Prefetching“ (předběžné načítání dat), což znamená, že databázový systém se snaží data načíst do paměti dříve, než budou potřeba. Je to možné z toho důvodu, že informace o přístupové cestě jsou již známy, a tedy i data, která bude potřeba načíst. V případě předběžného načítání jsou data z externího úložiště čtena více vlákny, aby se urychlilo jejich načtení. Výhoda využití vyrovnávací paměti je tedy v tom, že data jsou databázovým systémem přímo čtena z paměti, nikoliv z pevného disku. Oblast paměti určená jako vyrovnávací paměť udržuje kopie datových bloků v paměti, takže jsou blíže procesoru než v případě dat uložených na disku (Lui, 2011). V případě, kdy se již zde data nachází, není nutné znovu data číst z externího úložiště a odpadá tím časově náročná operace čtení z tohoto zařízení. Velikost této paměti je samozřejmě omezena a nemusí mít pozitivní vliv na výkon v případě, kdy je neustále přistupováno k různým datům. Data jsou zde spravována pomocí algoritmu Last recently used (LRU), kdy jsou v případě zaplnění odstraněna data, která jsou nejstarší, tedy mají nejvyšší rozdíl v čase posledního použití. Jak již bylo popsáno v předešlých kapitolách, data jsou z úložiště načítána po datových blocích/stránkách, nikoliv po řádcích tabulky. V paměti jsou tedy uchovávány celé bloky. Stejně tak jako v databázovém systému existuje paměť pro čtení dat, existuje také oblast pro

zápis dat. Při použití zapisovací paměti se zde data udržují a jsou v pravidelných cyklech a dávkách zapisovány na externí úložiště. Určujícím parametrem efektivity využití vyrovnávací paměti je parametr „hit ratio“. Ten udává procentuální poměr dat načtených rovnou z vyrovnávací paměti vůči celkovým operacím čtení dat. Pohybuje se v rozmezí 0-100 % a je snahou dosáhnout co nejvyššího čísla, neboť čím vyšší je tento parametr, tím více dat bylo čteno přímo z paměti a nebylo nutné je načítat pomocí časově náročnějších operací z pevných disků databázového serveru. Některé databázové systémy poskytují tuto hodnotu přímo na základě výpisu konkrétního příkazu, jiné poskytují nástroje, jak získat podkladová data, ze kterých se dá tento parametr vypočítat.

3.12 Tabulkový partitioning

Jednou z možností při návrhu fyzického modelu databáze je možnost využití tabulkového partitiononigu, neboli logického rozdělení tabulky, který může zlepšit nejen výkon databáze, ale i může usnadnit vlastní správu celého databázového systému. Pomocí partitioningu je možné rozdělit velkou databázovou tabulku na menší celky (Harrington, 2009), takzvané partition, podle určitého klíče. Tyto celky je pak možné spravovat buď společně, nebo jednotlivě. Každý jednotlivý celek tabulky může mít přidělen jiný typ úložiště. Mezi důležité vlastnosti partitioningu patří skutečnost, že na rozdíl od databázového indexu nemá další nároky na úložiště (partitionovaná tabulka zabere stejně místa jako nepartitionovaná) a také to, že z pohledu aplikace je vše transparentní a práce s takovouto tabulkou je z pohledu aplikace identická jako s tabulkou bez partitionongu. Existují dva druhy partitioningu, vertikální a horizontální. Vertikální partitioning rozděluje sloupce tabulky do více tabulek spojených primárním klíčem. Horizontální partitioning rozděluje řádky tabulky do více tabulek se shodnou strukturou (Harrington, 2009).

Příkladem horizontálního partitioningu může být třeba tabulka s objednávkami, ve které je v jednom sloupci uvedeno datum uskutečnění objednávky. Takovouto tabulku je možné rozdělit po měsících dle datumu objednávky. V případě dotazu do databáze na konkrétní objednávky jednoho měsíce je skenována pouze jedna partition, nikoliv celá tabulka. Všechny záznamy jsou podle konkrétního klíče jednoznačně rozmisťovány do různých partition. Nastavení kritérií pro rozdělení dat do jednotlivých partitition je velice

flexibilní a umožňuje rozdělení dat podle konkrétního rozmezí hodnot klíče, vyjmenovaných hodnot klíče nebo rozdělení pomocí hash funkce, která je aplikována na klíč.

3.13 Databázový index

V případě ladění výkonu databázového systému je databázový index nejčastěji používanou variantou pro snížení času nutného pro vrácení výsledku na dotaz do databáze. Podle Harringtona je indexování cesta k poskytnutí rychlé přístupové cesty k hodnotám sloupce nebo sloupcům tabulky (Harrington, 2009). Po prohledání indexu databázový systém vrací množinu unikátních identifikátorů řádků, kde jsou uložena požadovaná data. Tato jsou přečtena a vrácena zpět jako výsledek dotazu. Existuje více typů databázových indexů, respektive struktur, na kterých jsou založeny. Nejčastěji používaný index je založen na struktuře B stromu, která byla popsána v jedné z předcházejících kapitol. Nicméně existují i jiné varianty indexu, například bitmapový index, který je doporučován pro použití na datech, kde je nízká kardinalita. Vytvoření databázového indexu je většinou doporučováno na všech sloupcích tabulky, podle kterých se vyhledává, třídí, nebo spojují tabulky. Než administrátor databázového systému začne vytvářet indexy, je potřeba vzít do úvahy následující vlastnosti databázového indexu (Harrington, 2009):

- index zabírá diskový prostor v databázi,
- pokud jsou data přidávána, modifikována nebo mazána v indexovaném sloupci, je potřeba modifikovat i vlastní index,
- index rozhodně urychluje přístup k datům.

Databázový index tedy na jedné straně snižuje dobu odezvy na dotaz do databáze, na straně druhé zvyšuje čas na vložení, úpravu nebo smazání záznamu. Obě tyto skutečnosti včetně dodatečných požadavků na úložiště je potřeba uvažovat v případě fyzického návrhu databáze. Některé databázové systémy umožňují řešit částečně tento problém tím, že je vytvořen index, který není automaticky upravován při jakékoliv manipulaci s daty. Toto bývá využíváno v systémech, kde dochází k hromadnému nahrávání dat. V tomto případě jsou data nejdřív nahrána a až následně je index upraven dle změn v datech.

Databázové indexy je možné vytvářet jednoduché, tedy přes jeden sloupec tabulky nebo složené (kompozitní) přes více sloupců najednou. Někdy je možné se setkat s tzv.

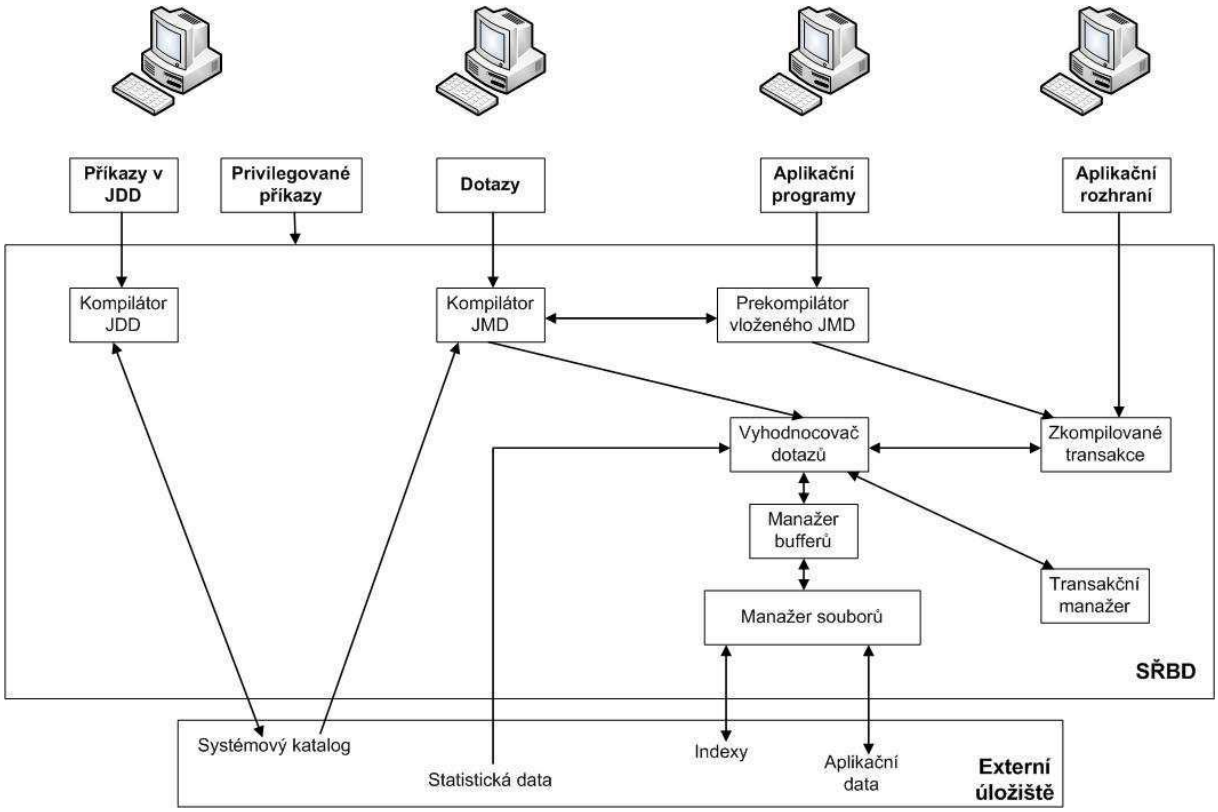
„covering“ indexem, který obsahuje všechny informace požadované v dotazu. V případě takovéhoto indexu není nutné pro data, která jsou požadována v dotazu, přistupovat do tabulky, přesněji řečeno číst datové bloky z úložiště, kde jsou uložena data tabulky, ale rovnou vrátit data načtená z indexu. Tento typ indexu má tedy velký vliv na výkon z pohledu vyhledávání v databázi. Další důležitou vlastností databázového indexu je také to, že musí být ze své podstaty setříděn a je tedy možno jej využít v situacích, kdy je potřeba pracovat se setříděnými daty.

Databázový index může existovat ve variantě non-clustered, nebo clustered. V prvním případě jsou data do tabulky, respektive datového bloku na disku, ukládána bez jakéhokoliv pořadí. Takováto struktura úložiště dat tabulky je nazývána heap-organized. V případě druhého typu indexu jsou data ukládána v pořadí daném klíčem indexu. Data, která leží logicky vedle sebe, jsou i uložena fyzicky vedle sebe v datovém bloku na disku a v případě požadavku na data ležící vedle sebe je snížen i počet nutných I/O operací.

Databázový index je tedy funkcionalita, která významně ovlivňuje celkový výkon databázového systému, a to pozitivně z pohledu získávání dat z databáze, a na druhé straně negativně při manipulaci s daty. Je tedy vždy potřeba najít správný kompromis pro různé situace.

3.14 Funkční architektura databázového systému

Databázový systém, stejně tak jako všechny počítačové programy, spoléhají na výkon hardware počítače, na kterém běží a ten určuje celkovou výkonnost systému. Důležitou součástí procesu ladění výkonu databázového systému je i optimální konfigurace hardware, na kterém tento systém poběží. Obecně platí, že výkon špatně navrženého databázového systému nevylepší sebevýkonnější hardware a naopak. Na následujícím obrázku je naznačena funkční architektura databázového systému včetně popisu jednotlivých komponent.



Obrázek 6: Funkční architektura DBS (Pokorný & Valenta, 2013)

Mezi nejdůležitější komponenty patří následující prvky (Pokorný & Valenta, 2013).

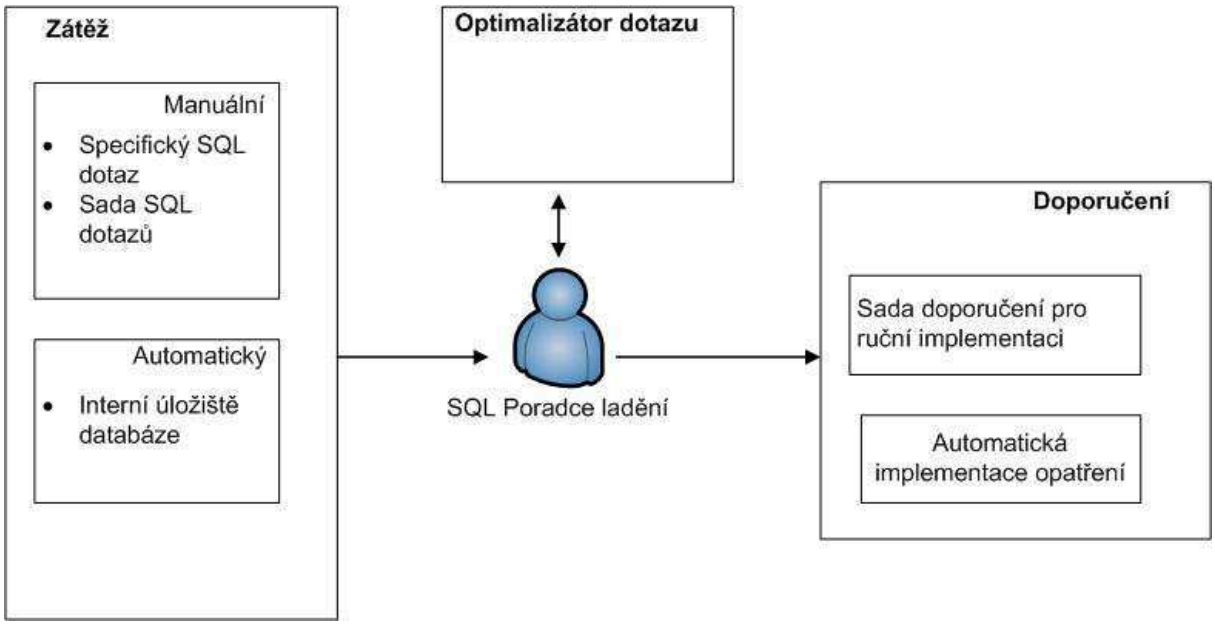
- Manažer souborů je vrstva mezi operačním systémem a daty databáze uložené na externím úložišti. Z tohoto je také patrné, že systém ukládání dat v databázi je oddělen od fyzického uložení dat.
- Manažer bufferů je komponenta spravující přidělenou paměť. Přes paměť prochází veškeré požadavky na čtení i zápis dat databáze. Je zde vidět, že data nejsou nikdy přímo zapisována na externí úložiště, ale je k tomu využita právě operační paměť z důvodu rychlosti.
- Kompilátor jazyka definice dat (JDD) zpracovává definici schématu databáze a ukládá ji do systémového katalogu. Tuto definici následně využívá komponenta kompilátoru jazyka modifikace dat (JMD).
- Kompilátor JMD – kompiluje požadavky aplikací na manipulaci daty v databázi.

Z obrázku je patrné, že výkon databázového systému bude ovlivněn zejména výkonem pevných disků, ze kterých je tvořeno externí úložiště a velikostí paměti neboli bufferu, kde

jsou uchovávána data použitá pro přístup k datům databáze. Čím rychlejší je externí úložiště, tím rychleji je z něj možné získat data. Výkon externího úložiště je většinou udáván v počtu I/O operací za sekundu, takzvanými IOPs. Velikost operační paměti přiřazená bufferům určuje množství dat, která je možné uložit v paměti, ve které jsou data spravována pomocí algoritmu LRU. Čím větší paměť, tím starší data jsou odstraňována z paměti a je tedy větší šance, že při dalším dotazu budou data poskytnuta rovnou z paměti a nebude nutno je číst z externího úložiště. Operační paměť není samozřejmě použita pouze pro data z JMD, ale i pro další účely, jakými je třeba prostor pro třídění dat. Důležitou komponentou je i vlastní procesor serveru, který provádí veškeré výpočetní operace požadované databázovým systémem. Důležitý je také výkon procesoru z pohledu paralelismu v databázovém systému.

3.15 SQL tunning advisor

Ladění výkonu jakéhokoliv databázového systému je poměrně náročný úkol a tak v podstatě všechny databázové systémy poskytují nástroj jakéhosi rádce s nastavením databázového systému pro co nejlepší výkon. Prvním, kdo takovýto nástroj implementoval do svého databázového systému, byla společnost Microsoft v roce 1998 ve verzi Microsoft SQL server 7.0 (Chaudhuri & Narasayya, 2007). Od té doby došlo ke spoustě vylepšení, aby úkol ladění výkonu databázového systému byl pro jejich administrátory co možná nejjednodušší. V průběhu zpracování jakéhokoliv dotazu do databáze komponenta vyhodnocovače dotazu vypracuje na základě dostupných informací optimální přístupový plán dle aktuálního fyzického designu databáze. Na tuto činnost má vyhodnocovač dotazovacího plánu poměrně málo času, neboť čas na vyhodnocení se samozřejmě započítává do celkového času zpracování dotazu. Proto existují nástroje, které provádějí hlubší analýzu vyhodnocení dotazovacího plánu. Databázový rádce pro ladění výkonu používá stejnou komponentu ve speciálním režimu ladění. V tomto režimu nejsou kladeny žádné zvláštní nároky na čas zpracování jako spíše nárok na nalezení nejoptimálnější varianty zpracování SQL dotazu. Na následujícím obrázku je schematicky naznačen proces fungování SQL poradce ladění výkonu.



Obrázek 7: SQL tunning advisor (vlastní zpracování podle (Oracle, 2016))

Celá myšlenka tohoto nástroje vychází z podstaty, že databázový systém je poměrně dynamický s možností výrazných změn struktury dat i požadavků na informace, které je potřeba z databáze získat. Je tedy zřejmé, že úkol ladění výkonu je neustálý proces. Vzhledem k tomu, že data v databázi mohou být rozličného charakteru, stejně tak jako požadavky na získání informací z databáze, je potřeba se na ladění výkonu databázového systému a následné změny do fyzického designu dívat z globálního hlediska. Není možné provést ladění databáze pouze pro jeden dotaz, neboť jeho vyladění může mít negativní dopad na ostatní dotazy. V případech ladění specifického dotazu je tedy minimálně vhodné provést validaci doporučených opatření z globálního pohledu.

Jak je vidět z obrázku, vstupem pro SQL poradce ladění výkonu může být jeden dotaz, sada dotazů, nebo může být využito interního úložiště databáze se seznamem všech SQL dotazů historicky prováděných nad databází. Tento vstup je označován anglickým slovem workload, neboli zátěž. Každý dotaz je následně zkoumán pomocí komponenty vyhodnocovače přístupového plánu mnohem detailněji než v průběhu vyhodnocení přístupového plánu v průběhu vlastního zpracování SQL dotazu. V tomto případě je analyzován současný fyzický design databáze, jsou vytvářeny hypotetické databázové indexy, zjišťovány rozličné druhy databázových statistik a další operace (Oracle, 2016). Výsledkem tohoto úkonu je seznam doporučení na úpravu fyzického designu databáze

včetně odhadovaného zlepšení v případě implementace těchto doporučení. Mezi takováto doporučení patří zejména informace o databázových indexech a statistikách nebo úprava tabulek pomocí partitioningu. Tyto změny je buď možné implementovat ručně databázovým administrátorem, nebo je nechat vyrobit automaticky.

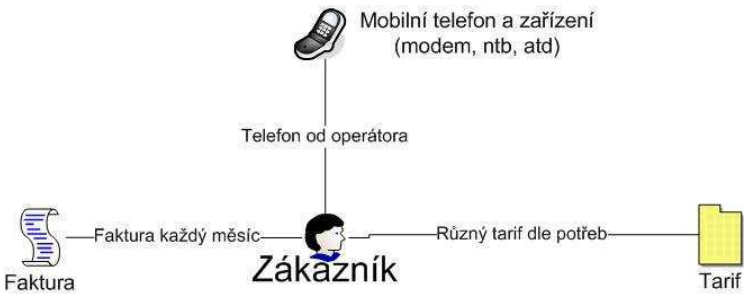
Tento nástroj je určen pro všechny správce databázových systémů, ať už se jedná o lidi s minimální praxí až po zkušené odborníky s mnohaletou praxí. Jednou z vlastností tohoto nástroje je i to, že administrátor si může ověřit dopad navrhovaných opatření bez toho, aby je skutečně implementoval. Mnoho současných databázových systémů umožňuje možnost spuštění této analýzy automaticky v době, kdy není systém vytížen a volný výkon serveru může být využit právě pro tyto analýzy.

4 Vlastní práce

V praktické části této práce byla provedena měření na relační databázi v oblastech popsaných v teoretické části.

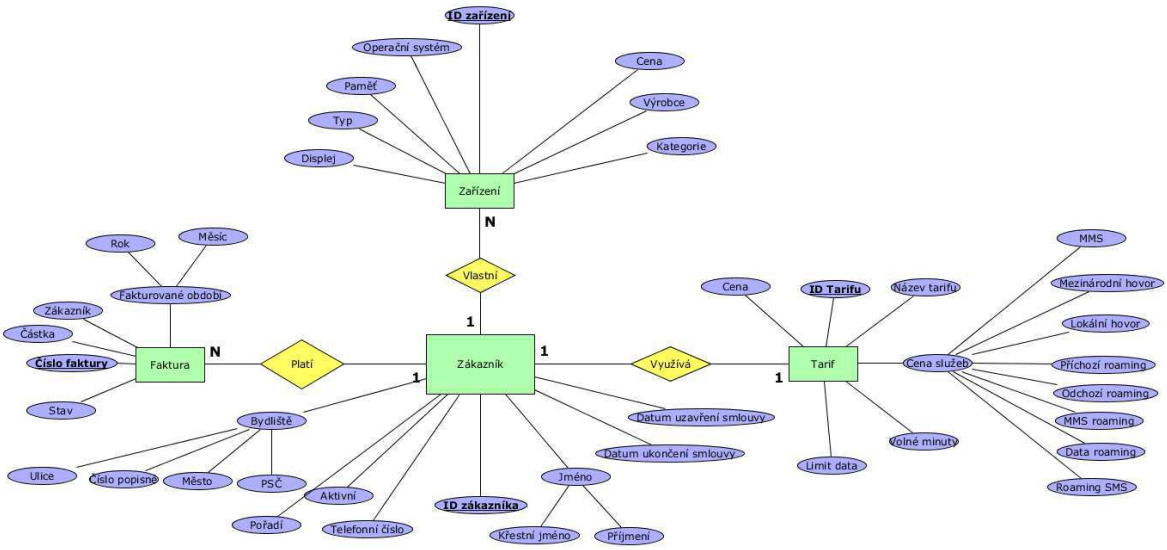
4.1 Návrh fiktivní relační databáze

Pro vlastní ověření teoretických podkladů z úvodní části této práce byla vytvořena databáze fiktivního telefonního operátora. Na následujícím obrázku je zobrazen konceptuální model fiktivní databáze. Nachází se v ní hlavní doména „Zákazník“, na kterou jsou navázané další domény, jakými jsou faktury za služby, přístroj(e), které zákazník obdržel od operátora a doména reprezentující zvolený tarif služeb.



Obrázek 8: Konceptuální model fiktivní databáze pro účely měření (vlastní zpracování)

Na následujícím obrázku je zobrazen model vztahů mezi jednotlivými entitami pomocí enterprise-relationship modelu (ERM). Je zde vidět, že každý zákazník může vlastnit více zařízení, vybírá si jeden tarif služeb a za ně mu jsou telefonním operátorem vydávány faktury.



Obrázek 9: Entity relationship model fiktivní databáze (vlastní zpracování)

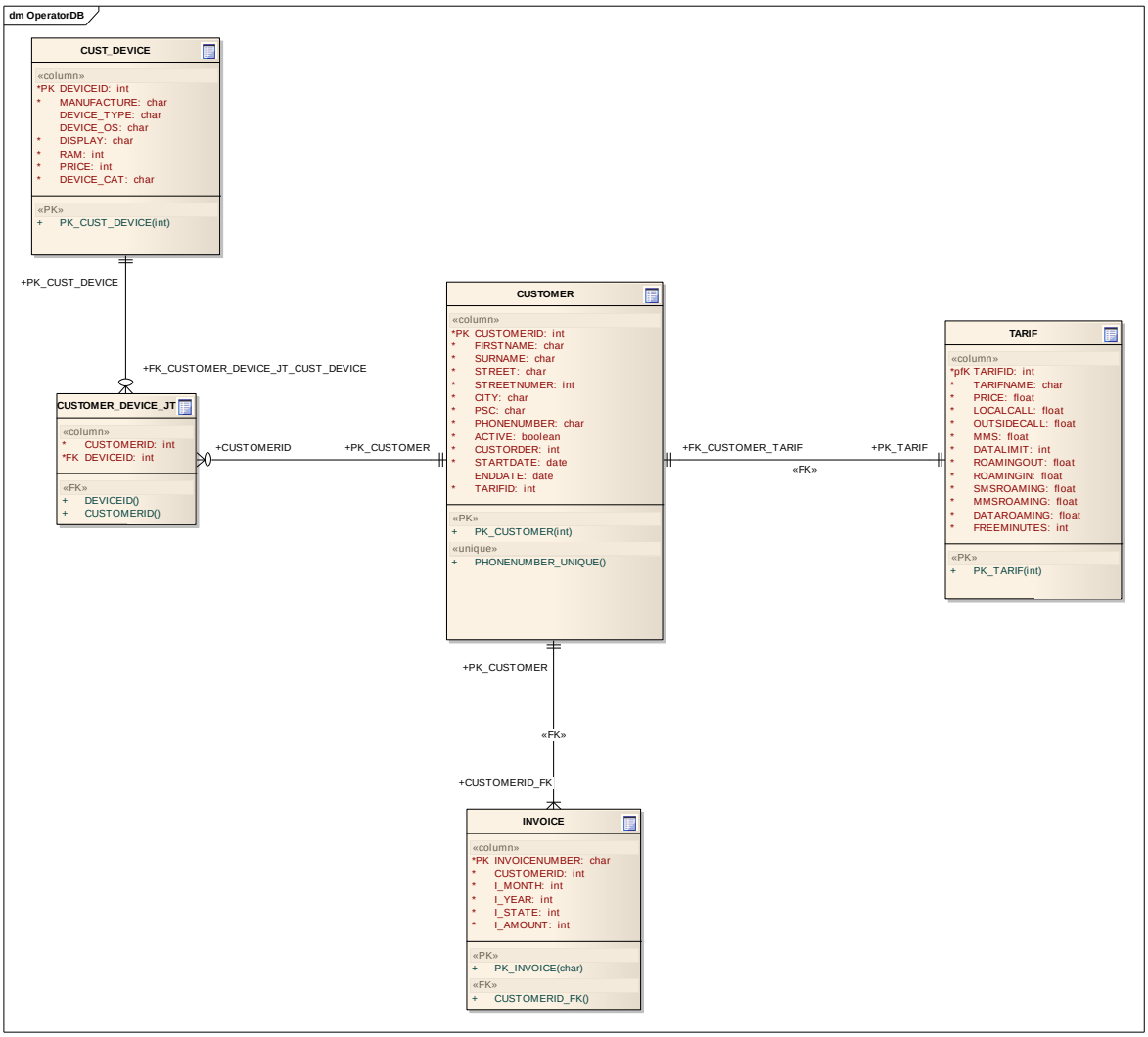
Detailnější popis jednotlivých databázových tabulek a vazeb mezi nimi v rámci logického datového modelu je uveden v přílohách (Příloha 1). Zde nejsou zmíněny databázové indexy, neboť ty byly dle potřeb vyráběny až v rámci vlastního měření.

V následující tabulce je uveden celkový počet záznamů vygenerovaných do jednotlivých databázových tabulek. Průměrná velikost řádky byla získána z databáze Oracle pomocí dotazu „SELECT avg_row_len FROM dba_tables WHERE table_name = '%název_tabulky%'“ (Oracle, 2016)

Název tabulky	**Počet záznamů**	**Průměrná velikost řádky (byte)**
CUSTOMER	13 600 000	96 (3 171)
TARIF	200	53
CUST_DEVICE	159	87
CUSTOMER_DEVICE_JT	27 150 000	9
INVOICE	147 500 000	34

Tabulka 3: Počet záznamů v databázových tabulkách (vlastní zpracování)

V případě tabulky customer byly vytvořeny pro potřeby konkrétních testů dvě různé verze. Druhá verze byla rozšířena o další sloupce s náhodnými daty pro navýšení průměrné velikosti řádku v tabulce. Velikost řádku takto upravené tabulky je uvedena v závorce.



Tabulka 4: Logická struktura databáze (vlastní zpracování)

Celková struktura databáze byla navrhována v souladu s normálními formami a splňuje podmínky třetí normální formy.

Veškeré relace obsahují hodnoty v atomické formě, hodnoty ve sloupcích tabulek jsou dále nedělitelné. Tento návrh tedy splňuje podmínky první normální formy. Příkladem, kdy by relace nesplňovala první normální formu, by bylo umístění adresy zákazníka v tabulce CUSTOMER do jednoho sloupce.

Všechny neklíčové atributy jsou plně závislé na celém primárním klíči, proto je splněna i podmínka druhé normální formy. Tato podmínka je většinou důležitá v případě použití složených primárních klíčů, což není případ použité testovací databáze.

Poslední třetí normální forma je také splněna, neboť pro žádný atribut neexistuje tranzitivní závislost na primárním klíči, tedy, že jakýkoliv atribut není závislý na jiném neklíčovém atributu, který je následně závislý na primárním klíči.

4.2 Příprava testovacího počítače

Testovací prostředí bylo kompletně nainstalováno na jeden počítač v následující konfiguraci:

- procesor Intel® Core i7 processor i7-920XM (2 GHz) with quad-core 8 MB L3 cache,
- 12 GB RAM DDR3,
- HDD1 SSD Crucial 1TB, HDD2 SATA 320 GB, otáčky 7 200 /min, buffer 16 MB, rozhraní Serial ATA 3.0 GB/s,
- síťová karta.

Vzhledem k tomu, že nejvíce výkon ovlivňující komponenta u databázových serverů jsou v praxi většinou pevné disky, byly před spuštěním testů provedeny výkonové testy na interní disky počítače, na kterých byly uloženy datové soubory databází. Pro testy byl využit nástroj od společnosti Microsoft Diskspd. Testy byly spuštěny na oba disky s následujícími parametry:

- velikost bloku 8 kB,
- doba běhu testu 60 s,
- zakázáno cachování,
- počet vláken 4,
- náhodné čtení a zápis rozdělen ve stejném poměru,
- velikost souboru pro test 100 MB.

Na následujících obrázcích je vidět výstup z testů pro oba interní disky.

```
Total IO
thread |       bytes     |     I/Os     |     MB/s    |  I/O per s |  file
------------------------------------------------------------------------------
     0 |      2837741568 |       346404 |       45.10 |    5772.36 | c:\io.dat
     1 |      2833752064 |       345917 |       45.03 |    5764.24 | c:\io.dat
     2 |      2834317312 |       345986 |       45.04 |    5765.39 | c:\io.dat
     3 |      2837798912 |       346411 |       45.10 |    5772.47 | c:\io.dat
------------------------------------------------------------------------------
total:       11343609856 |      1384718 |      180.27 |   23074.46

Read IO
thread |       bytes     |     I/Os     |     MB/s    |  I/O per s |  file
------------------------------------------------------------------------------
     0 |      1420550144 |       173407 |       22.57 |    2889.59 | c:\io.dat
     1 |      1417699328 |       173059 |       22.53 |    2883.79 | c:\io.dat
     2 |      1418190848 |       173119 |       22.54 |    2884.79 | c:\io.dat
     3 |      1414176768 |       172629 |       22.47 |    2876.63 | c:\io.dat
------------------------------------------------------------------------------
total:        5670617088 |       692214 |       90.12 |   11534.81

Write IO
thread |       bytes     |     I/Os     |     MB/s    |  I/O per s |  file
------------------------------------------------------------------------------
     0 |      1417191424 |       172997 |       22.52 |    2882.76 | c:\io.dat
     1 |      1416052736 |       172858 |       22.50 |    2880.45 | c:\io.dat
     2 |      1416126464 |       172867 |       22.50 |    2880.60 | c:\io.dat
     3 |      1423622144 |       173782 |       22.62 |    2895.84 | c:\io.dat
------------------------------------------------------------------------------
total:        5672992768 |       692504 |       90.15 |   11539.64
```

Obrázek 10: Výkonové parametry pevného disku SSD (vlastní zpracování)

```
Total IO
thread |       bytes     |     I/Os     |     MB/s    |  I/O per s |  file
------------------------------------------------------------------------------
     0 |        36741120 |         4485 |        0.58 |      74.74 | d:\io.dat
     1 |        37765120 |         4610 |        0.60 |      76.82 | d:\io.dat
     2 |        36438016 |         4448 |        0.58 |      74.12 | d:\io.dat
     3 |        36208640 |         4420 |        0.58 |      73.65 | d:\io.dat
------------------------------------------------------------------------------
total:         147152896 |        17963 |        2.34 |     299.33

Read IO
thread |       bytes     |     I/Os     |     MB/s    |  I/O per s |  file
------------------------------------------------------------------------------
     0 |        18415616 |         2248 |        0.29 |      37.46 | d:\io.dat
     1 |        18464768 |         2254 |        0.29 |      37.56 | d:\io.dat
     2 |        18554880 |         2265 |        0.29 |      37.74 | d:\io.dat
     3 |        17825792 |         2176 |        0.28 |      36.26 | d:\io.dat
------------------------------------------------------------------------------
total:          73261056 |         8943 |        1.16 |     149.02

Write IO
thread |       bytes     |     I/Os     |     MB/s    |  I/O per s |  file
------------------------------------------------------------------------------
     0 |        18325504 |         2237 |        0.29 |      37.28 | d:\io.dat
     1 |        19300352 |         2356 |        0.31 |      39.26 | d:\io.dat
     2 |        17883136 |         2183 |        0.28 |      36.38 | d:\io.dat
     3 |        18382848 |         2244 |        0.29 |      37.39 | d:\io.dat
------------------------------------------------------------------------------
total:          73891840 |         9020 |        1.17 |     150.31
```

Obrázek 11: Výkonové parametry SATA disku (vlastní zpracování)

Z výsledků je patrné, že SATA disk dosahuje značně nižší výkonnosti než SSD disk. Zatímco SSD disk dosahuje v testu zhruba 23 000IOPs, SATA disk dosahuje zhruba 300IOPs. Pro rozložení datových souborů databázových systémů byl zvolen SSD disk jako úložiště indexů a SATA disk sloužil jako úložiště pro vlastní data v tabulkách. V praxi není běžné využití SSD disku v serverech vzhledem k jejich ceně, proto byl použit klasický rotační pevný disk pro úložiště dat.

Dle zmíněné hardwarové konfigurace je jasné, že se použitý stroj nedá srovnávat se standardně používanými databázovými servery. Nicméně pro potřeby testování počítač poskytuje dostatek výkonu jak pro běh databázového stroje, tak i testovací aplikace.

4.3 Instalace databázových systémů

Na počítač byl nainstalován operační systém Windows 2008 R2 Enterprise, který je podporovaný všemi databázovými systémy použitými pro testy. Na server byla také nainstalována JAVA JRE verze 1.7 pro potřeby spouštění vyvinutých aplikací pro generování dat a testování. Databázové systémy byly nainstalovány v následujících verzích:

- Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 – 64 bit,
- Microsoft SQL server 2014,
- IBM DB2 version 10.5.

Pro zdrojovou databázi byla použita dedikovaná instance databáze Oracle.

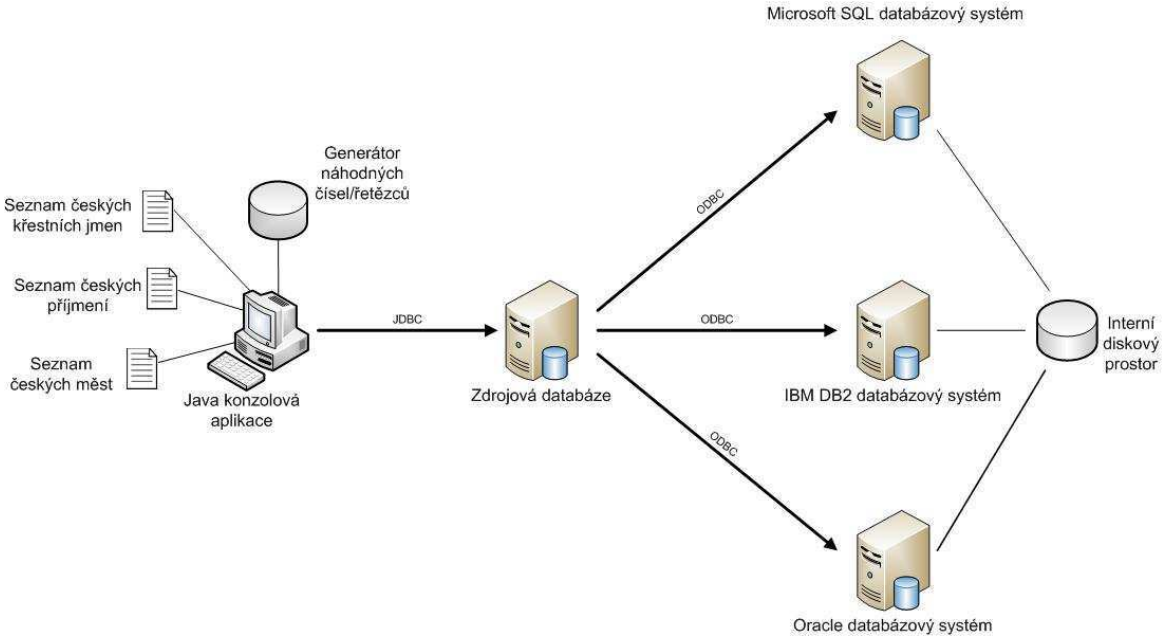
Všechny databázové systémy byly ponechány po instalaci ve svém výchozím nastavení. Znamená to, že většina parametrů databáze byla nastavena databázovým systémem na optimální hodnoty dle aktuální zátěže. Databázoví výrobci doporučují ve většině případů ponechat nastavení těchto parametrů v automatickém režimu, neboť v jejich systémech je implementována poměrně sofistikovaná logika dynamického nastavení parametrů dle aktuálních potřeb. Jediné parametry, které byly modifikovány pro provádění testů, byla velikost vyrovnávací paměti a velikost bloku datových souborů (tablespaců). Velikost datového bloku byla zvolena 8 kB jakožto střední hodnota z dostupných velikostí. Existuje doporučení vytvářet datový soubor databáze s menší velikostí bloku v případě záznamů v databázi o malé velikosti, kde dochází k náhodnému čtení dat. Naopak u sekvenčního čtení nebo v případě velkých záznamů v tabulce je doporučení na vyšší velikost bloku.

Po instalaci všech databázových systémů byly vytvořeny databázové instance dle parametrů popsaných v přílohách (příloha 2).

Do všech databází, včetně zdrojové, byla pomocí zakládacích skriptů vytvořena navržená relační databázová struktura dle předchozího návrhu. Jednotlivé zakládací skripty pro všechny databáze jsou dostupné v přílohách této práce.

4.4 Generování dat

Do zdrojové databáze byla nahrána náhodně vygenerovaná data pro potřeby následných měření. Tato data nijak nekorespondují s reálnými daty, slouží pouze pro testovací účely a v mnohých případech nedávají ani logický smysl. Data byla generována pomocí speciálně vyrobené java konzolové aplikace, která generovala vkládací skripty do databáze a následně tyto skripty po dávkách spouštěla. Na úrovni aplikace generující data nebyly prováděny žádné kontroly na integritní omezení použité relační databáze, tyto kontroly byly ponechány na vlastním nastavení tabulek databáze. Záznamy, které neprošly přes toto omezení, byly bez náhrady zahozeny. Všechna data byla nahrána do pracovní databáze, ze které se následně pomocí dávkových importů nahrávala do cílových databázových systémů, čímž byly zaručeny absolutně stejné podmínky z pohledu použitých dat pro následné měření nad různými databázovými systémy. Na následujícím obrázku je schematicky znázorněn celý proces přípravy dat od generování dat až po finální nahrání na měřený databázový systém.



Obrázek 12: Schéma procesu přípravy testovacích dat

Aplikace pro generování dat jako zdroj pro jméno zákazníka a město, ve kterém bydlí, byla použita volně dostupná data se seznamem křestních jmen, příjmení a měst v České republice. Tyto informace byly nahrány do pomocných tabulek zdrojové databáze, zakládací a vkládací skripty pro tyto tabulky jsou součástí příloh této práce (příloha 3). Tyto pomocné tabulky byly také vytvořeny a naplněny stejnými daty v databázích měřených databázových systémů. Data z nich byla použita v rámci testovací aplikace při sestavování dotazů do databází.

4.5 Testovací aplikace

Jako testovací aplikace posloužila speciálně vyvinutá java konzolová aplikace. Tato aplikace zasílala dotazy do databáze obdobným způsobem, jako se to děje v reálných situacích aplikací používajících jako zdroj dat relační databázi. Aplikace na vstupu očekávala následující parametry:

- název testovaného databázového systému,
- název testované oblasti,
- počet vláken,
- počet cyklů měření,

- detailní výstup na obrazovku pro každý dotaz (hodnota T - vypisuje detail na obrazovku, F – nevypisuje detail na obrazovku).

Název testovaného databázového systému může nabývat jedné z hodnot oracle, mssql a db2. Hodnoty parametru testované oblasti jsou uvedeny v následujících tabulkách včetně SQL dotazu a jeho popisu.

Hodnota parametru	Bufferpool
SQL dotaz	SELECT c.firstname, c.surname, c.city, i.invoice_number, i.i_amount, i.i_state FROM customer c JOIN invoice i ON c.customerid=i.customerid where c.surname = ?? AND c.firstname = ?? AND c.customerid >12000000
Popis	Dotaz testuje využití vyrovnávací paměti. Pomocí podmínky customerid > x se snižuje nebo zvyšuje oblast tabulky pro výsledek dotazu. Hodnoty pro zbylé sloupce v podmínce jsou doplněny náhodným výběrem hodnot z pomocných tabulek pro jméno a příjmení.

Tabulka 5: Testovací aplikace pro využití vyrovnávací paměti (vlastní zpracování)

Hodnota parametru	Dynamic
SQL dotaz	SELECT c.firstname, c.surname, c.city, i.invoice_number, i.i_amount, i.i_state FROM customer c JOIN invoice i ON c.customerid=i.customerid WHERE c.surname = ? AND c.firstname = ? AND c.active = ?
Popis	Aplikace posílá dotaz do databáze podle takzvaného prepared statement. Symbol jednoho otazníku znamená takzvanou „pojmenovanou proměnnou“, která je připojena k dotazu zaslanému do databáze. Hodnoty pro křestní jméno a příjmení jsou nahrazeny hodnotami z pomocných tabulek. Je vždy vybrána konkrétní hodnota na základě identifikace vlákna a aktuální pozice v cyklu. Hodnota pro podmínku active je doplněna statickou hodnotou „1“.

Tabulka 6: Testovací aplikace pro dynamické dotazy (vlastní zpracování)

Hodnota parametru	Static
SQL dotaz	SELECT c.firstname, c.suname, c.city, i.invoice_number, i.i_amount, i.i_state FROM customer c JOIN invoice i ON c.customerid=i. customerid WHERE c.surname = ?? AND c.firstname = AND c.active=1
Popis	Aplikace zasílá dotazy do databáze obdobným způsobem jako v případě testování dynamických dotazů s tím rozdílem, že hodnoty v podmínce jsou specifikovány přímo v dotazu.

Tabulka 7: Testovací aplikace pro statické dotazy (vlastní zpracování)

Hodnota parametru	Index
SQL dotaz	SELECT * FORM customer c WHERE c.surname = ?? AND c.firstname = ??
Popis	Aplikace zasílá dotazy do databáze, hodnoty pro křestní jméno a příjmení jsou nahrazeny hodnotami z pomocných tabulek. Je vždy vybrána konkrétní hodnota na základě identifikace vlákna a aktuální pozice v cyklu.

Tabulka 8: Testovací aplikace pro databázové indexy (vlastní zpracování)

Hodnota parametru	Datum
SQL dotaz	SELECT * FROM customer c WHERE startdate BETWEEN d1 AND d2
Popis	Aplikace zasílá dotazy do databáze, hodnota pro parametr d1 je generována v rozmezí 1. 1. 2000 až 31. 12. 2015. Hodnota pro d2 je vypočítána z hodnoty d1 přičtením 1 až 7 dní.

Tabulka 9: Testovací aplikace pro tabulkový partitioning (vlastní zpracování)

Hodnota parametru	Komprese
SQL dotaz	SELECT * FROM customer c WHERE c.surname = ??
Popis	Aplikace zasílá dotazy do databáze, hodnota pro příjmení je nahrazena hodnotou z pomocné tabulky. Je vždy vybrána náhodná hodnota.

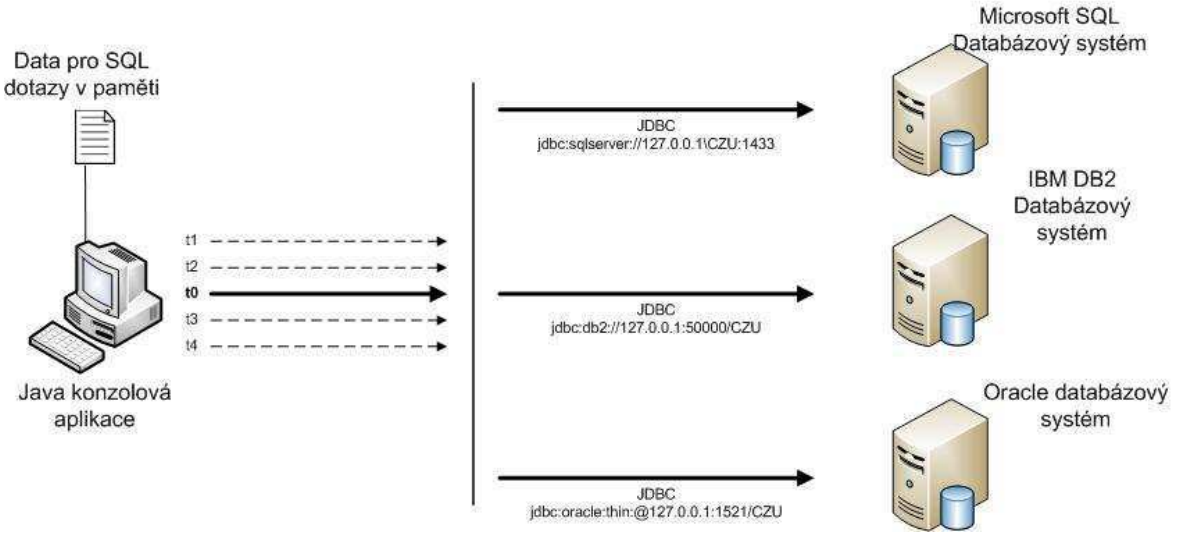
Tabulka 10: Testovací aplikace pro kompresy tabulek (vlastní zpracování)

Příklad spuštění aplikace pro testování účinnosti komprese tabulek databázového systému Oracle a zobrazení pouze výsledných časů na výstupu.

Java – jar GenerateDBLoad.jar oracle Komprese 5 10 F

Aplikace využívá připojení pomocí Java database connectivity (JDBC), což je jediná komponenta, která se mezi testy různých databázových systémů mění. Je samozřejmě jasné, že každá implementace JDBC ovladače může být různá i s ohledem na výkonnost, nicméně pro testování byla snaha využít vždy nejvyšší verzi ovladače podporovanou pro konkrétní databázový systém v konkrétní verzi.

Před každým testem jsou aplikací do paměti nahrána data potřebná pro provedení testu, aby v průběhu měření byly co nejméně zatěžovány komponenty databázového serveru a byl mu poskytnut maximální dostupný výkon. Na následujícím obrázku je jednoduše naznačeno schéma nasazení komponent pro testování.



Obrázek 13: Schéma nasazení testovací aplikace (vlastní zpracování)

4.6 Vlastní měření

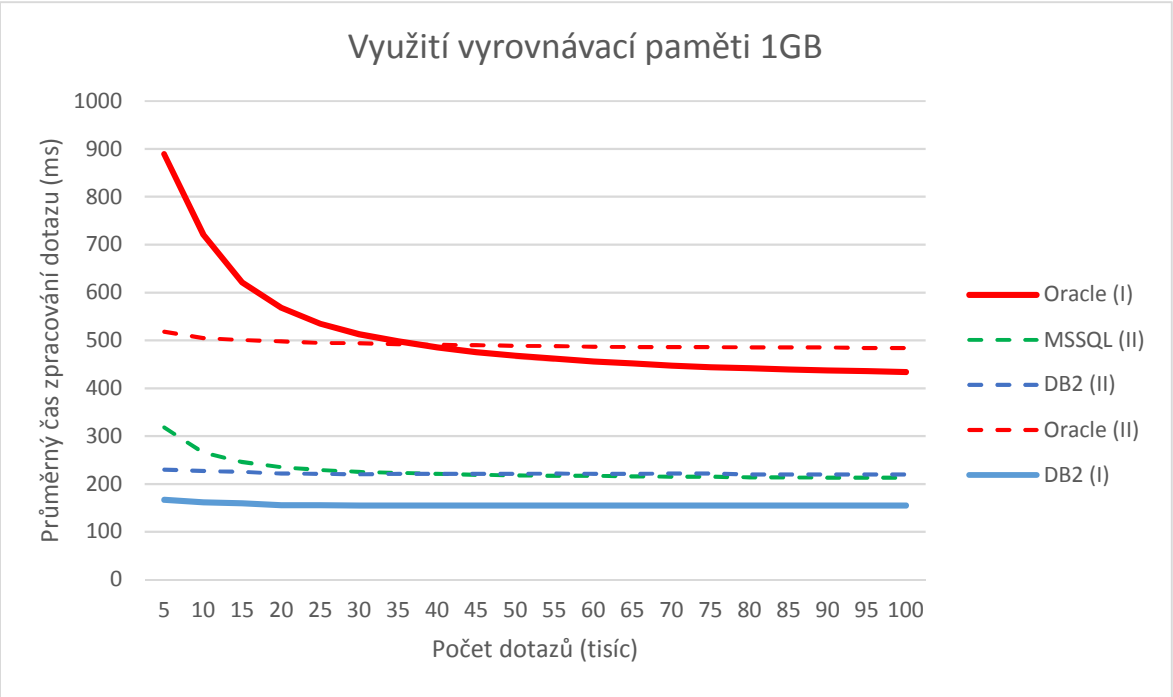
V následujících kapitolách budou popsány výsledky provedených testů včetně podmínek pro každý test v jednotlivých oblastech na vybraných databázových systémech.

4.6.1 Využití vyrovnávací paměti

V případě tohoto testu bylo před jeho zahájením potřeba upravit nastavení velikosti vyrovnávací paměti. U každého databázového systému jsou různé možnosti nastavení. Databázový systém Oracle umožňuje nastavit velikost této paměti na úrovni celé databázové instance, IBM DB2 databázový systém umožňuje nastavit paměť pro konkrétní bufferpool, který je svázán s datovým tablespace, na kterém je umístěna tabulka. Největší problém je s nastavením vyrovnávací paměti v případě Microsoft SQL serveru, který umožňuje nastavit pouze celkovou paměť využívanou databázovým systémem a velikost vyrovnávací paměti je dynamicky alokována podle aktuálních potřeb. Tam, kde to bylo možné, byla velikost vyrovnávací paměti nastavena na hodnotu 1 024 MB. Vzhledem k povaze oblasti využívání vyrovnávací paměti byl v tomto případě proveden dlouhodobý test pouze v jedné iteraci. Do databázového systému byly zasílány dotazy a vždy po dokončení pěti tisíc dotazů byl zaznamenán průměrný čas zpracování dotazu. Celkově bylo do každého databázového systému zasláno sto tisíc dotazů, výsledné hodnoty pak byly zaneseny do grafu, který srovnával průměrný čas zpracování dotazu pro různé počty historicky provedených dotazů do databáze. Tento test byl prováděn ve dvou různých variantách. V případě první varianty byla podmínka dotazu upravena pomocí hodnoty customerid > 12 000 000, v druhém případě byla hodnota této podmínky customerid > 1 000 000. Tím byla ovlivněna oblast tabulky, ze které byly načítány potenciální výsledky dotazů, kdy v prvním případě byla data čtena pouze z konkrétní malé části tabulky, kdežto v druhém případě mohly být výsledky čteny takřka z celé tabulky. V případě tohoto testu byla využita tabulka s uměle navýšenou velikostí řádky o 3 kB, aby nedošlo k nahrání celé tabulky do vyrovnávací paměti. Test byl spouštěn pomocí testovací aplikace následujícím způsobem:

java -jar GenerateDBLoad.jar název_db_systému Bufferpool 5 20000 F

Výsledky tohoto testu jsou zobrazeny v následujícím grafu. Vzhledem k nemožnosti úpravy velikosti vyrovnávací paměti u Microsoft SQL serveru byl v případě tohoto databázového systému test proveden pouze v druhé variantě.



Obrázek 14: Porovnání využití vyrovnávací paměti databázových systémů (vlastní zpracování)

Nejdůležitějším parametrem při ladění výkonu v této oblasti je „hit ratio“. Vzhledem k možnostem nastavení velikosti vyrovnávací paměti pro jednotlivé databázové systémy lze nejpřesnější údaj pro tento parametr získat z IBM DB2. Tento parametr nabýval hodnoty 62 % v první variantě a 59 % v druhé variantě. Příkladem využití vyrovnávací paměti může být následující příklad výpisu z Oracle SQL monitoru.

```
Global Stats   1st query execution
=================================================================
| Elapsed |    IO    |  Other   | Fetch | Buffer | Read | Read  |
| Time(s) | Waits(s) | Waits(s) | Calls |  Gets  | Reqs | Bytes |
=================================================================
|    0.02 |     0.02 |     0.00 |     1 |     43 |   37 | 296KB |
=================================================================

Global Stats   2nd query execution
=======================================
| Elapsed |  Other   | Fetch | Buffer |
| Time(s) | Waits(s) | Calls |  Gets  |
=======================================
|    0.00 |     0.00 |     1 |     20 |
=======================================
```

Obrázek 15: Výstup Oracle SQL monitoru s využitím vyrovnávací paměti (vlastní zpracování)

4.6.2 Statické versus dynamické dotazy

V rámci testování této oblasti byly provedeny dvě sady testů vždy po pěti měřeních v každé sadě. Mezi každým testem byla vyčištěna vyrovnávací paměť databázového systému, aby co nejméně tato komponenta ovlivňovala výsledné časy měření. U databázových systémů Oracle a IBM DB2 byla nastavena hodnota vyrovnávací paměti na 32 MB, v případě Microsoft SQL serveru toto možné není a byla tedy velikost vyrovnávací paměti ponechána na vyhodnocení databázovým systémem. Nejprve byla spuštěna sada testující statické dotazy, následně pak sada testů pro dynamické dotazy. K testům byla opět použita testovací aplikace, příklad spuštění pro statický, respektive dynamický dotaz je následující:

java -jar GenerateDBLoad.jar název_db_systému Static 5 100 F

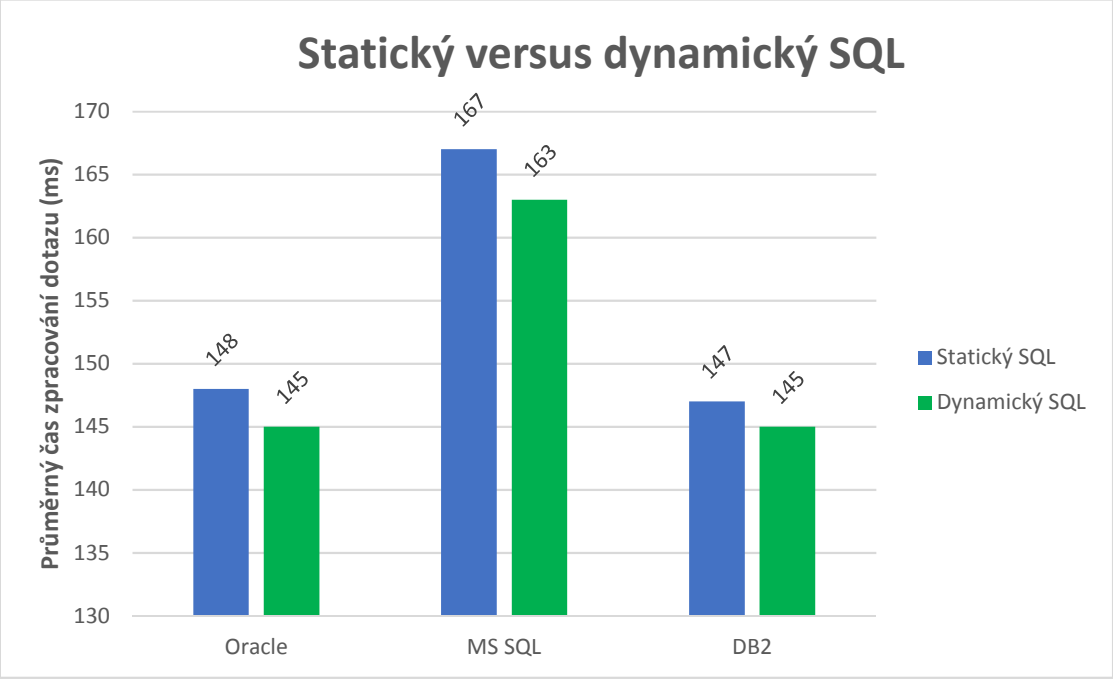
java -jar GenerateDBLoad.jar název_db_systému Dynamic 5 100 F

Pro všechna měření byly zaznamenány časy uvedené v následující tabulce.

	Oracle		MS SQL		DB2	
#	Statický SQL (ms)	Dynamický SQL (ms)	Statický SQL (ms)	Dynamický SQL (ms)	Statický SQL (ms)	Dynamický SQL (ms)
1.	140	143	160	158	139	137
2.	141	144	167	163	145	143
3.	**148**	**145**	**167**	**163**	**147**	**145**
4.	151	147	168	167	150	145
5.	160	162	169	172	150	152

Tabulka 11: Průměrný čas zpracování dotazu statický versus dynamický dotaz (vlastní zpracování)

Z výše uvedených výsledků měření byl pro každou kombinaci databázový systém, typ dotazu určen medián naměřených hodnot z pěti iterací. Pro lepší přehlednost byla tato hodnota zanesena do následujícího grafu.



Obrázek 16: Porovnání času zpracování statického a dynamického SQL dotazu (vlastní zpracování)

4.6.3 Databázové indexy

Cílem tohoto testu je porovnat různé konfigurace databázového indexu nad sloupci databázové tabulky. V praxi existují dva přístupy tvoření databázových indexů. Jedním jsou jednoduché indexy nad všemi sloupci tabulky, přes které jsou prováděny dotazy, druhým je

snaha vytvořit složené indexy nad více sloupci tak, aby co nejvíce splňovaly požadavky na dotazy do tabulek. V tomto testu byly porovnány oba přístupy. V první variantě byly vytvořeny dva oddělené indexy nad tabulkou customer pro sloupce firstname, respektive surname. Po vytvoření indexů a přepočtu statistik byla spuštěna sada dotazů, opět s pěti opakováními. Opět bylo provedeno vyčištění vyrovnávací paměti před každým testem s velikostí této paměti 32 MB pro databázové systémy, kde je to možné. Testovací aplikace byla spouštěna následujícím způsobem:

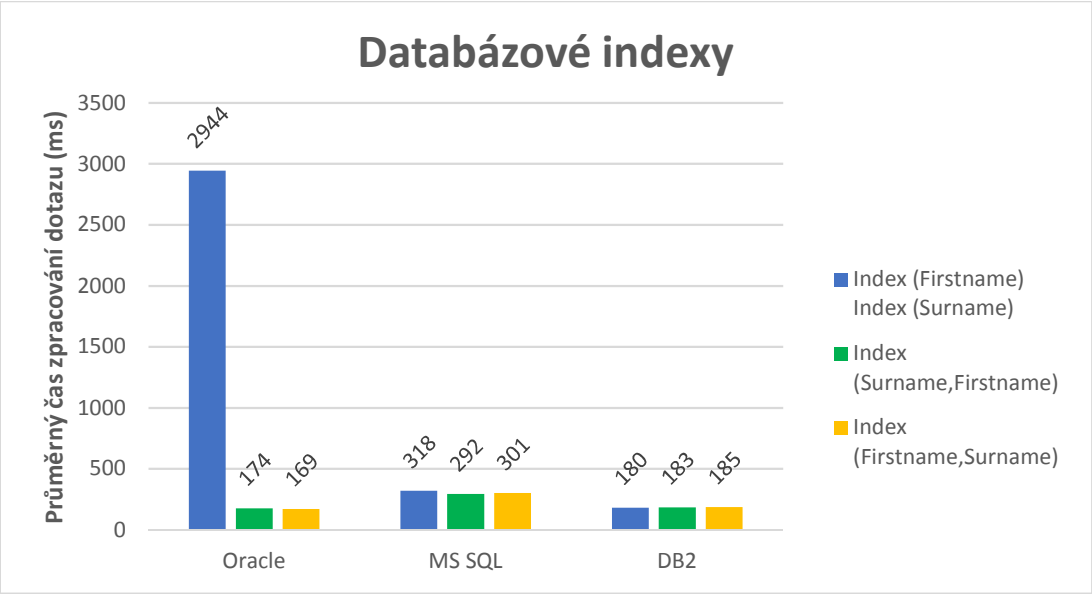
java -jar GenerateDBLoad.jar název_db_systému Index 5 5 F

Výsledné časy byly opět zaznamenány. Po dokončení této sady testů byly oba indexy zrušeny a nahrazeny jedním složeným s pořadím sloupců v indexu surname, firstname. Po přepočtu databázových statistik byly identickým způsobem spuštěny testy jako v předchozím případě a časy měření zaznamenány do následující tabulky. Posledním testem bylo otočení pořadí sloupců ve složeném indexu, tedy index nad tabulkou customer ve složení firstname, surname.

	Oracle			**MS SQL**			**DB2**		
#	**Index Firstname Index Surname (ms)**	**Index Surname, Firstname (ms)**	**Index Firstname ,Surname (ms)**	**Index Firstname Index Surname (ms)**	**Index Surname, Firstname (ms)**	**Index Firstname ,Surname (ms)**	**Index Firstname Index Surname (ms)**	**Index Surname, Firstname (ms)**	**Index Firstname ,Surname (ms)**
1.	2393	163	158	271	239	238	158	155	143
2.	2577	172	173	290	286	256	167	164	160
3.	**2944**	**174**	**169**	**318**	**292**	**301**	**180**	**183**	**185**
4.	3067	181	175	319	304	310	194	184	191
5.	3080	193	201	325	325	316	211	189	198

Tabulka 12: Průměrný čas zpracování dotazu jednoduchý versus složený index (vlastní zpracování)

Z výše naměřených hodnot byl opět určen medián a zanesen do grafu pro srovnání.



Obrázek 17: Porovnání času zpracování dotazu jednoduchý versus složený index (vlastní zpracování)

Vzhledem k tomu, že v případě databázových indexů je nutné se ohlížet i na opačnou stranu týkající se vkládání, modifikace a rušení dat, byl proveden další test vložení většího množství záznamů v případě existence databázového indexu. Po dokončení předchozích testů byl složený index surname, firstname ponechán nad tabulkou a bylo provedeno vložení zhruba jednoho milionu záznamů do tabulky customer s indexem. Následně byl zaznamenán čas nutný pro vložení tohoto počtu záznamů. Druhý test byl proveden obdobným způsobem, pouze s tím rozdílem, kdy byl databázový index zrušen. Počty vložených záznamů pro oba případy jsou uvedeny v následující tabulce a udává počet vložených záznamů za jednu sekundu.

Oracle		**MS SQL**		**DB2**	
Tabulka s indexem (počet záznamů)	**Tabulka bez indexu (počet záznamů)**	**Tabulka s indexem (počet záznamů)**	**Tabulka bez indexu (počet záznamů)**	**Tabulka s indexem (počet záznamů)**	**Tabulka bez indexu (počet záznamů)**
932	1980	2291	3043	885	1401

Tabulka 13: Počet vložených záznamů do tabulky s indexem versus bez indexu za jednu sekundu (vlastní zpracování)

4.6.4 Tabulkový partitioning

V tomto testu bylo ověřeno využití tabulkového partitioningu pro tabulku customer rozdělenou na jednotlivé partition podle sloupce startdate. Každá partition obsahovala data za jeden měsíc. V rámci tohoto testu byla prováděna celkem tři měření, a to konkrétně pro tabulku bez partitioningu, pouze s indexem nad sloupcem startdate (varianta A). Následně byla zrušena původní tabulka customer a vytvořena nová s nastaveným partitioningem (varianta B). Do takto upravené tabulky byla opět nahrána data a provedeno další měření. Poslední měření bylo provedeno na stejné tabulce s partitioningem, rozšířené o index nad sloupcem startdate (varianta C). Velikost vyrovnávací paměti byla nastavena na minimální hodnotu 32 MB a před každým měřením byla tato paměť vyčištěna. Testovací aplikace byla spouštěna následujícím způsobem:

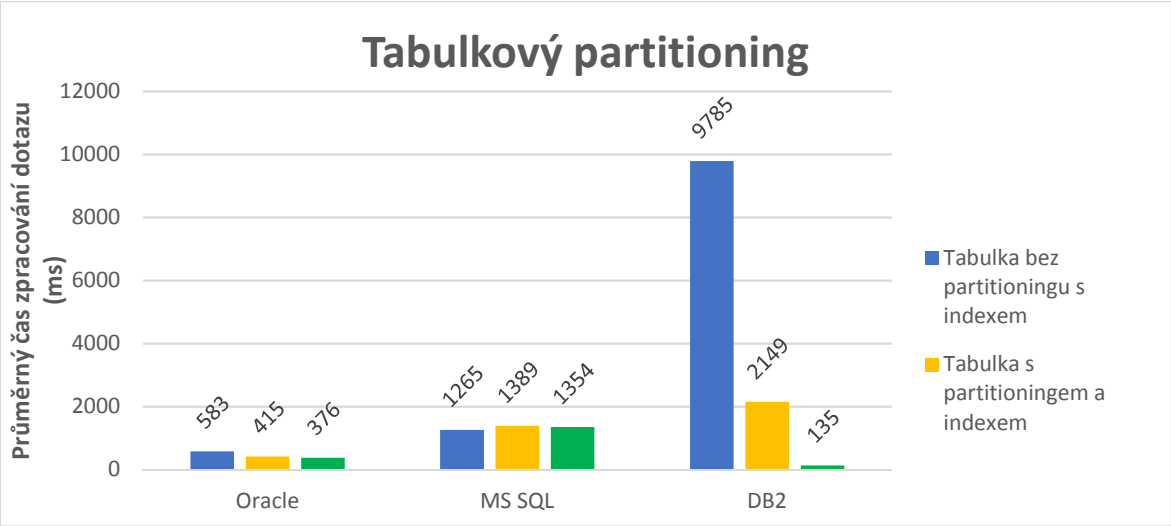
java -jar GenerateDBLoad.jar název_db_systému Datum 5 10 F

Výsledky jednotlivých měření, kterých bylo pět, byly zaznamenány do následující tabulky.

	Oracle			**MS SQL**			**DB2**		
#	**Var. A (ms)**	**Var. B (ms)**	**Var. C (ms)**	**Var. A (ms)**	**Var. B (ms)**	**Var. C (ms)**	**Var. A (ms)**	**Var. B (ms)**	**Var. C (ms)**
1.	546	290	300	1231	1357	1311	8308	1675	123
2.	556	317	348	1233	1367	1342	9750	1908	128
3.	**583**	**415**	**376**	**1265**	**1389**	**1354**	**9785**	**2149**	**135**
4.	626	423	453	1301	1394	1403	10134	2262	137
5.	629	459	647	1406	1431	1461	10672	2527	165

Tabulka 14: Porovnání času zpracování dotazu pro tabulku s indexem versus partitioningem (vlastní zpracování)

Z hodnot byl opět stanoven medián a zanesen do grafu pro porovnání.



Obrázek 18: Graf porovnání času zpracování dotazu pro tabulku s indexem versus partitioningem (vlastní zpracování)

Na následujících obrázcích je znázorněn výstup z dotazovacího plánu pro oba typy dotazů z databáze Oracle. Nejdůležitější na něm je rozdíl ve velikosti načítaných dat. V tomto konkrétním případě bylo nutné načíst pro tabulku s indexem celkem 92 MB, kdežto pro tabulku s partitioningem pouze 8 MB.

```
Global Stats
=============================================================================
| Elapsed |   Cpu   |    IO    | Other   | Fetch | Buffer | Read | Read  |
| Time(s) | Time(s) | Waits(s) | Waits(s) | Calls |  Gets  | Reqs | Bytes |
=============================================================================
|      36 |    0.03 |       36 |     0.02 |   241 |   1204 |  126 |   8MB |
=============================================================================

SQL Plan Monitoring Details (Plan Hash Value=98083385)
=========================================================================================================================================
| Id |        Operation         |    Name    |  Rows   | Cost |   Time    | Start  | Execs |   Rows   | Read | Read  | Activity | Activity Detail |
|    |                          |            | (Estim) |      | Active(s) | Active |       | (Actual) | Reqs | Bytes |   (%)    |   (# samples)   |
=========================================================================================================================================
|  0 | SELECT STATEMENT         |            |         |      |        59 |     +6 |     1 |    12016 |      |       |          |                 |
|  1 |   PARTITION RANGE SINGLE |            |   13957 |  299 |        59 |     +6 |     1 |    12016 |      |       |          |                 |
|  2 |    TABLE ACCESS FULL     | CUSTOMER   |   13957 |  299 |        65 |     +0 |     1 |    12016 |  126 |   8MB |          |                 |
=========================================================================================================================================
```

Obrázek 19: Dotazovací plán za použití partitioningu tabulky (vlastní zpracování)

```
Global Stats
=============================================================
| Elapsed |   Cpu   |    IO    | Fetch | Buffer | Read | Read  |
| Time(s) | Time(s) | Waits(s) | Calls |  Gets  | Reqs | Bytes |
=============================================================
|      85 |    0.41 |       84 |   241 |  12211 | 11829 |  92MB |
=============================================================

SQL Plan Monitoring Details (Plan Hash Value=2834641786)
=========================================================================================================================================
| Id |              Operation               |       Name        |  Rows   | Cost |   Time    | Start  | Execs |   Rows   | Read | Read  | Activity | Activity Detail |
|    |                                      |                   | (Estim) |      | Active(s) | Active |       | (Actual) | Reqs | Bytes |   (%)    |   (# samples)   |
=========================================================================================================================================
|  0 | SELECT STATEMENT                     |                   |         |      |        87 |     +2 |     1 |    12016 |      |       |          |                 |
|  1 |   TABLE ACCESS BY INDEX ROWID BATCHED | CUSTOMER          |   13948 | 13905 |        90 |     +0 |     1 |    12016 | 11795 |  92MB |          |                 |
|  2 |    INDEX RANGE SCAN                  | CUSTOMER_DATE_INX |   13948 |    39 |        87 |     +2 |     1 |    12016 |    34 | 272KB |          |                 |
=========================================================================================================================================
```

Obrázek 20: Dotazovací plán za použití databázového indexu (vlastní zpracování)

4.6.5 Tabulková komprese

Posledním prováděným testem byla měření času dotazu na komprimovaných tabulkách. V případě komprimace použité databázové systémy poskytují různé možnosti od komprese jednotlivých hodnot v tabulkách, přes komprese celých řádků až po komprese celých datových bloků. Databázové systémy Oracle a Microsoft SQL umožňují mimo jiné kompresi celých bloků, IBM DB2 kompresy jednotlivých řádků. Tyto způsoby komprese byly zvoleny pro provádění testů. Aby bylo možné porovnat výkonové parametry při různých kompresních poměrech, byly vytvořeny dvě verze tabulek zákazníků. V prvním případě bylo dosaženo kompresního poměru 15:1 (v případě DB2 30:1), v druhém případě byl kompresní poměr 9:1 (v případě DB2 18:1). Na obou verzích tabulek byly provedeny postupně testy měření času pro komprimovanou tabulku a tabulku bez komprese. Testovací aplikace byla spuštěna následujícím způsobem:

java -jar GenerateDBLoad.jar název_db_systému Komprese 5 10 F

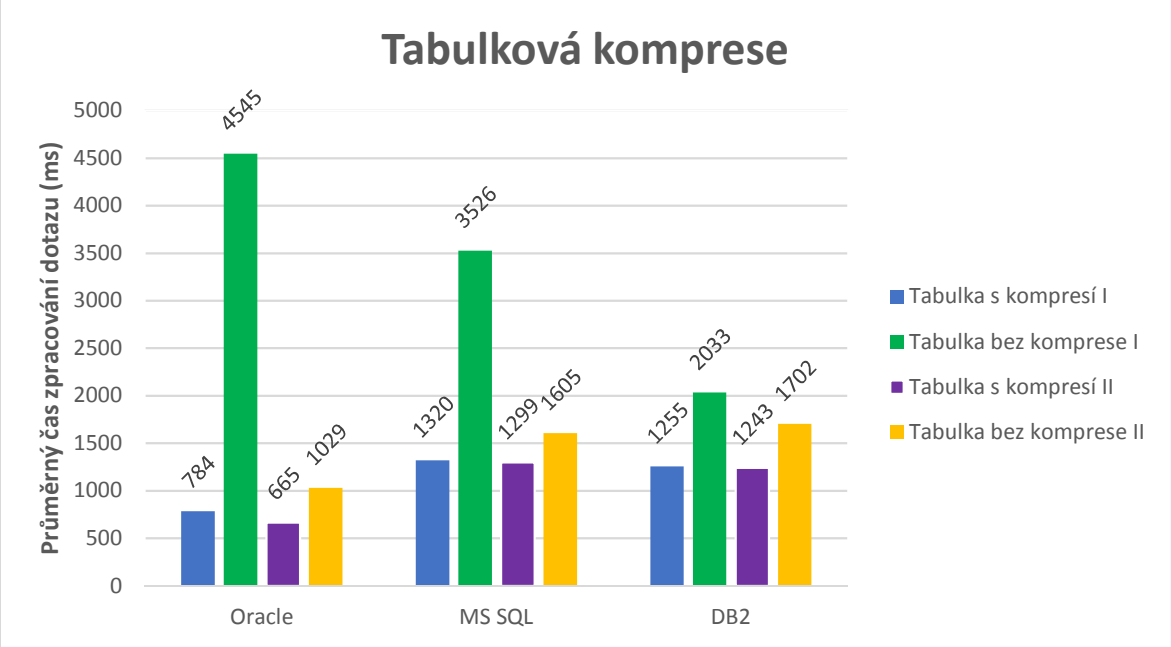
Pro každý typ měření bylo potřeba upravit databázovou tabulku a to buď verze bez komprese, nebo s kompresí. Nastavení vyrovnávací paměti bylo opět sníženo na minimum a paměť byla čištěna před každým cyklem testu, kterých bylo opět pět. Pro lepší přehlednost tabulky s naměřenými hodnotami byly použity následující zkratky:

- T1 – tabulka verze 1 bez komprese,
- TK1 – tabulka verze 1 s kompresí 15:1, respektive 30:1,
- T2 – tabulka verze 2 bez komprese,
- TK2 – tabulka verze 2 s kompresí 9:1, respektive 18:1.

	Oracle				**MS SQL**				**DB2**			
#	**TK1**	**T1**	**TK2**	**T2**	**TK1**	**T1**	**TK2**	**T2**	**TK1**	**T1**	**TK2**	**T2**
1.	654	4264	646	898	1071	2859	992	1543	1157	1985	1192	1632
2.	772	4435	650	1023	1191	3187	1050	1587	1205	2030	1207	1686
3.	**784**	**4545**	**665**	**1029**	**1320**	**3526**	**1299**	**1605**	**1255**	**2033**	**1243**	**1702**
4.	806	4632	728	1094	1743	3594	1669	1871	1383	2094	1299	1781
5.	820	5146	745	1123	1789	3857	1701	1903	1474	2142	1347	1803

Tabulka 15: Porovnání času zpracování dotazu pro komprimované tabulky s různým kompresním poměrem (vlastní zpracování)

Medián naměřených hodnot pro různé varianty komprimace tabulek je pro porovnání uveden v následujícím grafu.



Obrázek 21: Porovnání času zpracování dotazu různé způsoby komprese tabulek (vlastní zpracování)

5 Výsledky a diskuze

Z výsledků praktických měření se dá říct, že všechny zkoumané databázové systémy poskytovaly vesměs stejný výkon v případě použité testovací databáze a práce s ní. Jisté rozdíly nastávaly pouze v určitých oblastech.

V průběhu většiny měření bylo pozorováno, že největším faktorem ovlivňujícím celkový výkon databáze je úložiště určené pro data databázových tabulek. Ideálním řešením by bylo umístit většinu dat do paměti databázového serveru, což bylo dokázáno v praktické části v rámci ověření využití vyrovnávací paměti. Toto je opět v praxi ve většině případů nemožné, neboť databázové servery nedisponují takovou kapacitou operační paměti, která by byla schopna pojmout většinu dat databáze. Je tedy potřeba věnovat pozornost rozmístění datových souborů databáze na různá fyzická úložiště. Prvním způsobem oddělení dat je umístění souborů pro data tabulek a data pro indexy nad tabulkami na různá úložiště. Dalším řešením je umístění souborů pro data tabulky s vyšší frekvencí přístupů na dedikované úložiště. S výhodou lze využít i různě výkonná úložiště ve spojení s tabulkovým partitioningem, pokud lze určit, že data v konkrétních částech tabulky budou využívána mnohem častěji než zbylá data v této tabulce. Pak lze umístit tyto partitions na rychlejší disky a zbylé ponechat na pomalejším úložišti. V tomto případě jsou všechna data dostupná, s tou výhodou, že častěji využívaná data jsou uložena na rychlejším úložišti, a tedy i přístup k nim je rychlejší.

Další zkoumanou oblastí poskytující možnost zlepšení výkonu, byla komprese dat v tabulkách. Z výsledků měření na použité databázi se tato oblast jeví spíše vhodná z pohledu ušetření nutného diskového prostoru než jako způsob, jak vylepšit výkon databázového systému. Nejvyšší pozorované zlepšení bylo v případě operací skenující celou tabulku, což ale není většinový případ práce s daty v praxi. Při měření rychlosti odezvy se rozdíly mezi komprimovanou tabulkou a tabulkou bez komprese smazávaly už při úrovni komprese okolo 10:1. V případě nižšího kompresního poměru nebyly rozdíly skoro žádné. Před rozhodnutím, zda použít kompresy dat v tabulkách, je dobré použít nástroje databázových systémů pro odhad výsledného kompresního poměru konkrétní tabulky.

Nejpoužívanějším prostředkem pro zlepšení výkonu relační databáze jsou indexy. Existuje poučka, že v případě složeného indexu je dobré určit pořadí sloupců tabulky v indexu podle jejich selektivity. Toto se v praktické části nepotvrdilo a výsledné časy

zpracování dotazu byly totožné pro různé pořadí sloupců v indexu. Obdobných výsledků bylo dosaženo i při porovnávání času zpracování v případě více jednoduchých indexů oproti použití složeného indexu. Zde totiž databázové systémy s výjimkou databáze Oracle dosahovaly podobných časů. Důležitou volbou v případě tvorby databázových indexů je způsob vyhledávání. V případě složených indexů je dobré na první místo spíše umisťovat sloupečky, které jsou používány i v jiných kombinacích dotazu a je tedy možné stejný index použít i pro jiný typ dotazu. Důležité je také neustále monitorovat využívání indexů databází a v případě nepoužívání indexu jej zrušit, neboť i z výsledků měření je patrné, že už pouhý jeden index složený ze dvou sloupců poměrně značně snižuje rychlost vkládání záznamů do tabulky s indexem.

Následuje souhrnná SWOT analýza pro všechny zkoumané oblasti.

Databázový index

Silné stránky	**Slabé stránky**
• Zpracování dotazu a přímý přístup k datům pomocí identifikátoru řádky • Nižší zatížení úložiště dat databáze	• V případě nízké kardinality sloupce nepřináší efekt • Restrukturalizace při změně dat
Příležitosti	**Hrozby**
• Rychlý přístup uživatele k datům	• Pomalé ukládání dat • Extra požadavky na úložiště

Tabulka 16: SWOT analýza databázového indexu (vlastní zpracování)

Tabulkový partitioning

Silné stránky	**Slabé stránky**
• Nemá extra nároky na diskový prostor • Pro data přistupuje pouze do konkrétní partition, nezatěžuje tolik úložiště	• Možné definovat pouze na jeden sloupec tabulky
Příležitosti	**Hrozby**
• Rychlý přístup uživatele k datům pomocí hodnot sloupce definující partition • Možnost použít rychlejší úložiště pro data s vyšší frekvencí přístupu	• Při neočekávané změně hodnot ve sloupci určující partion nelze uložit data

Tabulka 17: SWOT analýza tabulkového partitioningu (vlastní zpracování)

Dynamické dotazy

Silné stránky	**Slabé stránky**
• Nevyhodnocuje se přístupový plán pro identické dotazy, nezatěžuje databázový server	• Nutno definovat při vývoji aplikace
Příležitosti	**Hrozby**
• Rychlejší vyhodnocení dotazu a tedy i přístup uživatele k datům	• V případě změny dat a tím i změny přístupového plánu hrozí použití neefektivního plánu

Tabulka 18: SWOT analýza dynamických SQL dotazů (vlastní zpracování)

Statické dotazy

Silné stránky	**Slabé stránky**
• Přístupový plán je vyhodnocen pro každý dotaz, tedy se znalostí aktuální skladby dat	• Přístupový plán je vyhodnocován, i když to není nutné
Příležitosti	**Hrozby**
• Efektivnější při časté změně skladby dat	• Nutný čas i výkon databázového serveru pro vyhodnocení přístupového plánu

Tabulka 19: SWOT analýza statických SQL dotazů (vlastní zpracování)

Vyrovnávací paměť

Silné stránky	**Slabé stránky**
• Eliminace časově náročných operací čtení z externího úložiště	• Nepřináší efekt v případě různorodých požadavků na data obsahující většinu databáze • Různé možnosti konfigurace pro různé databázové systémy
Příležitosti	**Hrozby**
• Rychlé odezvy na požadavky v případě majority požadavků dat z určité oblasti	• V případě požadavků vracející větší objem dat jsou data v paměti neustále měněna

Tabulka 20: SWOT analýza využití vyrovnávací paměti (vlastní zpracování)

Komprese tabulek

Silné stránky	**Slabé stránky**
• Nižší požadavky na prostor úložiště • Nižší požadavky na čtení z disku	• Vyšší nároky na CPU • Nejedná-li se o sekvenční čtení dat, nemusí přinášet efekt • Závisí na kompresním poměru
Příležitosti	**Hrozby**
• Rychlejší záloha i obnova dat databáze	• Pomalejší vkládání dat zejména dávkového

Tabulka 21: SWOT analýza komprese dat v tabulkách (vlastní zpracování)

V následující části jsou uvedeny tři vybrané typické způsoby využití relačních databází v praxi. Pro každý typ existují různé nároky na výkon. Různé možnosti implementace jsou zde rozebrány z pohledu zkoumaných oblastí této práce.

Využití prvků relačních databází pro ladění výkonu datových skladů

Vyrovnávací paměť	Využití vyrovnávací paměti v tomto případě nemá velký smysl, neboť analýza je prováděna nad všemi uloženými daty. Z pohledu paměti je v tomto případě kladen spíše důraz na dostatek paměti pro třídění a podobné operace relační databáze.
Statické versus dynamické dotazy	Vzhledem k tomu, že v tomto případě se dotazy do databáze vyznačují svou složitostí nikoliv počtem, je vhodné použití statických dotazů. Pro tyto je vždy vyhodnocen dotazovací plán na základě hodnot v podmínkách dotazu a aktuálních databázových statistikách.
Databázový index	V případě databázového indexu je doporučení pokrytí všech používaných dotazů indexy. Pro datové sklady není kladen primární výkonový požadavek na vložení dat, ale hlavně na práci s nimi. Data jsou do datového skladu vkládána pomocí dávkových importů maximálně několikrát za den. V tomto případě se dá přepočet indexu provést až po ukončení dávky vložení dat.
Tabulkový partitioning	Ačkoliv by se v případě datového skladu většinou dalo najít spoustu možností, jak data kategorizovat podle určitého sloupce tabulky, způsob práce s daty v tomto případě nevyužije výhod tabulkového partitioningu.
Tabulková komprese	Toto je oblast, o které je v případě datových skladů dobré uvažovat. Je to nejen z toho důvodu, že datové sklady obsahují velké množství dat, ale také z toho důvodu, že zde může existovat velké množství duplicit. Proto je použití tabulkové komprese vhodné pro použití v datových skladech.

Tabulka 22: Doporučení řešení výkonu pro datové sklady (vlastní zpracování)

Využití prvků relačních databází pro ladění výkonu provozních databázi s rozlišením aktivních a pasivních dat.

Vyrovnávací paměť	Vzhledem k tomu, že se v majoritě případů pracuje s konkrétní menší sadou dat, je vhodné využít vyrovnávací paměť a nastavit její velikost tak, aby byla schopna udržet většinu dat týkajících se aktuálních informací.
Statické versus dynamické dotazy	Tento typ databáze bude ve většině případů poskytovat odpovědi na velké množství dotazů, které ale budou svojí strukturou identické. Proto je vhodné použití dynamických dotazů s využitím hostitelských proměnných.
Databázový index	V tomto případě je potřeba volit rozumný počet databázových indexů a najít vhodný poměr mezi rychlostí vkládání a práce s daty. Je zde potřeba pravidelný monitoring a snaha udržovat pouze relevantní databázové indexy.
Tabulkový partitioning	Tento nástroj je velice vhodný pro využití v případě tohoto typu relační databáze. Ideální je rozdělení partition podle sloupce, který kategorizuje aktuální a pasivní data v tabulkách. Pokud je to možné, je vhodné přesunout datový soubor s aktuálními partitions na rychlejší nebo oddělené úložiště.
Tabulková komprese	Před použitím komprese dat je potřeba provést jejich analýzu pomocí nástrojů databázových systémů pro odhad výsledného kompresního poměru. Pokud je odhad dosažení kompresního poměru alespoň 5:1, pak je využití komprese vhodné.

Tabulka 23: Doporučení řešení výkonu provozní databáze s rozdělením aktuálních a historických dat (vlastní zpracování)

Využití prvků relačních databází pro ladění výkonu provozních databázi bez rozlišení aktivních a pasivních dat.

Vyrovnávací paměť	Vzhledem k tomu, že zde je předpoklad, že uživatelé budou pracovat s většinou dat, je potřeba v tomto případě monitorovat parametr „buffer cache hit ratio“ a snažit se dostat jeho hodnotu na co nejvyšší číslo. Některé databázové systémy, například od společnosti Oracle, poskytují nástroje pro odhad efektu navýšení vyrovnávací paměti, který je možné použít při ladění.
Statické versus dynamické dotazy	Platí obdobné doporučení jako v předchozím případě. I zde platí předpoklad na větší počet identických dotazů do databáze.
Databázový index	I zde je doporučení totožné s předchozí databází, je opět potřeba najít kompromis mezi časem pro vložení a vyhledání záznamu. Opět je nutný pravidelný monitoring a reakce na změny v užívání databázových indexů.
Tabulkový partitioning	Zde je doporučena důkladná analýza, neboť správné nastavení partitioningu může rapidně zrychlit dobu odezvy na uživatelské požadavky. Důležité je zde určení správného sloupce tabulky pro kategorizaci dat. Tento sloupec musí být také součástí uživatelských dotazů. Správa uložení dat pro partitionovanou tabulku zde není nutná, zrychlení je dáno tím, že databázový systém přistupuje pro data pouze do určité oblasti úložiště dle hodnoty podmínky v dotazu.
Tabulková komprese	Opět stejné doporučení jako v předchozím případě, nutno před nasazením provést odhad kompresního poměru a v případě hodnoty alespoň 5:1 je vhodné kompresy použít.

Tabulka 24: Doporučení řešení výkonu provozní databáze bez rozdělení aktuálních a historických dat (vlastní zpracování)

6 Závěr

Ladění výkonnosti relační databáze je velmi komplexní úloha a jedná se v podstatě o každodenní práci správce databázového systému. V této práci byly zpracovány pouze určité oblasti ovlivňující výkon relačních databází. V praxi je tento problém mnohem rozsáhlejší a oblastí, na které je potřeba se zaměřit, je mnohem více. Také použitá databáze pro měření se zcela jistě svým objemem liší od v praxi používaných databází. Nicméně již na takovéto velikosti databáze se ve zkoumaných oblastech projevovaly určité rozdíly a je jisté, že se s velikostí databáze a počtem požadavků na ni budou zvětšovat.

Ideálním případem by byla situace, kdy je v průběhu nasazení aplikace provedeno i naladění výkonu použitého databázového systému, které je efektivní po celou dobu životního cyklu aplikace. Toto ale ve světě informačních technologií neplatí a v případě relačních databází už vůbec ne. I v průběhu zpracovávání praktické části této práce došlo k situaci, kdy v případě generování dat se celý proces s přibývajícími daty v databázi zpomaloval. Toto je i reálný případ většiny aplikací a jejich databázových systémů, neboť v současnosti je spíše rostoucí trend objemu dat, která je nutno v databázi udržovat. Problém výkonnosti databázového systému není dán pouze počtem požadavků na data v databázi, ale také právě jejich objemem.

Hlavním cílem této práce bylo navrhnout použití jednotlivých prostředků pro ladění výkonu relační databáze s ohledem na její využití a způsob nasazení. V reálných situacích není v podstatě možné najít dvě stejné relační databáze a navrhnout pro ně stejné úpravy pro dosažení optimálního výkonu. Je tedy hlavně důležité použít správné prostředky pro každou konkrétní implementaci relační databáze.

V teoretické části této práce byly popsány základy relačních databází a algoritmů, které jsou využívané při manipulaci s daty relační databáze. Dále byly také popsány vybrané oblasti pro ladění výkonu relační databáze, které byly následně použity v praktické části pro ověření chování na různých databázových systémech. V této části byly také poskytnuty informace o nástrojích, které jsou vhodné pro potřeby monitoringu výkonu relační databáze.

V praktické části byla na vybraných databázových systémech provedena měření, která následně posloužila pro návrh využití jednotlivých možností pro ladění výkonu pro různé způsoby implementace relačních databází. Měření probíhala za simulace reálného provozu na malé až střední databázi co do objemu dat v tabulkách databáze. Bylo zjištěno, že všechny

použité databázové systémy se ve zkoumaných oblastech příliš neliší a poskytují obdobný výkon. V úvodní fázi praktické části byl pro potřeby generování testovacích dat použit databázový systém MySQL, který se řadí mezi střední databázové systémy na rozdíl od databázových systémů použitých v této práci, které se řadí mezi špičku. Tento databázový systém již při objemu jednotek milionů záznamů v databázi vykazoval značně nižší výkon než ostatní databázové systémy, a proto byl nahrazen jiným a v této práci dále nevyužit. Tato práce si nekladla za cíl určit nejvýkonnější databázový systém, ale z výsledků vyplývá, že všechny poskytují zhruba stejný výkon. Rozhodnutí o tom, jaký databázový systém vybrat pro vlastní aplikaci, bude spíše určen jinými faktory, jakými může být znalost jednotlivých databázových systémů správci aplikace, politikou specifikace používaného software společnosti, podporou databázového systému aplikací a v neposlední řadě i cenou. Všechny databázové systémy při správném nastavení a konfiguraci poskytnou optimální výkon pro aplikaci.

7 Citovaná literatura

Bayer, R. (2002). *Organization and maintanance of large ordered indexes.* Springer Berlin Heidelberg.

BOUCHER, T. O., & Ali, Y. (2006). *Design of industrial information systems.* Boston: Elsevier Science. ISBN 9780123704924

Codd, E. F. (1970). *A relational model of data for large shared data banks.* San Jose: IBM.

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms.* Cambridge, Massachusetts: MIT Press.

Harrington, J. (2009). *Relational Database Design and Implementation.* Elsevier Science. ISBN 9780123747303

Chaudhuri, S., & Narasayya, V. (2007). *Self-Tuning Database Systems: A Decade of Progress.* VLDB Endowment.

Inmonn, W. H. (2002). *Building the Data Warehous* (3rd. vyd.). New York: John Wiley & Sons, Inc. ISBN 0-471-08130-2

Kent, W. (1983). *A simple guide to five normal forms in relational database theory.* San Jose: IBM.

LoForte, R., Wort, S., & Jorgensen, A. (2012). *Professional Microsoft SQL Server 2012 Administration.* John Wiley & Sons, Incorporated. ISBN 9781118106884

Lui, H. (2011). *Oracle database performance and scalability.* IEEE Computer Society Press. ISBN 9781118135532

Nagabhushana, S. (2006). *Data Warehousing OLAP and Data Mining.* New Delhi, India: New Age International. ISBN 9788122417647

Nielsen, P., White, M., & Parui, U. (2009). *Microsoft SQL server 2008 bible.* Indianapolis: John Wiley & Sons, Incorporated.

Niemann, T. (2008). *Sorting and searching algorithms.* Portland: epaperpress.com.

Oracle. (5 2015). *Oracle Database Concept*. Získáno 11. 3 2017, z Oracle Help center: https://docs.oracle.com/cd/E11882_01/server.112/e40540/logical.htm#CNCPT302

Oracle. (11 2016). Načteno z Oracle Database Reference, 12c Release 1 (12.1): https://docs.oracle.com/database/121/REFRN/

Oracle. (9 2016). *Analyzing SQL with SQL tuning advisor*. Získáno 11. 3 2017, z Oracle help center: https://docs.oracle.com/database/121/TGSQL/title.htm

Oracle. (10 2016). *Oracle Database Administrator's Guide, 12c Release 1*. Získáno 11. 3 2017, z Oracle help center: https://docs.oracle.com/database/121/ADMIN/create.htm#ADMIN11108

Pokorný, J., & Valenta, M. (2013). *Databázové systémy.* Praha: České vysoké učení technické. ISBN 978-80-01-05212-9

Powell, G. (2011). *Oracle performance tuning for 10gR2.* Elsevier Science. ISBN 9781555583453

Rahayu, W., Leung, C., & Taniar, D. (2008). *High Performance Parallel Database Processing and Grid Databases.* John Wiley & Sons, Incorporated. ISBN 9780470107621

Selinger, P., Astrahan, M., Chamberlin, D., Lorie, R., & Price, T. (1973). *Access Path Selection in a Relational Database Management System.* San Jose: IBM Research division.

Sherrod, A. (2007). *Data Structures and Algorithms for Game Developers.* Course Technology. ISBN 9781584504955

Stephens, R. (2009). *Beginning Database Design Solutions.* Wiley. ISBN 9780470385494

8 Přílohy

8.1 Příloha 1 – Logický model fiktivní databáze

CUSTOMER	**Hlavní tabulka, slouží pro správu zákazníků.**
Název sloupce	**Popis**
CUSTOMERID	Jednoznačný identifikátor zákazníka v systému, jedná se o primární klíč tabulky.
FIRSTNAME	Křestní jméno zákazníka
SURENAME	Příjmení zákazníka
STREET	Součást adresy zákazníka
STREETNUMBER	Součást adresy zákazníka
CITY	Součást adresy zákazníka
PSC	Součást adresy zákazníka
PHONENUMBER	Přidělené telefonní číslo zákazníka. Ve spojení s hodnotou v sloupci CUSTORDER musí být unikátní
CUSTORDER	Pořadí zákazníka, kterému bylo přiděleno identické telefonní číslo
ACTIVE	Hodnota určuje, je-li zákazník aktivní, nebo je bývalý zákazník
STARTDATE	Datum uzavření smlouvy se zákazníkem
ENDDATE	Datum ukončení smlouvy se zákazníkem, pokud je prázdné, zákazník je stále aktivní.
TARIFID	Slouží jako cizí klíč do tabulky TARIF. Touto vazbou je přidělen zákazníkovi tarif.

Tabulka 25: Popis databázové tabulky CUSTOMER (vlastní zpracování)

TARIF	Tabulka udržuje informace o jednotlivých nabízených tarifech společnosti.
Název sloupce	**Popis**
TARIFID	Jednoznačný identifikátor tarifu, primární klíč
TARIFNAME	Název tarifu
PRICE	Měsíční cena tarifu
LOCALCALL, OUTSIDECALL, MMS, ROAMINGOUT, ROAMINGIN, SMSROAMING, MMSROAMING, DATAROAMING	Cena jednotlivých služeb poskytovaných v rámci tarifu
DATALIMIT	Datový limit konkrétního tarifu
FREEMINUTES	Počet volných minut v rámci tarifu

Tabulka 26: Popis databázové tabulky TARIF (vlastní zpracování)

CUST_DEVICE	**Tabulka udržuje seznam zařízení poskytovaných zákazníkům operátorem.** Zákazník může vlastnit 0-n zařízení. Pro tuto vazbu je použita vazební tabulka CUSTOMER_DEVICE_JT.
Název sloupce	**Popis**
DEVICEID	Jednoznačný identifikátor zařízení, primární klíč
MANUFACTURE	Výrobce poskytovaného zařízení
DEVICE_TYPE	Typové označení výrobku výrobcem
DEVICE_OS	Operační systém zařízení
DISPLAY	Velikost displeje zařízení
RAM	Velikost paměti
PRICE	Zvýhodněná cena zařízení
DEVICE_CAT	Kategorie zařízení (telefon, tablet, notebook)

Tabulka 27: Popis databázové tabulky CUST_DEVICE (vlastní zpracování)

INVOICE	**Tabulka se záznamy o fakturách za poskytované služby**
Název sloupce	**Popis**
INVOICENUMBER	Jednoznačný identifikátor faktury, primární klíč
CUSTOMERID	Jednoznačný identifikátor zákazníka, kterému byla faktura vydána. Cizí klíč do tabulky CUSTOMER na sloupec CUSTOMERID
I_MONTH	Období (měsíc), za který byla faktura vydána
I_YEAR	Období (rok), za který byla faktura vydána
I_STATE	Stav, ve kterém se faktura nachází 0 – vystavená, 4 – zaúčtovaná
I_AMOUNT	Fakturovaná částka

Tabulka 28: Popis databázové tabulky INVOICE (vlastní zpracování)

8.2 Příloha 2 – Konfigurace databázových instancí

Databázový systém Oracle

- Název databázové instance = CZU
- Znaková sada = Unicode (AL32UTF8)
- Schéma uživatel = czu
- Datový tablespace = DATA_01 (CREATE TABLESPACE DATA_01 DATAFILE 'D:\TABLESPACES\ORACLE\DATA01.DAT' SIZE 10G AUTOEXTEND ON)
- Indexový tablespace = INDEX_01 (CREATE TABLESPACE INDEX_01 DATAFILE 'C:\TABLESPACES\ORACLE\INDEX01.DAT' SIZE 10G AUTOEXTEND ON)

Následuje konfigurace instance zdrojové databáze

- Název databázové instance = CZUZDROJ
- Znaková sada = Unicode (AL32UTF8)
- Schéma uživatel = czu_z
- Datový tablespace = DATA_01 (CREATE TABLESPACE DATA_01 DATAFILE 'D:\TABLESPACES\ORACLE\DATA01Z.DAT' SIZE 10G AUTOEXTEND ON)
- Indexový tablespace = INDEX_01 (CREATE TABLESPACE INDEX_01 DATAFILE 'C:\TABLESPACES\ORACLE\INDEX01Z.DAT' SIZE 10G AUTOEXTEND ON)

Databázový systém IBM DB2

- Název databázové instance = CZU
- Znaková sada = UTF-8 TERRITORY CZ
- Schéma uživatel = czu
- Datový tablespace = CZU_DATA_01 (CREATE BUFFERPOOL CZUBP IMMEDIATE SIZE -1 PAGESIZE 8 K; CREATE REGULAR TABLESPACE CZU_DATA_01 PAGESIZE 8 K MANAGED BY DATABASE USING (FILE 'D:\TABLESPACES\DB2\DATA01.DAT' 8G) EXTENTSIZE 32 OVERHEAD 10.5 PREFETCHSIZE AUTOMATIC TRANSFERRATE 0.14 BUFFERPOOL CZUBP)

- Indexový tablespace = INDEX_01 (CREATE REGULAR TABLESPACE CZU_INX_01 PAGESIZE 8 K MANAGED BY DATABASE USING (FILE 'C:\TABLESPACES\DB2\INDEX01.DAT' 8G) EXTENTSIZE 32 OVERHEAD 10.5 PREFETCHSIZE AUTOMATIC TRANSFERRATE 0.14 BUFFERPOOL CZUBP)

Microsoft SQL server

- Název databázové instance = CZU
- Znaková sada = SQL_Czech_CP1250_CS_AS
- Schéma uživatel = czu
- Datový tablespace = DATA_01 (D:\TABLESPACES\MSSQL\DATA01.MDF SIZE 10G AUTOEXTEND; filegroup DATA)
- Indexový tablespace = INDEX_01 (C:\TABLESPACES\MSSQL\INDEX01.MDF SIZE 10G AUTOEXTEND; filegroup INDEX)

8.3 Příloha 3 – Skripty pro plnění databáze

Pro vytvoření databázových struktur pro jednotlivé databázové systémy slouží skripty v DB2_tables.zip, Oracle_tables.zip, MSSQL_tables.zip. Pro následné plnění databáze slouží skripty InsertData.zip a aplikace FillDB.zip. V souboru FillDB.properties je potřeba vyplnit přihlašovací údaje pro databázi Oracle.

8.4 Příloha 4 – Testovací aplikace

Testovací aplikace je součástí komprimovaného souboru GenerateDBLoad.zip. Pro spuštění je potřeba vyplnit přihlašovací údaje do testovaných databází. V souboru GenerateDBLoad.properties je potřeba vyplnit přihlašovací údaje pro všechny databáze.