VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

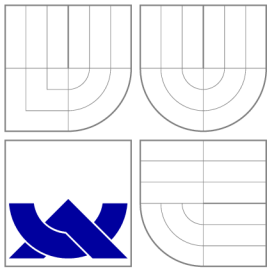
ZOBRAZOVÁNÍ DATOVÉ KOSTKY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

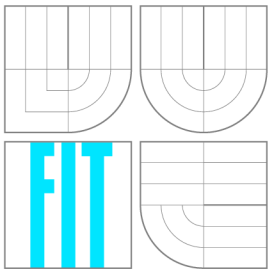
AUTOR PRÁCE
AUTHOR

Bc. PETR DITTRICH

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ZOBRAZOVÁNÍ DATOVÉ KOSTKY

DATA CUBE VISUALIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR DITTRICH

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. TOMÁŠ HRUŠKA, CSc.

BRNO 2009

Abstrakt

Tématem diplomové práce je návrh a implementace prototypu aplikace TOPZ demonstrující datové skladiště. Nejprve je v krátkosti rozebrána problematika datových skladišt a jsou navrženy možnosti vylepšení. Následuje specifikace požadavků a návrh koncepce demonstrační aplikace, na které budou vylepšení testována. V poslední kapitole je na rozebráno testování výkonnosti datového skladiště.

Abstract

The topic of the master's thesis is a concept and implementation of the prototype application TOPZ demostrating the data warehouse. All theoretical facts about the data storage space are discussed at first. It is also specified the areas of possible improvements of the data warehouse. The specification of requirements and concept of the demonstration application are described in the following part. Testing the performace of the data warehouse is discussed in last chapter.

Klíčová slova

OLAP, Realtime OLAP, ETL, datové skladiště, dimenze, multidimenzionální model

Keywords

OLAP, Realtime OLAP, ETL, data warehouse, dimension, multidimensional model

Citace

Petr Dittrich: Zobrazování datové kostky, diplomová práce, Brno, FIT VUT v Brně, 2009

Zobrazování datové kostky

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Prof. Ing. Tomáše Hrušky, CSc..

.....
Petr Dittrich
21. května 2009

Poděkování

Děkuji vedoucímu práce za cenné rady, rodině za podporu při psaní této práce a Bc. Jaroslavu Bartoňovi za pomoc při testování.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

BIOS – Basic Input-Output System – základní vstupně-výstupní systém

CORBA – Common Object Request Broker Architecture – technologie zprostředkovávající předávání požadavků mezi obecnými objekty

CSV – Comma Separated Values – čárkou oddělované hodnoty

DBMS – Data Base Management System – viz SŘBD

EFI – Extensible Firmware Interface – rozšířené firmwarové rozhranní

ETL – Extract, Transform, Load – vytáhni, převeď, načti

ETTL – Extract, Transfer, Transform, Load – vytáhni, přesuň, převeď, načti

FSB – Front Side Bus – přední systémová sběrnice

GC – Garbage Collector – sběrač nedostupné paměti

GUI – Graphical User Interface – grafické uživatelské prostředí

IDL – Interface Definition Language – jazyk pro popis rozhraní

JDBC – Java DataBase Connectivity – standardizované rozhraní pro práci s databází v Javě

JVM – Java Virtual Machine – Java virtuální stroj

MMU – Memory Management Unit – jednotka správy paměti

MVC – Model-View-Controller – Model-Pohled-Řadič – návrhový vzor

OLAP – Online Analytical Processing – systém pro analytické zpracování dat

OLTP – Online Transaction Processing – systém pro transakční zpracování dat

OMG – Object Management Group – nezisková organizace sdružující přez 800 členů

OSS – Open Source Software – software s otevřeným zdrojovým kódem

RAM – Random Access Memory – paměť s náhodným přístupem

ROM – Read Only Memory – paměť jen pro čtení

SŘBD – Systém Řízení Báze Dat

USB – Universal Serial Bus – univerzální seriová sběrnice

XML – eXtensible Markup Language – rozšiřitelný značkovací jazyk

XMPP – eXtensible Messaging and Presence Protocol – rozšiřitelný protokol pro výměnu zpráv, známý též pod označením Jabber

Kapitola 1

Úvod

Na konci minulého století došlo k velkému rozvoji informačních technologií. S tímto rozvojem úzce souvisí vytváření obrovského množství dat, která lze použít k získání užitečných informací. Přesnost těchto informací ovšem závisí na množství dostupných dat. Hodnota informací však s postupujícím časem od jejich získání a vyhodnocení výrazně klesá. Jde tedy o to, jak získaná data co nejdříve zpracovat, a tak z nich získat požadované informace co nejrychleji s maximální validitou. Zpracovávaná data lze skladovat různě, ovšem nejlepší je jejich skladování ve specializovaných datových skladištích.

Datová skladiště představují jednou z významných částí informačních systémů, které se velmi rychle rozvíjejí a mění své požadavky nejen z důvodu změn požadavků manažerů a nárůstu množství zpracovávaných dat, ale i z důvodu růstu hardwarového výkonu počítačů.

Informace z datových skladišť je nutno získávat a analyzovat. Pro tyto účely jsou používány OLAP analýzy a transformace.

Tato práce se zabývá návrhem prototypu datového skladiště, zvláště jeho optimalizací pro potřeby Realtime OLAP. Zabývá se testováním zrcadla zdrojového systému jako jednoho z prostředku urychlení promítnutí změn ve zdrojovém systému do datového skladiště.

V úvodu práce je nastíněna problematika datových skladišť, jejich dosavadní vývoj a problematika manipulace s daty (kapitoly 2 a 4). V kapitolách 5 a 6 jsou vzneseny požadavky na celkový vytvářený systém a návrh řešení jednotlivých částí systému. Následují výsledky testování chování datového skladiště (kapitola 8). V závěrečné kapitole 9 jsou nastíněna možná pokračování projektu a možná budoucí zlepšení.

V práci bylo využito poznatků z mé bakalářské práce, zvláště pak v kapitole 4, a poznatků z mého semestrálního projektu v kapitole 5.

Kapitola 2

Uvedení do problematiky

Dávno jsou již doby, kdy pouze nejsilnější udávali tempo růstu civilizace. V dnešní době jsou informace a znalosti o prostředí hybnou silou prakticky ve všech oblastech lidské činnosti (nejnázorněji v obchodu) a také nejmocnější zbraní ve válečných konfliktech. I ta nejdokonalejší technika je bez příslušných informací k ničemu. Z tohoto důvodu cena informací výrazně vzrůstá.

Na tomto místě je nutno upozornit na zásadní rozdíl mezi běžně používanými pojmy jako jsou *informace* a *data*. Pod pojmem data je chápáno cokoliv, co lze uložit. Informace jsou data s významem. Tento význam vytváří z dat informace.

Pojem *znalosti* pak označuje informace, které jsou nové a nepředvídatelné z kontextu ostatních informací. Nepředvídatelnou informaci lze definovat jako informaci, která v době vzniku nevyplývá z jiných dosud známých informací. Kromě pojmu data existuje pojem *metadata*, který označuje data popisující data. Metadata mohou být uložena stejně jako data nebo existovat jako textový či slovní popis. Metadata mohou být samozřejmě dále popsána $meta^2 daty$ atd.

Jak již byl v úvodu zmíněno s rozvojem informačních systémů, jde ruku v ruce tvorba obrovského množství dat. Může jít o protokoly funkcí zařízení, chybová hlášení všeho druhu, výsledky výkonnostních analýz, informace o dislokaci, pohybu a síle nepřítele a podobně. Většina těchto hlášení je zaprotokolována a po nějaké době bez užitku odstraněna, podobně je mnoho chybových hlášení uživateli nepřečteno a ignorováno. Přitom tato data obsahují ve většině případů mnoho hodnotných informací. Problém spočívá v tom, že informace v těchto datech obsažené nejsou přímo zjistitelné, protože jejich význam je pro počítač neznámý. K jejich interpretaci a využití je nutné mít co nejvíce dat v co možná nejaktuálnější podobě. Tato data je vhodné uložit do specializovaného skladiště, které umožní maximálně rychlou manipulaci s celým souborem dat. Tento soubor pak lze podrobit nejrůznějším analýzám, které mohou identifikovat a izolovat užitečné informace.

2.1 Ukládání dat

Úložištěm počítačových dat označujeme dle [16] komponenty počítače, zařízení a záznamová média, která udrží data užitá pro práci na počítači po požadovaný časový interval. Ukládání počítačových dat je dnes jedna ze zásadních funkcí počítače.

Základní součástí všech počítačů je paměť, která bývá spojená s centrální procesorovou jednotkou a realizuje dle [17] tzv. von Neumannovský počítačový model používaný již od čtyřicátých let.

V současném pojetí je pamětí obvykle označována polovodičová součástka realizující paměť s náhodným přístupem (RAM) případně jiné formy rychlého dočasného úložiště. Podobně úložištěm se dnes víceméně označuje velkokapacitní paměť – optické disky, magnetické pásky, pevné disky a jiné typy pamětí pomalejší než RAM. Historicky chápáno je termín paměť a úložistě užívan pro primární a sekundární úložiště dat.

2.1.1 Účel ukládání

Bylo vypracováno mnoho různých forem ukládání dat, založených na různých přírodních jevech. Žádné univerzální záznamové médium však nebylo doposud objeveno, každá forma ukládání má některé výhody, ale zároveň i nevýhody. Proto počítačové systémy obvykle využívají několik typů úložišť, každé se specifickým účelem.

Digitální počítač využívá k reprezentaci dat binárního číslicového systému. Text, čísla, obraz, zvuk a téměř každá jiná forma informací může být přeměněna na řetězec bitů nebo binární čísla. Nejvíce používanou jednotkou objemu uskladnění dat je bajt, který je roven 8 bitům. Část informace může být zpracována na libovolném počítači, jehož úložiště je dost velké pro skladování binární reprezentace informace, nebo jednoduše dat. Například s využitím osmi miliónů bitů, což je asi jeden megabajt, může běžný počítač uložit malou povídku.

Nejdůležitější částí každého počítače je centrální procesorová jednotka (CPU nebo jednoduše procesor), protože ta vlastně manipuluje s daty, vykonává většinu výpočtů a řídí všechny ostatní součásti počítače.

Bez dostatečně velké paměti by byl počítač schopný vykonat pouze fixní operace a okamžitě dodat výsledek. Poté by musel být překonfigurován pro změnu svého chování. Toto je přijatelné pro zařízení typu kalkulačka nebo pro jednoduchý digitální signálový procesor. Stroje von Neumannovské architektury se liší v tom, že obsahují paměť, ve které si uloží svůj operační kód a data. Takové počítače jsou univerzální, neboť nepotřebují mít hardware rekonfigurovaný pro každý nový program, ale mohou prostě být reprogramovány výměnou kódu v paměti. Díky tomu lze v relativně jednoduchém procesoru vytvořit postupnými výpočty komplexní výpočetní systém. Většina moderních počítačů jsou von Neumannovské architektury.

Prakticky všechny počítače používají širokou paletu pamětí, organizovanou v hierarchii kolem procesoru, jako kompromisu mezi výkonem a cenou. Paměti v hierarchii blíže k procesoru mívají větší šířku pásma a kratší přístupovou dobu. Tradičně používané rozdělení úložišť mezi primární, sekundární, terciární a offline úložistě je mimo jiné parametry charakterizovano cenou za uloženou jednotku.

2.1.2 Hierarchie úložišť

Primární úložiště

Primární úložiště, většinou označované jako operační paměť, je jediné přímo přístupné pro CPU. CPU nepřetržitě čte instrukce skladované v této paměti a vykonává tyto instrukce podle potřeby. Všechna data a instrukce aktivně používaná jsou uložena ve společné paměti.

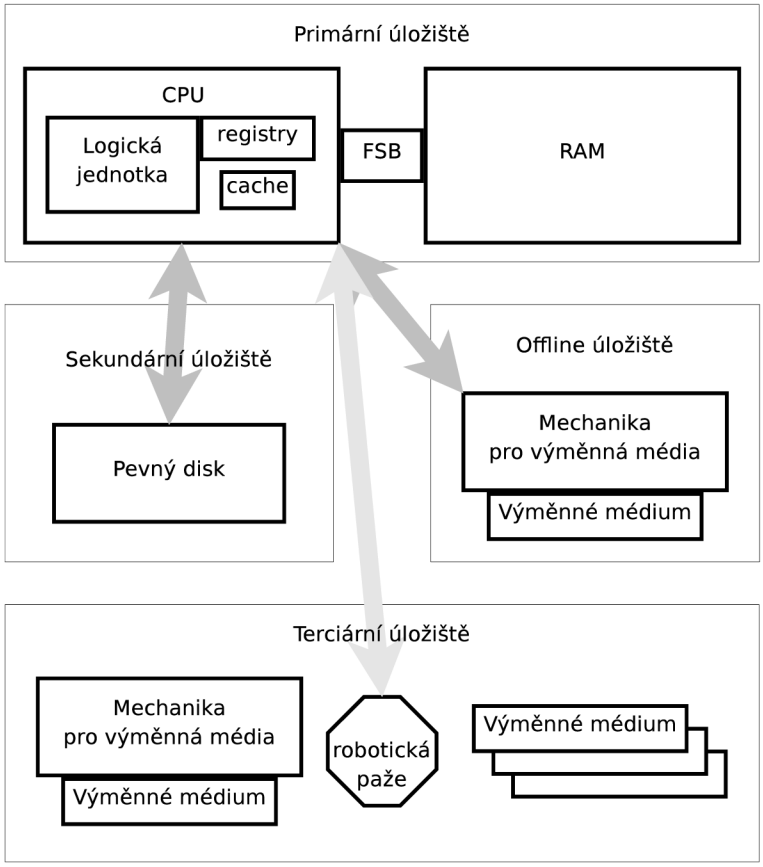
Počítače historicky používaly jako primární úložiště zpožďovací vedení, Williamsovy trubky nebo rotační magnetické bubny. V roce 1954 byly tyto nespolehlivé prvky nahrazeny magnetickou pamětí, která byla stále poněkud těžkopádná. S vynálezem tranzistoru však byla nepochybně odstartována revoluce, která ve krátké době umožnila neuvěřitelnou miniaturizaci elektronické paměti a následným rozvojem technologie křemíkových čipů.

Tento vývoj vedl k moderní paměti s náhodným přístupem (RAM), což je malá, lehká, ale zároveň nezanedbatelně drahá technologie. Zvláštní druhy RAM užité jako primární úložiště jsou nestálého typu, tj. ztratí informace po ztrátě napájení.

Jak je ukázano v digramu na obrázku 4.3, vedle hlavní operační paměti obvykle existují dvě vrstvy primárního úložiště, vedle hlavní operační paměti:

- Registry jsou technicky nejrychlejší úložiště pro ukládání počítačových dat. Registry procesoru jsou umístěny uvnitř procesoru. Každý registr si typicky pamatuje slovo (dnes většinou 32 nebo 64 bitů). Instrukce procesoru určují jaké aritmetické a logické operace se mají vykonat s těmito daty.

- Vyrovnávací paměť procesoru (tzv. *cache*) je uprostřed mezi rychlými registry a o hodně pomalejší hlavní pamětí. Představuje pouze prostředek zlepšení výkonu počítače. Nejvíce používaná data z hlavní paměti jsou jen kopírovány do rychlé vyrovnávací paměti, která umožňuje rychlejší zpracování, ale disponuje výrazně menší kapacitou. Na druhé straně je operační paměť o hodně pomalejší, ale mnohem větší než registry procesoru. Vícevrstvé hierarchické vyrovnávací paměti jsou běžně používané — primární vyrovnávací paměť (označovaná L1 cache) je nejmenší, nejrychlejší a je uložena uvnitř procesoru. Sekundární vyrovnávací paměť (označovaná L2 cache) je poněkud větší a pomalejší a je umístěna v blízkosti procesoru.



Obrázek 2.1: Diagram umístění úložišť v okolí procesoru

Hlavní paměť je buď přímo nebo nepřímo připojená k procesoru přes sběrnici paměti. Tato sběrnice bývá označována jako FSB[1]. FSB se skládá ze dvou částí, a to adresové a datové sběrnice. Vlastní operace s pamětí probíhá tak, že CPU nejdříve pošle adresu přes adresovou sběrnici, kde adresa označuje místo v paměti, na tomto místě jsou umístěna požadovaná data. Pro zasílání čtených nebo zapisovaných dat poté CPU využívá datovou sběrnici. MMU[2] je malé zařízení mezi procesorem a operační pamětí, které přepočítává aktuální pamětovou adresu. MMU tak může poskytovat abstrakci virtuální paměti nebo usnadnit přepínání úloh.

Operační paměti užité jako primární úložiště jsou nestálé a jejich obsah se ztratí po odpojení napájení. Počítač obsahující jen takové úložiště by neměl zdroj pro čtení instrukcí při startu počítače. Proto primární úložiště obsahuje malý startovací program (BIOS[3] případně EFI[4]), určený pro start počítače, který provede načtení většího programu ze stálého sekundárního úložiště do RAM a zahájí jeho vykonávání. Stálá technologie používaná pro tento účel je dle [8] nazývána ROM[5] (terminologie může být poněkud matoucí, protože většina druhů ROM je schopna náhodného přístupu).

Mnoho druhů paměti typu „ROM“ není vysloveně určeno pouze pro operace čtení. Některé paměti typu ROM umožňují i změnu jejich datového obsahu. Nicméně je nutno konstatovat, že zápisové operace jsou časově pomalé a paměť musí být před zápisem nového obsahu zcela vymazána. Některé vestavěné systémy umožňují přímé spuštění programu přímo z paměti ROM (nebo podobné paměti například EEPROM či Flash). V tomto případě odpadá nevýhoda pomalé aktualizace obsahu paměti ROM a to proto, že programy umístěné v pamětech typu ROM bývají měněny relativně velmi málo.

Paměť typu ROM nebývá používána pro ukládání běžně používaných programů a dat. To z důvodu, že běžně používané programy jsou relativně veliké a paměťové požadavky na ukládání dat bývají obrovské. Pro tyto účely je mnohem vhodnější použití sekundárních uložišť, které se také vyznačují stálostí uchování dat, ale oproti paměti typu ROM je jejich cena výrazně nižší.

Sekundární úložiště

Sekundární úložiště, neboli úložiště v populárním použití, se liší od primárního úložiště v tom, že není přímo přístupné CPU. Počítač obvykle používá své vstupně-výstupní kanály přístupu k sekundárnímu úložišti a pro přenos požadovaných dat používá vyhrazenou oblast v primárním úložišti. Sekundární úložiště nepřijde o svá data v případě výpadku napájení. Na jednotku dat je také typicky vyrazně levnější než primární úložiště. Moderní počítačové systémy mají typicky větší sekundární úložiště než primární a data pro dlouhodobé skladování jsou umístěna v sekundárním úložišti.

V moderních počítačích jsou pevné disky obvykle využívány jako druhotné úložiště. Přístupová doba k informacím uloženým na pevném disku je typicky několik milisekund. Naopak přístupová doba k informacím uloženým v paměti s náhodným přístupem je měřitelná v nanosekundách. To je důvod objasňující velmi významný rozdíl času, který rozlišuje operační paměť od rotačních magnetických paměťových zařízení. Pevné disky jsou typicky o šest řádů pomalejší než operační paměť. Rotační optická paměťová zařízení, například CD a DVD disky, mají ještě delší přístupové doby.

[1] FSB – Front Side Bus – Přední systémová sběrnice
[2] MMU – Memory Management Unit – Jednotka správy paměti
[3] BIOS – Basic Input-Output System – Základní vstupně-výstupní systém
[4] EFI – Extensible Firmware Interface – Rozšířené firmwarové rozhranní
[5] ROM – Read Only Memory – Paměť jen pro čtení

Některé jiné příklady technologií sekundárních úložišť jsou: flash paměti (např. USB[6] klíče), diskety, magnetické pásky, děrné pásky, děrné štítky nebo ZIP[7] mechaniky.

Druhotné úložiště je často zapouzdřeno souborovým systémem, který poskytuje abstrakci nutnou k organizaci dat do souborů a adresářů, poskytuje také doplňující informace (tzv. metadata) popisující vlastníka daného souboru, čas posledního přístupu pro čtení a pro zápis, povolení k přístupu a další informace.

Většina počítačových operačních systémů využívá virtuální paměť, dovolující využití větší kapacity primární paměti než je fyzicky dostupné v systému. Jak dochází k plnění primární paměti, systém přesunuje nejméně využívané části paměti (stránky) do sekundární paměti (do odkládacího nebo stránkovacího souboru) a natahuje je zpátky do paměti v případě potřeby. Čím více je přístupů do sekundární paměti, tím více je degradován celkový výkon systému.

Terciární úložiště

Terciární úložiště nebo terciární paměť, poskytuje třetí úroveň ukládání dat. Typicky je představována robotickým mechanismem, který vkládá a vytahuje vyměnitelná záznamová média podle požadavků systému. Data na uložená na terciárních úložištích jsou většinou kopírována před použitím na sekundární úložiště. Terciární úložiště bývají běžně používána pro archivní, zřídka zpřístupňované informace, protože jsou výrazně pomalejší než sekundární úložiště (např. 5–60 sekund vs. 1–10 milisekund). Terciární úložiště jsou primárně užitečné pro mimořádně velká datová skladiště, zpřístupňovaná bez lidských operátorů. Typické příklady zahrnují páskové knihovny a hudební automaty.

V okamžiku, kdy počítač potřebuje číst informace z terciárního úložiště, musí nejprve projít katalog pro určení, která páska nebo disk obsahuje informaci. Následně musí počítač navést robotickou paži pro médium a vložit ho do mechaniky. Když počítač ukončí čtění informace, robotická paže vrátí použité médium na jeho původní místě v knihovně.

Offline úložiště

Offline úložiště, také označované jako odpojitelné úložiště, je úložiště počítačových dat na médiu nebo zařízení, které není pod kontrolou procesoru. Musí být vložené nebo připojené lidským operátorem před opětovným zpřístupněním počítači. Narozdíl od terciárního úložiště nemůže být zpřístupněno bez zásahu člověka.

Offline uskladnění je obvykle využíváno k přenosu dat, protože po odpojení může být médium snadno fyzicky transportované. Dále v případě pohromy, například ohně, který zničí originální data, médium ve vzdálené lokalitě bude s velmi vysokou pravděpodobností nedotčené, a umožňuje tak zotavení z pohromy. Offline úložiště zvětší bezpečnost obecné informace poté, co je fyzicky nedostupné z počítače. Zajistí také konzistentnost a integritu dat, protože nemohou být ovlivněny počítačovým útokem. Jestliže informace uložené pro archivní účely jsou zpřístupňovány zřídka nebo nikdy, offline úložiště je levnější než terciární úložiště.

V moderních osobních počítačích je většina sekundárních a terciárních záznamových médií také využitelná pro offline ukládání. Optické disky a flash paměti jsou pro tyto účely velmi populární. V podnikových použitích převládá využití magnetické pásky. Dříve se pro

[6] USB – Universal Serial Bus – Univerzální seriová sběrnice

[7] ZIP je registrovanou ochrannou známkou firmy Iomega Corporation

tyto účely využívalo například zejména disket různých formátů, Zip disků nebo děrných štítků.

2.1.3 Formát uložení dat

Nejjednodušší možností uložení dat představuje jejich uložení do *souboru na pevný disk*, který lze libovolně strukturovat. Takto realizované uložení dat má tu výhodu, že je naprosto bez problémů přenositelné mezi různými systémy zpracovávajícími data. Mezi formáty, které lze doporučit pro strukturu souboru pro uložení na pevný disk, je nutno zejména zmínit *prokotolový soubor* (běžně také označovaný jako *logfile*). Protokolový soubor obsahuje na každém řádku jeden záznam v definovaném formátu, většinou uvozený informací o čase a zdroji a zakončený podrobnostmi. Tento soubor nemá hlavičku a bývá často v čitelném formátu (tzn. lze ho přímo zobrazit na obrazovce). Tento formát je využíván především pro chybové protokoly nebo pro soubory s triviální strukturou (například čas, původce, popis).

Pro strukturovaná data je také využitelný formát *CSV*[8]. Formát CSV bývá využíván jako úložiště velkých objemů pevně strukturovaných dat. Tento formát má volitelnou hlavičku, která se může nacházet před prvním záznamem na prvním řádku souboru. Na každém řádku, se podobně jako v případě protokolového souboru, nachází jeden záznam. Tento záznam má však pevněji definovanou strukturu. Jednotlivá pole jsou oddělena – jak už je z názvu patrné – čárkou případně středníkem. Pro případ existence čárky, resp. středníku uvnitř datového pole, existuje možnost pole ohraničit uvozovkami případně dvojitými uvozovkami.

Další, v poslední době značně využívaný formát, je *XML*[9]. Formát XML bývá využíván u velmi složitých datových struktur, nehodí se naopak u velkých souborů dat, která mají relativně jednoduchý a stále stejný formát. Formát XML má výhodu v možnosti uložit i složitější strukturu dat. Soubor může mít v hlavičce uloženo kodování textových polí, což usnadňuje přenositelnost na maximální možnou míru. Další výhodou formátu XML je dle [4], že jeho struktura může být popsána v externím souboru a lze tak usnadnit detekci vadného formátu souboru. Jako formátu pro popis struktury XML souboru může být použito DTD[10], případně XSD[11], který je také ve formátu XML. Toto může usnadnit kontroly interoperability mezi systémy.

Jinou možností uložení dat je formou *databáze*. Mezi v současnosti používané databáze lze dle [11] řadit *relační databáze*, které umožňují modelovat vztahy mezi daty a nabízí možnost mnoha transformací dat a rozličných dotazů nad daty. Mezi jinými databázemi lze zmínit *objektovou databázi*, která ale není příliš využívána pro její zaměření spíše na modelování procesů.

Poslední a asi nejvyužívanější možností formátu uložení dat jsou *specializovaná skladiště dat*. Tato specializovaná skladiště jsou nazývána dle jejich obsahu jako datová skladiště a jsou většinou optimalizována pro dotazování nad značnými objemy dat. Jako základ bývají využívany relační databáze případně soubory ve formátu CSV. Nad těmito úložišti jsou vybudovány *abstraktní vrstvy*, které umožnují odstínění uživatele od specifického uložení, ale dovolí uživateli s těmito daty manipulovat. Relační databáze je používána často pouze v první *normální formě*. Normální forma je pojem vyjadřující míru *normalizace* relační da-

[8] CSV – Comma Separated Values – čárkou oddělované hodnoty
[9] XML – eXtensible Markup Language – rozšiřitelný značkovací jazyk
[10] DTD – Document Type Definition – Definice typu dokumentu
[11] XSD – XML Schema Definition – XML popis schématu XML

tabáze a byla definována E. F. Coddem. První úroveň normalizace dle [1] požaduje pouze, aby každý sloupec obsahoval maximálně jednu hodnotu. Díky tomu mohou v databázi vznikat redundance a zvyšuje se tak možnost vzniku nekonzistencí. Protože však toto skladiště modifikuje pouze jeden proces, lze tento nedostatek považovat za nepodstatný problém. Výhodou tohoto přístupu je, že tyto systémy umožňují optimalizovat uložení pro rychlost přístupu. Rychlost přístupu k datům uloženým v relační databázi v první normální formě spočívá ve snížení počtu míst, kde jsou data rozprostřena. Většinou jde pouze o jednu tabulku s daty a několik tabulek určených k vyhledání dat. Správné strukturování vyhledávacích tabulek však není triviální záležitostí, protože vyžaduje detailní znalost povahy ukládaných dat.

2.2 Relační databáze

Za otce relačních databází je považován Edgar Frank Codd, který v roce 1970 dle [14] načrtl možnost, jak použít relační kalkul a algebru i pro netechnické uživatele při ukládání a manipulaci s daty. Dle jeho pojetí k tomu mělo být používáno srozumitelných příkazů vycházejících z běžné angličtiny, tedy ne specifický jazyk, který vypadá jako revoluce nepísmenových značek, jak to bývá u některých jiných jazyků. Už tato původní koncepce předpokládala ukládání dat do tabulek. Tento návrh byl založen na nezávislosti dat na použitém hardware, na způsobu jejich fyzického uložení, dále pak na přístupu k datům pomocí neprocedurálního jazyka. Uživatel měl mít možnost specifikovat operaci nad jím definovanou množinou dat namísto pouhé manipulace s jedním záznamem.

Relační databáze je tedy systém určený pro uložení dat a manipulaci s těmito daty. Pro přístup k datům využívá programu označovaného jako SŘBD[12] (v anglické literatuře označovaného jako DBMS[13]). Programy SŘBD pro organizaci dat a jejich správu využívají relační model dat. Název tohoto modelu vychází z relační algebry, což je matematický aparát, na kterém relační model dat staví. V tomto modelu jsou údaje uspořádány do tabulek. Tabulka je dvourozměrný útvar, jehož řádky označují záznamy a sloupce obvykle označují v databázové terminologii položky nebo atributy. Obsah tabulky lze popsat jako relaci a relaci jako uspořádanou multimnožinu n-tic prvků.

Dle relační teorie lze pomocí základních operací (sjednocení, kartézský součin, rozdíl, selekce, projekce a spojení) uskutečnit veškeré operace s daty a ostatní operace jsou již jen kombinacemi těchto pěti operací.

[12] SŘBD – systém řízení báze dat
[13] DBMS – Data Base Management System

2.3 PostgreSQL

PostgreSQL je dle [10] nejvyspělejší databázový systém, který byl uvolněn jako OSS[14]. OSS představuje inovativní přístup k vývoji programů s pomocí sítě Internet. OSS projekty nevyvíjí žádná komerční společnost, ale mnoho schopných vývojářů připojených k Internetu. Přispívání nápady, kódem a jakoukoliv pomocí je naprosto dobrovolné a je otázkou cti pro vývojáře.

Předchůdcem projektu PostgreSQL byl systém Ingres vyvíjený na univerzitě v Berkley v letech 1977 až 1985. V Berkley působil rovněž Michael Stonebraker, který vedl tým vyvíjející objektově–relační databázový server Postgres. Projekt v letech 1994 až 1995 doplnili Jolly Chen a Andrew Yu o podporu jazyka SQL a projekt označili jako Postgres95.

V průběhu roku 1996 byl vývoj přesunut na Internet, zdrojové kódy byly umístěny na CVS server. CVS server je server určený pro sdílení zdrojových kódů a podporu týmového vývoje. V průběhu roku 1996 se na vývoji podíleli nejvíce Marc Fournier, Thomas Lockhart, Vadim Mikheev a Bruce Momjian, všichni v rámci svého volného času. Na sklonku roku 1996 se název změnil na PostgreSQL, což byl sice název poněkud dlouhý, ale skrýval v sobě odkaz na původní projekt z Berkley a zároveň také jeho podporu jazyku SQL. Kód měl v této době řádově dvě stě padesát tisíc řádků zdrojového kódu. Ačkoliv byl systém PostgreSQL nástupcem univerzitního projektu, byl velmi dobře rozdělen na moduly, protože byl využíván v rámci mnoha výzkumých projektů.

[14]OSS – Open Source Software – Software s otevřeným zdrojovým kódem

Kapitola 3

Použité technologie

3.1 Java

Použitý programovací jazyk Java[1] je jazyk původně vymyšlený Jamesem Goslingem v Sun Microsystems v roce 1995. Hlavní požadavky při vývoji jazyka JAVA byly dle [3] následující:

- jednoduchost, objektovost a přívětivost,
- architekturní neutrálnost a přenositelnost,
- robustnost a bezpečnost,
- vysoká výkonnost,
- měl být interpretovaný, dynamický a pracovat s více vlákny.

Pro dosažení jednoduchosti a přívětivosti Javy byl jako základ použit jazyk C++, ze kterého byly odstraněny všechny problematické části, například preprocesor a práce s ukazateli, protože tyto části zapříčiňovaly nejčastější problémy programů a chyby programátorů. Díky tomu je pro programátora, který v C++ pracoval, přechod velmi jednoduchý a nový programátor se nemusí učit mnoho konstrukcí.

Práce s ukazateli byla zjednodušena objektovým přístupem. Vše v Javě je objekt s výjimkou několika jednoduchých typů s pevně danou velikostí, konkrétně celočíslené 8bitový `byte`, 16bitový `short`, 32bitový `int`, 64bitový `long`, pro čísla s plovoucí řádovou čárkou 32bitový `float`, 64bitový `double`, a pomocné 16bitový `char` – znak a `boolean` pro logickou hodnotu. Robustnost zajišťuje GC[2], který se stará o vracení paměti, díky čemuž nedochází ke spotřebovávání paměti tzv. *memory leakům*. Memory leak je programátorská chyba, při které dochází ke spotřebovávání paměti a po určitém čase k jejímu vyčerpání a pádu programu, nebo v horším případě celého systému.

Díky pevně stanoveným délkám jednoduchých typů a použití interpretovaného *bajtkódu* je možné přenášet programy mezi operačnímy systémy a také mezi architekturami procesorů. Programy v Javě jsou překládány do bajtkódu. Bajtkód je posloupnost instrukcí optimalizovaná pro speciální software, který je schopen přeložit bajtkód do instrukcí pro procesor. Díky využití speciálního softwaru nazývaného JVM[3], lze spustit program na všech

[1] Java je registrovanou obchodní známkou firmy Sun Microsystems, Inc.
[2] GC – Garbage Collector – sběrač nedostupné paměti
[3] JVM – Java Virtual Machine – Java virtuální stroj

operačních systémech a architekturách procesorů, pro které je tento speciální software k dispozici. Dynamickým natahováním je docíleno zrychlení startu většiny aplikací, protože načítání probíhá průběžně dle potřeby programu, díky tomu lze také snížit paměťovou náročnost, protože nepoužité části programu není nutné natahovat do paměti.

Bezpečnost programů v Javě je zajištěna dynamičností bajtkódu a také povinnou typovou kontrolou. Je tak zabezpečeno, že není možné způsobit přetečení nebo podtečení zásobníku či předání chybného množství parametrů. Mezi další prvky zajišťující bezpečnost lze řadit řízení přístupu a vynutitelné kryptografické podepsání bajtkódu.

Java je vybavena prostředky pro práci ve více vláknech a pro synchronizaci těchto vláken. Vlákna jsou odlehčenou variantou procesů, na rozdíl od kterých sdílejí adresový prostor. Výkonnost programů spuštěných v JVM je ovlivněna nutností překladu do strojového kódu aktuální platformy, nicméně rychlost programů je srovnatelná s programy v jazyce C či C++.

3.2 CORBA

CORBA[4] je standard definovaný OMG[5], který umožňuje softwarovým komponentám napsaným v různých jazycích, umístěným na různých počítačích, komunikovat mezi sebou.

CORBA[6] nabízí mechanizmus pro standardizaci volání metod. CORBA verze 1.0 byla vydána v říjnu roku 1991. CORBA využívá IDL[7] k popisu rozhraní spravovaných objektů. Existuje také mnoho překladačů, které umožňují namapování CORBA volání do standardních jazyků jako například C, C++, Java, Cobol, Smalltalk a mnoha dalších.

[4]CORBA – Common Object Request Broker Architecture – technologie zprostředkovávající předávání požadavků mezi obecnými objekty

[5]OMG – Object Management Group, Inc. – nezisková organizace sdružující přez 800 členů

[6]CORBA je registrovanou obchodní známkou Object Management Group, Inc.

[7]IDL – Interface Definition Language – jazyk pro popis rozhraní

Kapitola 4

Systém pro analytické zpracování dat OLAP

OLAP[1] lze dle [6] definovat jako přístup k analýzám velkých množství dat. Za protipól OLAPu lze považovat *OLTP*[2]. Hlavním rozdílem mezi OLAP a OLTP systémy je jejich orientace. Zatím co OLTP je orientován na *manipulace s daty*, OLAP je orientován na *objekty zájmu*. Objektem zájmu může být cokoliv, co nás zajímá, například položka prodeje, faktura, měření fyzikální veličiny a podobně.

Příroda okolo nás je dle [7] vícerozměrná. Díky této skutečnosti jsou i data, která počítačové systémy zpracovávají, vícerozměrná. Pro potřeby analýzy a zpracování dat je třeba jednoznačně definovat strukturu uložených dat. Data určená pro analýzy se skládají z fakt a jejich atributů.

Data v OLTP systémech jsou rozprostřena v tabulkách a v jejich vazbách.

V OLAP systémech je primární dělení dat do tzv. *OLAP kostky*, případně vícerozměrných kostek, které vždy pokrývají právě jeden objekt zájmu. OLAP kostka se skládá většinou z *tabulky faktů* a *tabulek dimenzí*, které obsahují atributy určené k vyhledávání či popisu dat. Dimenzní tabulky mohou být navázány buď pouze přímo na tabulku faktů, nebo nepřímo k tabulce faktů přes další dimenzní tabulky. Tato uspořádání se nazývají schémata. Mezi nejčastěji použitá schémata patří *schéma hvězda*, kde jsou všechny dimenzní tabulky navázány na tabulku faktů, nebo *schéma sněhová vločka*, kde jsou některé dimenzní tabulky navázány na jiné dimenzní tabulky.

4.1 Zobrazení metodou OLAP

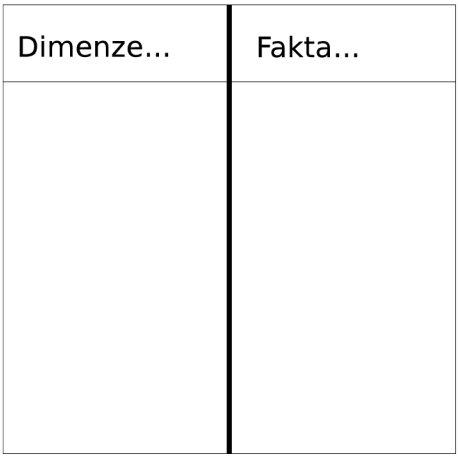
Nejčastějším způsobem komunikace uživatele s OLAP systémem je prostřednictvím *relační tabulky* či *kontingenční tabulky*. Méně rozšířeným, i když velmi přehledným způsobem komunikace je komunikace prostřednictvím *kontingenčního grafu*. Každá z těchto metod má svá využití a nelze proto některou z nich vyloučit.

Relační tabulka zobrazuje v každém řádku jeden fakt. Tabulka je rozdělena svisle na dvě části viz obrázek 4.1. Ve sloupcích v levé části jsou uvedeny jednotlivé dimenze, v pravé části potom hodnoty z tabulky faktů. Tento způsob je výhodný zvláště při řídkém obsazení dat mezi dimenzemi, protože jsou zobrazeny jen existující fakta. Při zobrazení kontingenční tabulkou lze opět jednoduše a názorně zjišťovat závislosti mezi více dimenzemi. V záhlaví

[1] OLAP – On Line Analytical Processing – systém pro analytické zpracování dat.

[2] OLTP – On Line Transaction Processing – systém pro transakční zpracování dat

sloupce a na začátku řádku jsou uvedeny jednotlivé dimenze, uvnitř kontingenční tabulky zase hodnoty z tabulky faktů. V kontingenčním grafu jsou nejlépe detekovatelné trendy vývoje či různé anomálie. Kontingenční graf je většinou výsledkem transformace z kontingenční tabulky pro zvýraznění určitého trendu.



Obrázek 4.1: Schéma struktury relační tabulky

4.2 OLAP operace

OLAP systém umožňuje dle [5] většinou několik základních operací s daty, například:

- *Drill–down* – jestliže se zvolená položka dané dimenze nenachází na nejnižší úrovni v hierarchii této dimenze, dojde k přesunu na tuto nižší úroveň.
- *Roll–up* – operace opačná k operaci zanoření, je-li to možné, je zvýšena úroveň pohledu v rámci dané dimenze,
- *Dice* – vybere pouze ty údaje, které splňují zadanou podmínku. Úroveň dimenzí ani jejich počet se nezmění.
- *Slice* – výběr jediného fixního prvku dané dimenze, čímž dojde k výběru pouze těch údajů, které mají průsečík s vybranou hodnotou dané dimenze a těmi, které byly zvoleny před touto operací.
- *Pivot* – při zobrazení dat v kontingenční tabulce dojde k otočení tak, že dimenze umístěné do sloupců jsou přemístěny do řádků a naopak. Tím dojde ke změně umístění jednotlivých hodnot a může být docíleno přehlednějšího zobrazení dat.
- *Drill through* – dojde k zanoření až na konkrétní neagregované údaje, zde může být proveden přístup přímo do zdrojového systému.

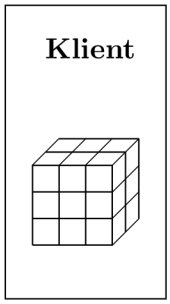
4.3 Ukládání dat v systémech OLAP

OLAP systémy označují systémy pracující s datovými skladišti a realizující OLAP operace. Systémy realizující OLAP dotazy jsou principielně děleny na:

- desktop OLAP,
- multidimenzionální OLAP,
- relační OLAP a
- hybridní OLAP.

4.3.1 Desktop OLAP

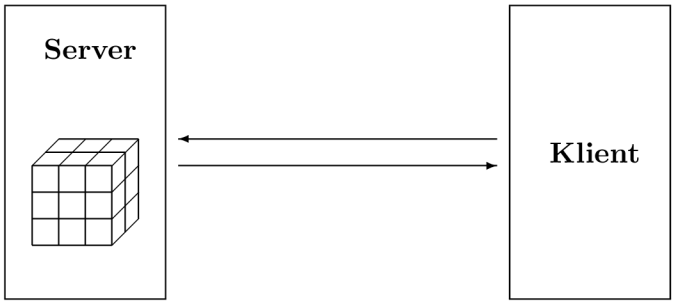
Desktop OLAP je nejjednodušší systém realizující OLAP, běžící na jediném klientském počítači. Desktop OLAP se vyznačuje limitací své velikosti, protože jde o systém běžící celý na počítači uživatele, viz obrázek 4.2. Ten většinou nedisponuje velkými systémovými prostředky v porovnání se servery. Bývá používán jen v opravdu vyjimečných případech. Jedním z těchto případů může být například offline manipulace s malou částí velkého OLAP systému.



Obrázek 4.2: Schéma rozprostření vrstev systému Desktop OLAP

4.3.2 Multidimenzionální OLAP

Multidimenzionální pole je datová struktura, která je určena pro uložení dat. Je implementována jako pole s n–rozměry a přesně kopíruje strukturu kostky. Může také s výhodou obsahovat různé agregace, které mohou urychlit manipulace s daty uloženými v kostce. Díky tomu je způsob uložení optimalizován pro OLAP operace, má tudíž velmi vysoký výkon při vyřizování dotazů a vyžaduje relativně malý prostor pro uložení obsahu kostky. Je vhodný pro malé až středně velké datové sklady. Některá řešení mívají však problémy s větším množstvím dimenzí než cca 10, záleží samozřejmě na kardinalitě dimenze. Pokud však lze kostku celou načíst a manipulovat s ní v paměti serveru, nemá toto řešení výkonostní konkurenci. Multidimenzionální OLAP je nejčastěji realizován jako dvouvrstvý OLAP systém, který udržuje všechna data ve speciálním formátu, tzv. *multidimenzionálním poli*. Schéma rozprostření vrstev je uvedeno v obrázku 4.3.



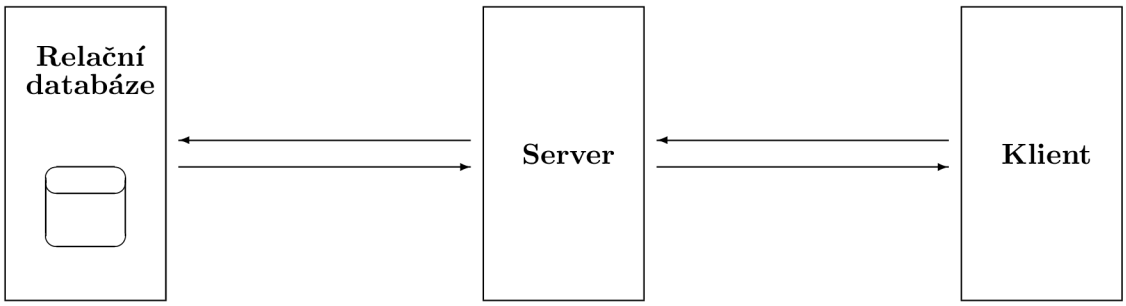
Obrázek 4.3: Schéma rozprostření vrstev systému Multidimenzionální OLAP

V obrázku 4.3 je znázoněno, že server uchovává veškerá data v multidimenzionálním poli.

4.3.3 Relační OLAP

Relační OLAP je datové skladiště, které jako úložitě využívá relační databázi. Toto řešení již není limitováno velikostí zpracovávaného datového souboru. Ten může růst do velikosti desítek gigabajtů až stovek terabajtů dat. Při zpracování takto velkých objemů dat nesmí být opomenuta skutečnost, že relační databáze mívají cca o jeden řád větší alokovaný diskový prostor, než je objem dat v nich uložený. Z tohoto důvodu bývá řešení koncipováno jako třívrstvé. Větší problém zde ale nastává při importu dat do relačního OLAPu (tzv. natahování dat do skladiště). Pro tuto operaci relační databáze neobsahuje žádný nástroj. Z tohoto důvodu je nezbytné doplnit serverovou vrstvu tak, aby tyto činnosti zajišťovala. Kapitola 4.4 se zabývá problematikou natahování dat do datových skladišť.

Narozdíl od multidimenzionálního OLAPu není u relačního OLAPu problém s konzistencí dat v případě výpadku serveru. Většina relačních databází má velmi dobře vyřešeno obnovení systému po pádu. Relační OLAP využívá mnoho velkých hráčů na trhu a řešení je nejčastěji organizováno tak, jak je patrno z obrázku 4.4.

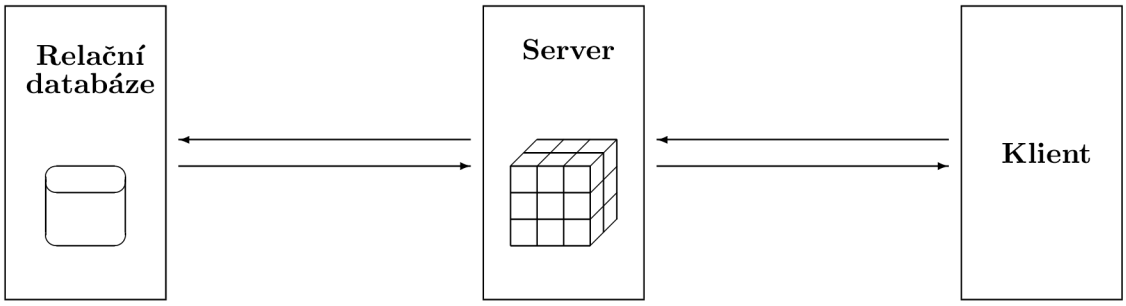


Obrázek 4.4: Schéma rozprostření vrstev systému Relační OLAP

V obrázku 4.4 je znázorněno, že data jsou umístěna v databázi a server neuchovává žádná data. Server pouze zprostředkovává data klientovi.

4.3.4 Hybridní OLAP

Hybridní OLAP představuje nejsofistikovanější OLAP. Hybridní OLAP spojuje relační a multidimenzionální přístup, snaží se eliminovat jejich nedostatky, například složité řešení konzistence multidimenzionálního OLAPu, a maximalizovat jejich výhody rychlost manipulací s multidimenzionálním OLAPem. Spojení relačního a multidimenzionálního přístupu řešení je velmi efektivní, ale také nejsložitější, protože vyžaduje udržovat všechna data aktuální. Výhoda spočívá ve skutečnosti, že hybridní OLAP udržuje všechna data v datovém skladišti realizovaném jako relační databáze a jejich agregace udržuje v multidimenzním poli (viz obrázek 4.5). Lze jej s výhodou použít v případě velkého datového skladiště, ze kterého je využívána vždy pouze dílčí část jako výřez pro dotazování a analýzy. Tato dílčí část může také být při otevření kostky zkopírována do multidimenzního pole, kde manipulace trvají řádově kratší dobu.



Obrázek 4.5: Schéma rozprostření vrstev systému Hybrid OLAP

V obrázku 4.5 je znázoněno, jak jsou data v relační databázi uložena a jak server uchovává jejich část, případně jejich agregace v multidimenzionálním poli.

4.4 Plnění datových skladišť

Značný problém představuje plnění datových skladišť. Důvodem je, že většina zdrojů dat není standardizována, nelze tudíž ani snadno standardizovat získávání těchto dat, ani jejich formát. Z těchto důvodů je nutno považovat získávání dat a jejich transformaci na standardní formát za netriviální operace. Tyto operace náleží do ETL[3] či ETTL[4] skupiny. Pro tyto skupiny operací existuje řada nástrojů. Jejich vstupem bývá zdrojový systém prostřednictvím:

- přímého přístupu do databáze,
- přístupu do databáze prostřednictvím pohledu (například z důvodu bezpečnosti),
- souboru na souborovém systému,
- vzdáleného volání,

[3]ETL – extract, transform, load – vytáhni, převeď, načti
[4]ETTL – extract, transfer, transform, load – vytáhni, přesuň, převeď, načti

- zprávy s využitím komunikačního protokolu (například elektronickou poštou, pomocí protokolu XMPP[5], známým též pod označením Jabber[6] ... apod.).

Převod dat ze zdrojového systému bývá realizován jako sada zřetězených transformací. K tvorbě řetězu transformací je zapotřebí experta, který zná povahu dat. Tyto zřetězené transformace mohou být vizualizovány do orientovaného grafu, který modeluje datové toky.

Řetěz transformací je zpravidla zahájen rozložením vstupu na jednotlivé záznamy, se kterými se posléze pracuje odděleně, což lze velmi efektivně využít k paralelizaci zpracování.

Výsledkem ETL operací je většinou tabulka, která obsahuje data v standardizovaném formátu. Kvalita ETL operací je nutnou podmínkou pro získání kvalitních informací a znalostí, nicméně není jedinou podmínkou, jako další lze například zmínit dobu zpracování dat.

Pokud mají být informace získané z dat efektivně využity, musí být k dispozici pro zpracování co nejdříve po jejich vzniku. Z tohoto důvodu je nutné minimalizovat zpoždění mezi vznikem dat a jejich zobrazením v datovém skladišti. Další urychlení reakce zpracování dat na vznik nových dat může být provedeno informováním dalších navazujících systémů, například OLAP nástrojů, na změnu v datech.

4.5 Nepřipravené systémy

Ne všechny systémy umožňují výstup dat ve formátu vhodném pro další zpracování a pro získávání informací a znalostí. Některé systémy nejsou schopny vytvořit iterativní výstup, který lze zpracovávat postupně. Je proto nutné po každé změně dat ve zdrojovém systému provést rekonstrukci dat v datovém skladišti. To může být problém v případě, že existuje snaha udržovat opravdu aktuální stav dat v datovém skladišti a současně je objem zpracovávaných dat značný. Existují například systémy obsahující jednotky gigabajtů až desítky terabajtů dat.

4.5.1 Zrcadlo zdrojového systému

Jednou z variant řešení problému s rekonstrukcí dat u nepřipravených systémů je v rámci ETL operací je udržování kopie tzv. zrcadlo zdrojového systému, která bude synchronizována výstupy ze zdrojového systému. Následně je možno jako jeden z výstupů synchronizace vytvořit protokol o změnách, který lze dále využít v rámci natahování dat do datového skladiště. Tento proces může být na první pohled značně zdlouhavý, ale dle [12] jej využívá například SAP BW[7]. V rámci kapitoly 8 je prezentována metodika výkonových testů udržování zrcadla zdrojového systému, natahování kompletního souboru dat do datového skladiště a natahování přírůstků s pomocí protokolu zrcadla zdrojového systému.

4.5.2 Výhody užití zrcadla zdrojového systému

Udržování zrcadla zdrojového systému přináší výhodu možného oddělení serveru pro natahování dat do datového skladiště a serveru pro dotazování a získávání informací z dat. Další výhodou je možnost kontroly natahovaných dat a jejich úpravy či provedení ETL operací. Zvláště při dlouhodobém využívání datového skladiště je toto jedna ze zásadních

[5]XMPP – Extensible Messaging and Presence Protocol – rozšiřitelný protokol pro výměnu zpráv
[6]Jabber je registrovanou obchodní známkou firmy Jabber, Inc.
[7]SAP BW – je registrovanou ochrannou známkou firmy SAP AG

výhod. Požadavky na data v datovém skladišti se vždy s časem vyvíjejí a mění se často i kvalita či podrobnost informací získávaných z produkčních systémů. Na tento jev je nutné občas reagovat modifikací dat v datovém skladišti. Modifikace historických dat patří k jedné z nejobtížnějších operací při údržbě datového skladiště vůbec.

Jinou výhodou užití zrcadla zdrojového systému je možná paralelizace natahování dat do datového skladiště, která v případě natahování kompletního stavu zdroje do datového skladiště není možná (z důvodu přetížení diskového subsystému serveru, neboť většina datových skladišť svojí velikostí přesahuje velikost operační paměti serverů).

Další výhodou využití zrcadla zdrojového systému je možnost oznámení změny datového skladiště pouze těm prvkům, které s těmito daty pracují. Tím se omezí plýtvání výpočetního výkonu pro zajištění zobrazení nových dat u uživatelů či nástrojů, které nepatří mezi zasažené aktuální změnou dat.

4.5.3 Úskalí užití zrcadla zdrojového systému

Využití zrcadla zdrojového systému způsobuje zvětšení nároků na diskový prostor. Jako pozitivum lze označit snížení zatížení zdrojového systému.

Možné další úskalí může spočívat ve zpomalení průchodu dat datovým skladištěm a vznikem prodlevy při tvorbě zrcadla zdrojového systému. Velikost prodlevy bude měřena v kapitole 8.

4.6 Realtime OLAP

Zobrazení co možná nejčerstvějších dat je u systémů pro získávání informací a znalostí jednou ze základních podmínek. Realtime OLAP staví tuto podmínku jako hlavní požadavek. Pokud by systém byl těsně spojen se systémem, ve kterém data vznikají, je možno realtime datové skladiště realizovat snadno. Bohužel většina dodavatelů informačních systémů nepočítá se spojením informačního systému se systémem pro získávání informací a znalostí. Pokud je v informačním systému počítáno s podporou takového systému, velmi často chybí v předávaných datech alespoň části dat, které jsou dostupné pouze v offline formě. Proto lze považovat realtime datové skladiště za jistý ideál, ke kterému existuje snaha se maximálně přiblížit.

Dle [15], je Realtime OLAP vystavěn na architektuře multidimenzionálního OLAPu. Toto řešení je, jak již bylo v 4.3.2 zmíněno, limitováno velikostí operační paměti serveru. Pro potřeby však Realtime OLAPem rozumíme OLAP operace nad datovým skladištěm, které se velmi těsně blíží realtime datovému skladišti. Realtime datové skladiště není požadováno přímo, protože OLAP operace ve většině případech trvají řádově jednotky sekund až desítky minut, dle složitosti operace. Díky tomu je tolerováno zpoždění průchodu dat datovým skladištěm cca několik desítek sekund až jednotek minut. Pokud jsou tyto operace řízeny uživatelem, je zde požadavek, aby se do výsledků operací nad daty projevily všechny změny v datech do odeslání posledního příkazu.

4.7 Operace nad daty

Pro potřeby analýzy výkonnosti a efektivnosti datového skladiště je nezbytné primárně zjistit, jaká data se ukládají do datového skladiště a jaké operace jsou prováděny nad těmito

daty v datových skladištích. Data v datovém skladišti bývají složena ze sady *atributů* (podstatných znaků či neodlučitelných vlastností). Atributy lze dle [18] rozdělit na tři základní druhy:

- *diskrétní atribut* – u něho je obor hodnot spočetný,
- *ordinální atribut* – je rozšíření diskrétního atributu o uspořádání hodnot, které může atribut nabývat,
- *kvalitativní atribut* – někdy také numerický či spojitý, je atribut nabývající číselných hodnot, jejichž obor je nespočetný.

Operace nad atributy mohou být sumační nebo analytické. Výsledkem sumačních operací je číselná hodnota tzv. *míra*. Ty lze rozdělit dle rozdělitelnosti algoritmu do tří kategorií:

- *distributivní míra* – míra, jejíž výpočet lze rozložit rozdělením souboru atributů na části, spočítáním dílčích výsledků těchto částí a jejich spojením,
- *algebraická míra* – míra, kterou lze spočítat algebraicky z jiných algebraických nebo distributivních měr,
- *holistická míra* – míra, kterou lze získat pouze výpočtem nad celým souborem dat.

Pokud jsou zanedbány holistické míry, lze při výpočtu distributivních nebo algebraických měr využít data k libovolnému okamžiku a následně dopočítávat a zpřesňovat výsledky pomocí zrcadla zdrojového systému, viz 4.5, konkrétně z dat z protokolu o změnách. Tím lze opravovat míry dat v datovém skladišti velmi rychle, protože velikost změn je ve srovnání s velikostí datového skladiště zanedbatelná.

Kapitola 5

Požadavky na vytvářenou aplikaci

Pro otestování chování OLAPu s použitím zrcadla zdrojového systému bude vytvořena aplikace implementující datové skladiště a zrcadlo zdrojového systému. Aplikace bude provozovatelná na běžných platformách Windows[1], Linux[2] a Mac OS[3] X. Pro komunikaci komponent tvořících aplikaci bude použita síťová infrastruktura s protokolem TCP/IP, využívaným v celosvětové síti Internet.

V aplikaci bude použito natažení dat do datového skladiště *plným natažením* a *natažením přírůstků* s použití zrcadla zdrojového systému. Plné natažení bude probíhat vyprázdněním datového skladiště a následně natažením celého datového souboru do datového skladiště. Natažení přírůstků bude využívat protokol o změnách zrcadla zdrojového systému a bude natahovat pouze změny od posledního natažení datového skladiště.

V rámci řešení nebude mít uživatel přímý přístup ke kostce a bude od ní oddělen pomocí *pohledu na kostku*. Uživatel tak musí pro manipulaci s daty nejdříve tento pohled vytvořit. Pohled bude potom možno pomocí OLAP operací (viz 4.2) modifikovat. Systém bude pokrývat případy použití znázorněné v obrázku 5.1.

5.1 Složitost a cena

Cena je v dnešní době hlavním kritériem hodnocení většiny lidských činností. Předpokládejme, že cena každého výpočtu je dána dobou jeho zpracování na počítači, pak se toto rozložení ceny nazývá uniformní. Uniformní rozložení považuje všechny operace trvající stejnou dobu jako stejně náročné.

Analýzu složitosti výpočtu lze dle [19] chápat jako vyjádření požadovaných zdrojů pro tento výpočet jako funkci závisející na délce vstupního řetězce.

Přesná informace o složitosti je pro naše potřeby většinou zbytečná. Urychlení výpočtu díky využití výkonnějšího počítače je jednoduché. Charakter růstu náročnosti výpočtu ale zůstává zachován. Zajímá nás pouze „důležitá“ část informace o složitosti. Proto se často složitost popisuje pomocí tzv. *asymptotických odhadů složitosti*. Nejvyužívanější odhad je asymptotické horní omezení funkce O.

Nechť $\mathcal{F}$ je množina funkcí $f : \mathbb{N} \to \mathbb{N}$, které vyjadřují projekci délky vstupu na hodnotu složitosti algoritmu. Pro danou funkci $g(n) \in \mathcal{F}$ definujeme množinu funkcí $O(f(n))$ vzorcem 5.1.

[1] Windows je registrovanou obchodní známkou Microsoft Corporation, společnosti organizované a existující podle zákonů státu Washington

[2] Linux je registrovanou obchodní známkou pana Linuse Torvaldse

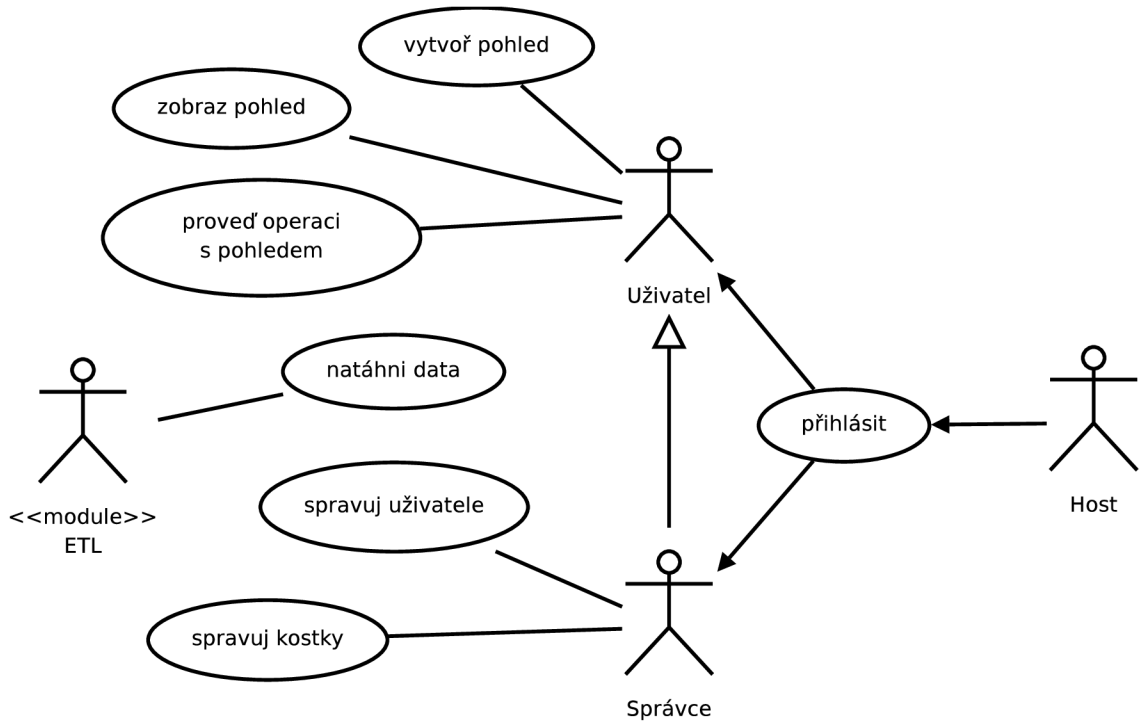
[3] Mac OS je registrovanou obchodní známkou firmy Apple Inc.

$$O(f(n)) = \{g(n) \in \mathcal{F} | \exists c, n_0 \in \mathbb{N}\, \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq g(n) \leq c \cdot f(n)\} \tag{5.1}$$

kde $g(n)$ je skutečná funkce udávající složitost algoritmu a funkce $f(n)$ určuje třídu funkcí realizujících horní asymptotický odhad složitosti.

Množiny O mají obvykle tvary:

- $O(1)$ – pro konstantní složitost,
- $O(n)$ – pro lineární složitost,
- $O(n^2)$ – pro kvadratickou složitost,
- $O(2^n)$ – pro exponenciální složitost.

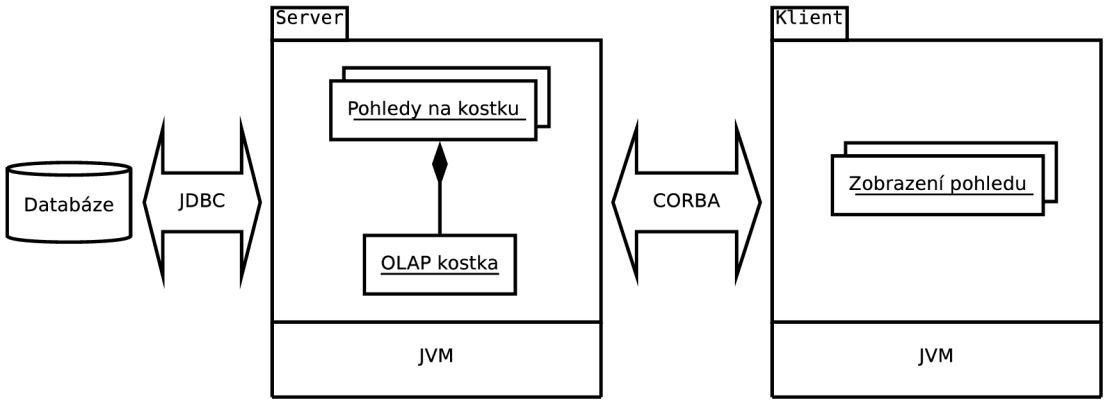


Obrázek 5.1: Schéma případů použití OLAP serveru

Kapitola 6

Návrh aplikace

Vytvářená aplikace bude realizována jako třívrstvá (databáze, server, klient). Bude využívat databázový systém PostgreSQL a bude implementována v programovacím jazyce Java nad JVM. Pro komunikaci mezi databází a serverem bude využito rozhraní JDBC[1] a mezi serverem a klientem bude využito rozhraní CORBA. Aplikace bude nazývána TOPZ. Schéma architektury aplikace TOPZ znázorňuje obrázek 6.1.



Obrázek 6.1: Schéma architektury TOPZ

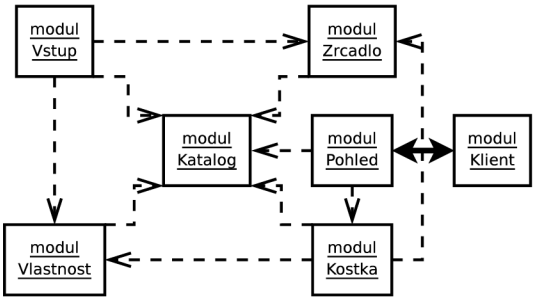
Díky využití JVM je možné věnovat se problémům, které TOPZ řeší, zároveň však umožňuje odstínění od architektury počítače a operačního systému na něm běžícího. Technologie CORBA poskytuje rozhraní pro komunikaci komponent aplikace a zjednodušuje možnost interoperability a rozšiřitelnosti řešení.

Aplikace TOPZ bude sloužit k zjištění náročnosti udržování zrcadla zdrojového systému a rozdílu času plného natažení datového skladiště od natažení přírůstků. Dále bude provedeno přesunutí databáze do RAM-disku s následným testem zrcadla, plného natažení a natažení přírůstků.

[1]JDBC – Java DataBase Connectivity – standardizované rozhraní pro práci s databází v Javě

6.1 Moduly aplikace TOPZ

Aplikace TOPZ bude rozdělena do sedmi modulů, které rozdělí aplikaci na logické celky. Rozložení modulů a jejich vzájemné závislosti jsou zobrazeny v obrázku 6.2.



Obrázek 6.2: Schéma rozložení a závislostí modulů

Základním modulem bude modul Katalog. Tento modul nebude na žádném jiném modulu závislý. Modul Katalog bude využit pro abstrakci databázové vrstvy tak, aby neexistovala závislost na tvaru systémového katalogu.

Hlavním modulem celého systému bude modul Kostka, který bude tvořit jádro celého systému. Veškerý vstup dat do celého systému se bude dít prostřednictvím modulu Vstup, který bude zapouzdřovat celou proceduru vkládání dat do systému. Pro popis dat v datovém skladišti bude sloužit modul Vlastnost a pro zprostředkování dat uživateli modul Pohled a pro prezentaci dat uživateli. Pro čištění a kontrolu dat bude sloužit modul Zrcadlo. Tento modul bude z pohledu diplomové práce zásadní, protože testování popisované v kapitole 8 se bude zabývat právě tímto modulem spolu s hlavním modulem Kostka.

Implementace modulů bude založena na balíčcích JVM, které budou obsahovat dva druhy tříd (s vyjímkou modulu Klient). První druh tříd bude mít oporu přímo v databázi a bude tak realizovat persistenci. Další třídy budou pomocné a budou doplňovat vlastnosti persistentních tříd.

6.1.1 Modul Katalog

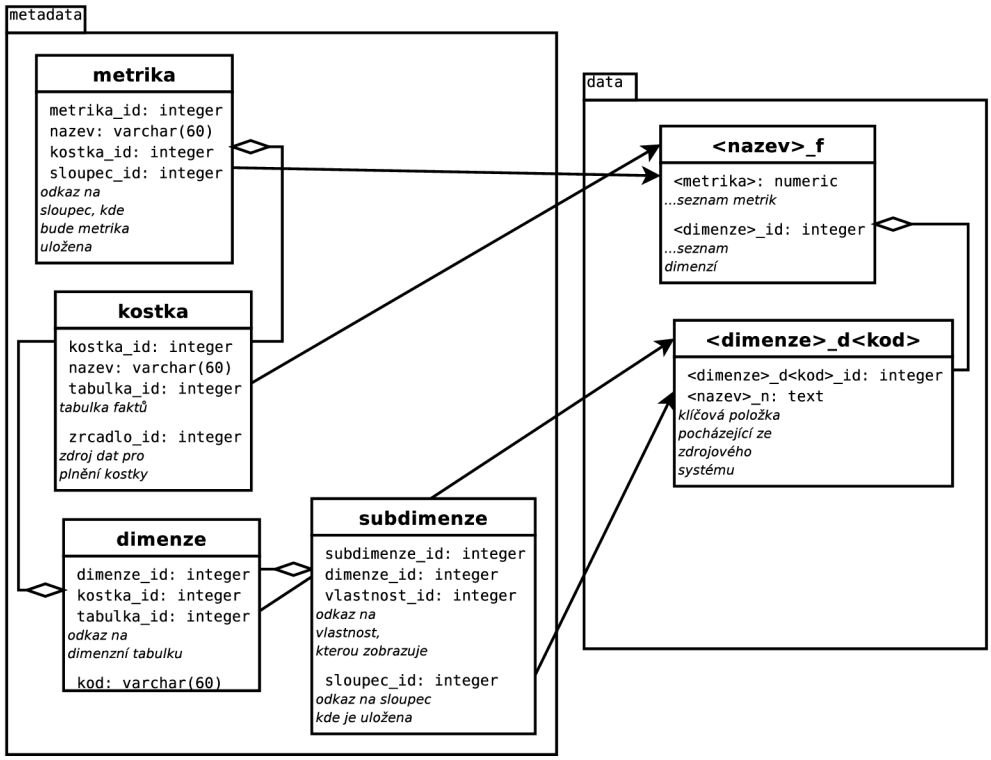
Modul Katalog bude sloužit k manipulaci se základními objekty v databázi (sekvencemi, tabulkami a sloupci). Pomocí zapouzdření bude umožnovat jejich vytváření a správu. Modul bude realizovat duplikovaný systémový katalog. Modul Katalog bude základním kamenem aplikace TOPZ a budou ho využívat téměř všechny ostatní moduly. Bude také umožnovat vytvoření základních tabulek (včetně obsahu) v čisté a prázdné databázi.

6.1.2 Modul Kostka

Hlavní modul realizující datové skladiště bude modul Kostka. Data v datovém skladišti budou, jak bývá v datových skladištích zvykem, organizována do multidimenzionálních kostek.

Modul kostka se bude skládat ze tříd zajištujících perzistenci metadat v databázi a tříd zajištujících import dat do databáze. Tyto dva druhy tříd lze považovat z podmnožinu

návrhového vzoru MVC[2], konkrétně zde není podpora zobrazení dat. Na obrázku 6.3 je znázorněno provázání tabulek metadat a příklad tabulek obsahující data kostky.



Obrázek 6.3: ER diagram modulu Kostka

Každá kostka bude obsahovat jednu až n dimenzí. Každá dimenze se může skládat z jedné až n vlastností. Tyto vlastnosti sdílející jednu dimenzi by měly mít vysokou míru korelace, díky čemuž je možné zmenšit počet dimenzí a zároveň urychlit práci s tabulkou faktů. Tyto vlastnosti sdílející jednu dimenzi bývají často využívány u časové dimenze, kde například vyvstává požadavek na dvojí přístup k času (kalendářní rok vs. zúčtovací období).

Kostka bude realizována jako tabulka faktů (s příponou `_f`) svázaná s jednou až n dimenzními tabulkami.

V tabulce *subdimenze* se budou nacházet odkazy na vlastnosti, která vytvářejí dimenzi. V tabulce *metrika* bude seznam metrik dané kostky.

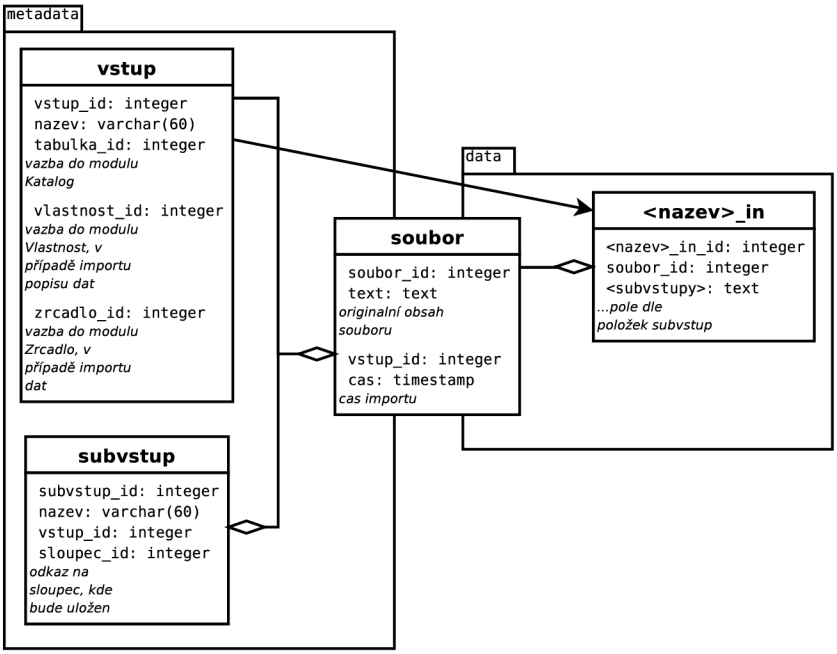
6.1.3 Modul Vstup

Vstup dat z externích zdrojů bude realizovat modul Vstup. Data budou uložena v originálním tvaru a v předzpracovaném tvaru po rozdělení na záznamy. Při přidání nového vstupu bude vytvořena tabulka (s příponou `_in`), do které se budou data rozdělená na záznamy ukládat. Následně bude možno přidat sadu textových sloupců, ze kterých se budou skládat záznamy.

[2]MVC – Model-View-Controller – Model-Pohled-Řadič – návrhový vzor

V prototypu aplikace bude implementován vstup dat ze souboru strukturovaném jako položky oddělované znakem číslo 124 (hexadecimálně `7C` – známém jako svislá čára). Na prvním řádku (v záhlaví) souboru budou místo položek názvy jednotlivých sloupců. Po vložení dat v originálním stavu budou do příslušné tabulky uložena data také ve formě rozdělené na jednotlivé záznamy a sloupce dle specifikace v záhlaví souboru. V souboru bude nutné mít všechny očekávané sloupce, mohou být ovšem v libovolném pořadí.

Na obrázku 6.4 je znázorněno provázání tabulek metadat, tabulky *soubor* a příkladu tabulky dat.



Obrázek 6.4: ER diagram modulu Vstup

V tabulce *subvstup* budou odkazy do modulu Katalog na sloupce příslušného vstupu. Tabulka *soubor* bude na rozhranní metadat a dat. Tato tabulka bude obsahovat jednotlivé načtené soubory dat a také s její pomocí budou data seskupena dle souboru a pořadí importu.

Modul Vstup bude doplněn funkcí automatického generování vstupních souborů dat pro časovou dimenzi.

6.1.4 Modul Vlastnost

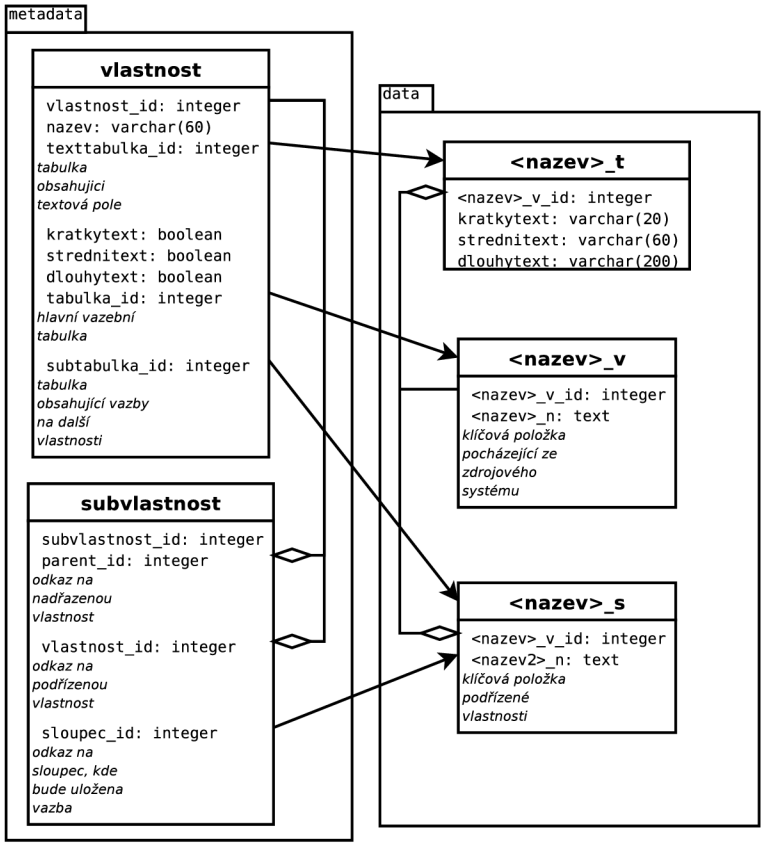
Modul Vlastnost bude umožňovat vytváření stromové struktury pro popis dat uložených v datovém skladišti.

Popis jednoho aspektu dat v datovém skladišti budeme nazývat vlastnost. Každá vlastnost bude mít textový klíč, který bude pro ni unikátní. Tento klíč bude použit pro svázání s daty a bude vhodné, aby byl shodný s unikátním identifikátorem ve zdrojovém systému. Pro zobrazení v aplikaci bude možno použít tři různě dlouhá textová zobrazení (v délkách

20, 60 a 200 znaků), což umožní zmenšit objem dat přenášených ze zdrojového systému a zároveň nesníží komfort práce s daty.

Každá vlastnost bude rozložena do jedné až tří tabulek. První tabulka (s příponou `_v`), která bude existovat vždy, bude obsahovat klíč a unikátní číslo sloužící k rychlejšímu spojování tabulek vlastnosti. Druhá tabulka (s příponou `_t`) bude obsahovat textová zobrazení jednotlivých položek vlastností. Třetí tabulka (s příponou `_s`) bude obsahovat možná propojení vlastností do stromové struktury.

Na obrázku 6.5 je znázorněno provázání tabulek metadat a příklad tabulek obsahující data vlastnosti.



Obrázek 6.5: ER diagram modulu Vlastnost

V tabulce *subvlastnost* budou odkazy na podřízené vlastnosti. Pomocí těchto odkazů je možné vytvořit stromovou strukturu přesně kopírující vztah dat k realitě.

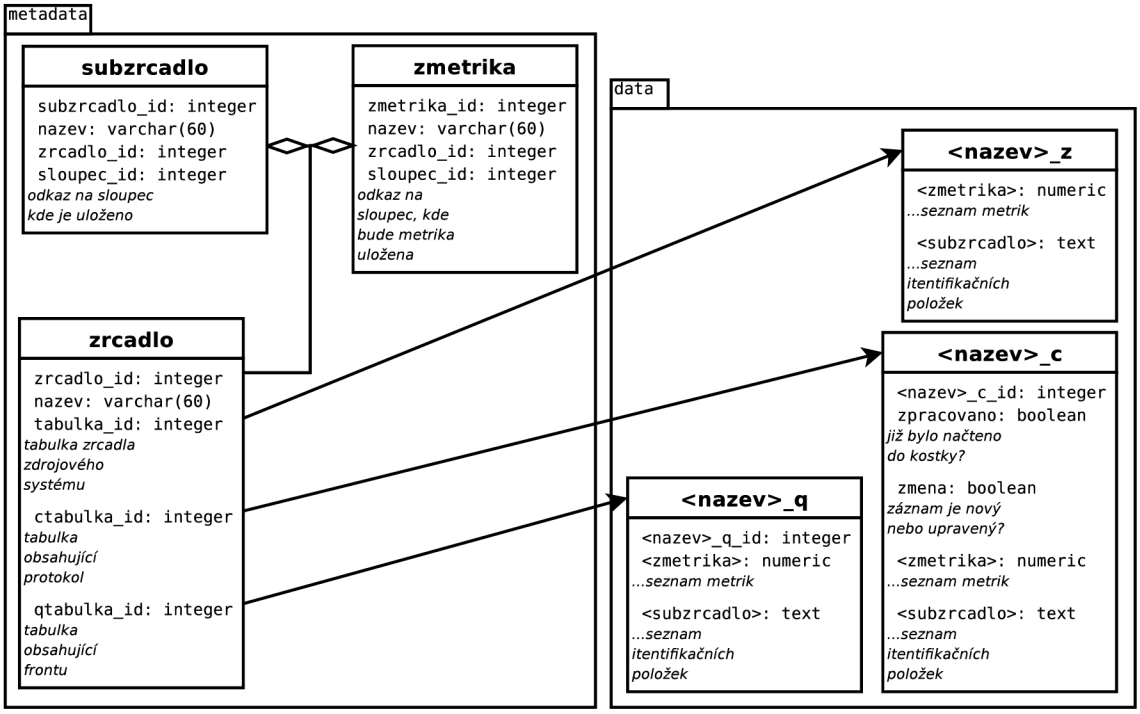
6.1.5 Modul Zrcadlo

Modul Zrcadlo bude modul, který vytvoří prostředí pro testování hypotéz z kapitoly 4.5. Modul bude realizovat zrcadlo zdrojového systému a tvorbu protokolu.

Každé zrcadlo bude složeno ze tří tabulek. První tabulka (s příponou `_z`) bude zobrazovat obsah zdrojového systému (realizuje tzv. zrcadlo) a bude modulem Kostka využíván ke kompletní obnově kostky ze zrcadla. Druhá tabulka (s příponou `_c`) bude protokolem

o změnách zdrojového systému a bude využívána k iterativnímu obnovování kostky do aktuálního stavu. Třetí tabulka (s příponou _q) bude používána jako fronta informací, které se podaří získat ze zdrojového systému. Tato tabulka bude vyprazdňována při importu dat do zrcadla.

Na obrázku 6.6 je znázorněno provázání tabulek metadat a příklad tabulek tvořících zrcadlo.



Obrázek 6.6: ER diagram modulu Zrcadlo

V tabulce *subzrcadlo* se budou nacházet odkazy na sloupce, které popisují data v zrcadle, což odpovídá klíči vlastnosti. V tabulce *zmetrika* budou uloženy sloupce obsahující data v zrcadle, což odpovídá metrice v kostce.

Zrcadlo nebude však třeba vytvářet ručně. Zrcadlo bude vytvářeno na základě nakonfigurované kostky automaticky, a to včetně vytvoření vazby na tuto kostku.

6.1.6 Modul Pohled

Modul Pohled bude modul, který bude zprostředkovávat zobrazení obsahu dat uložených v kostkách. Modul bude transformovat data v kostce dle zadaných parametrů. Transformace bude prováděna jako kopie dat z kostky a její úpravy dle požadavků od uživatele.

Pro každý pohled bude vytvořena jedna tabulka a bude využívat dimenzní tabulky modulu Kostka. V tabulce *Subpohled* se budou nacházet odkazy na vlastnosti a parametry popisující transformace dat v kostce.

6.1.7 Modul Klient

Modul Klient bude jediný modul, který bude obsahovat GUI[3]. Modul Kient bude pomocí GUI umožnovat uživateli sledovat obsah kostky a posílat datovému skladišti pokyny pro transformace dat.

Realizace tohoto modulu bude vycházet z návrhového vzoru MVC, ovšem vrstva Model nebude implementována v modulu Klient, ale budou využívány třídy z ostatních modulů.

Komunikace modulu Klient s ostatními moduly bude probíhat pouze pomocí protokolu CORBA. Díky tomu bude možné, aby modul Klient byl umístěn i na jiném počítači než pouze na serveru.

[3]GUI – Graphical User Interface – Grafické uživatelské prostředí

Kapitola 7

Implementace aplikace TOPZ

Aplikace TOPZ byla implementována jako třívrstvá aplikace s využitím databázového serveru PostgreSQL. Vrstvy aplikace byly označeny jako databázová vrstva, serverová vrstva a klientská vrstva. Serverová vrstva byla implementována nad JVM. Pro komunikaci serverové a klientské vrstvy aplikace TOPZ bylo použito komunikačního protokolu CORBA. Pro předávání zpráv protokolem CORBA je využíván server ORBD, který je součástí distribuce JVM firmy Sun Microsystems. Klientská vrstva byla implementována pomocí rozhraní Swing nad JVM. Swing je doporučené rozhraní JVM pro tvorbu GUI.

7.1 Serverová vrstva

Serverová vrstva sdružuje serverovou část aplikace TOPZ a server ORBD. Serverová část aplikace TOPZ se skládá z těchto šesti modulů:

- modul Katalog,
- modul Kostka,
- modul Vstup,
- modul Vlastnost,
- modul Zrcadlo,
- modul Pohled.

Každý z těchto modulů se skládá z několika tříd, které zajištují persistenci metadat v databázi a obsahuje také metody pro vytvoření svých tabulek metadat v databázi.

Serverová část aplikace TOPZ obsahuje čtyři významné třídy zajišťující manipulaci s daty v datovém skladišti:

- třída ZrcadloProcess,
- třída KostkaPlnyDBProcessor,
- třída KostkaPrirustekProcessor,
- třída PohledProcessor.

Tyto čtyři třídy jsou odvozeny od základní třídy JVM java.lang.Thread, která zapouzdřuje jednotlivá vlákna aplikací běžících nad JVM. Díky tomu je možné u datového skladiště realizovaného aplikací TOPZ využít více než jedno jádro procesoru. To je dosti zásadní výhodou v dnešní době, kdy zvyšování výkonu počítačů jde směrem přidávání procesorů či procesorových jader.

7.1.1 Třída ZrcadloProcess

Modul Zrcadlo obsahuje třídu ZrcadloProcess zajišťující zpracování dat získaných ze zdrojového systému, která slouží k synchronizaci zrcadla zdrojového systému se zdrojovým systémem.

Data v zrcadle zdrojového systému jsou organizovány pomocí n–tice specifikované pomocí tabulky subzrcadlo. Tato n–tice přesně označuje řádek a je tudíž kandidátem na primarní klíč. Tabulka obsahuje n sloupců specifikovaných v tabulce subzrcadlo a m sloupců specifikovaných v tabulce zmetrika. Tabulka zmetrika popisuje vlastní data uložená v zrcadle zdrojového systému, v terminologii datových skladů tzv. *fakta*.

Třída ZrcadloProcess provádí svou činnost postupným průchodem zdrojovou tabulkou (s příponou `_q`) a pro každý řádek provede dotázání na tabulku zrcadla (s příponou `_z`). Pokud v tabulce zrcadla na pozici dané sloupci subzrcadla nenajde položku, založí ji a vytvoří o vytvoření záznam v protokolové tabulce (s příponou `_c`). Pokud je záznam nalezen je pouze změněn a o této změně je vytvořen záznam v protokolové tabulce.

Tento proces průchodu zdrojovou tabulkou umí třída ZrcadloProcess provádět automaticky v odděleném vlákně.

7.1.2 Třída KostkaPlnyDBProcessor

Modul Kostka obsahuje třídu KostkaPlnyDBProcessor zajištující plné natažení dat ze zrcadla zdrojového systému do multidimenzionální kostky.

Plné natažení dat probíhá pouze v databázové vrstvě. Nejprve jsou pomocí dotazu do tabulky zrcadla zdrojového systému vybrány nové záznamy do dimenzních tabulek. Záznamy jsou do dimenzních tabulek vloženy. Poté je obsah kostky v tabulce faktů (s příponou `_f`) vyprázdněn. Nakonec jsou pomocí komplexního dotazu nataženy fakta ze zrcadla zdrojového systému do tabulky faktů. U tohoto komplexního dotazu je využito speciální vlastnost databázového systému PostgreSQL, která umožnuje použít jako zdroj dat pro příkaz INSERT příkazu SELECT.

S výjimkou metadat nejsou mezi databázovou vrstvou a serverovou vrstvou aplikace TOPZ přenášena žádná data. Díky tomu je plné natažení dat ze zrcadla zdrojového systému optimalizována pro maximální výkon při umístění zrcadla zdrojového systému ve stejné databázi, ve které je umístěna multidimenzionální kostku.

7.1.3 Třída KostkaPrirustekProcessor

Součástí modulu Kostka je třída KostkaPrirustekProcessor zajištující natažení přírůstků dat z protokolu zrcadla zdrojového systému do kostky.

Natažení přírůstků probíhá postupným průchodem protokolovou tabulkou zrcadla zdrojového systému a postupným zpracováním zaznámů z této tabulky. Pokud je záznam v protokolové tabulce označen příznakem změna, který značí, že záznam mění již předchozí záznam, je upraven odpovídající záznam v tabulce faktů. Jinak je založen nový záznam v tabulce faktů.

Protože přesun dat mezi zrcadlem zdrojového systému a kostkou probíhá přes serverovou část aplikace TOPZ, není tento způsob natažení tolik optimalizovaný pro maximální výkon.

7.1.4 Třída PohledProcessor

Posledním ze čtyř tříd pracujících s daty v datovém skladišti je třída PohledProcess. Tato třída je jako jediná nepřímo dostupná uživateli. Třída zprostředkovává možnost provádět transformace dat v datovém skladišti.

Pro manipulaci s daty lze použít tři metody:

1. metodu rollup – umožňující provést operaci Roll–Up,
2. metodu drilldown – umožňující provést operaci Drill–Down,
3. metodu filter – umožnující provést operace Slice a Dice.

Tyto metody provedou změnu metadat tak, aby reflektovaly požadovaný výstup, a spustí úpravy dat v pohledu. Úprava dat probíhá nejdříve doplněním dimenzních tabulek o záznamy, které obsahují sesumované vlastnosti. Následuje vyčištění obsahu pohledu a jeho vytvoření komplexním dotazem.

Pro každou vlastnost použitou v multidimenzionální kostce je záznam v tabulce subpohled. Tyto záznamy obsahují nastavení pohledu a jsou také připraveny na přidání metody pivot, která by umožnovala zobrazení dat v kontingenční tabulce.

7.2 Klientská vrstva

Klientská vrstva se skládá pouze z klientské části aplikace TOPZ. Po spuštění klientské části aplikace je zobrazen dialog pro zadání uživatelského jména a hesla. Po odeslání přihlašovacích údajů pomocí protokolu CORBA je vrácen objekt zajišťující komunikaci mezi serverovou a klientskou částí aplikace a je zobrazeno hlavní okno.

Klientská část aplikace se skládá ze tří oken a tří dialogů. Z hlavního okna je možné vyvolat okno se seznamem uživatelů, ve kterém je možné vytvářet a spravovat uživatele. Hlavní okno obsahuje seznam dostupných kostek, po jejichž otevření je zobrazeno okno se seznamem pohledů na danou kostku.

Pohledy je možno vytvářet a mazat. Při otevření pohledu jsou zobrazena data dané datové kostky. Při vytvoření nového pohledu je podleh nastaven pro maximální sumaci dat (nad všemi vlastnostmi je provedena operace Roll–Up), tudíž je zobrazen pouze jeden řádek. Při označení sloupce je možné v menu datového okna volat metody měnící pohled na data v kostce.

7.3 Komunikace serverové a klientské části aplikace

Komunikace mezi serverovou a klientskou částí aplikace TOPZ je zprostředkována serverem ORBD, který zajišťuje doručování volání. Z CORBA protokolu je mimo základní volání metod vzdálených objektů použita jmenná služba (tzv. *Naming Service*) pro získání přístupu k objektu *AuthFactory*.

Objekt AuthFactory umožňuje získání objektu *Auth*, který je hlavním objektem pro komunikaci mezi serverovou a klientskou částí aplikace. Objekt Auth obsahuje metody pro

získání objektu *IUzivatel*, který slouží pro správu uživatelů a jejich práv, a objektu *IKostka* zprostředkovávajícímu práci s datovým skladištěm.

Objekt IKostka nabízí tři metody:

- Metoda *list*, která vrací seznam kostek umístěných v datovém skladišti.
- Metoda *open*, která pro danou kostku vytvoří objekt *IPohled*.
- Metoda *delete* umožňující uživatelům s oprávněním správce smazat kostku.

Objekt IPohled si pamatuje na straně serverové části aplikace kostku, se kterou je svázán a nabízí pro manipulaci s ní čtyři metody:

- Metoda *list*, která vrací seznam pohledů nad danou kostkou v datovém skladišti.
- Metoda *open*, která zpřístupní data v multidimenzionální kostce přes pohled.
- Metoda *create*, která vytvoří k dané kostce nový pohled. Jméno nového pohledu je předáno jako parametr.
- Metoda *delete* umožňující uživateli smazat pohled.

Metoda open jako jeden z parametrů obdrží objekt *PohledListener* a zaregistruje ho k danému pohledu. Registrací objektu PohledListener dojde k zařazení objektu na seznam objektů informovaných o změně pohledu. Metoda open vrátí objekt *KostkaListener*, pomocí kterého lze serverové části oznámit požadavky na změnu parametrů pohledu pomocí metod rollup, drilldown a filter. Objekt KostkaListener je propojen s objektem PohledProcessor, který realizuje všechny operace s pohledem.

7.4 Testovací mód serverové části aplikace TOPZ

Serverová část aplikace TOPZ bude obsahovat speciální mód určený pro potřeby testování výkonu viz kapitola 8. Tento mód provede vytvoření tabulek databáze, vytvoří dvě vlastnosti (produkty a zakaznici) a čtyři časové vlastnosti. Následně vytvoří vstupy k těmto vlastnostem.

Z vlastností vytvoří dvě kostky a k nim zrcadlo zdrojového systému. Na závěr provede vygenerování třiceti šesti tisíc záznamů, které pětkrát natáhne jako celek a pětkrát po rozdělení na deset částí. Do jedné kostky je natažení prováděno plným natažením a do druhé natažením přírůstů. O všech natahovacích operacích podá serverová část informaci o době běhu.

Kapitola 8

Testování

Vytvořená aplikace TOPZ bude podrobena testování výkonnosti a měření spotřeby systémových prostředků jednotlivých počítačů. Testování proběhne na několika počítačích, budou zjištěny všechny dostupné informace jak o hardwaru (tzn. o procesoru, paměti, diskovém a síťovém subsystému), tak i o softwaru (například: verze operačního systému, verze JVM, verze databázového serveru apod.). Před vlastním testováním bude provedeno zjištění propustnosti paměťového a diskového subsystému. Měření diskové propustnosti bude provedeno 5× a bude zpracováno statisticky. Hodnota propustnosti paměťového subsystému bude získána programem Memtest86 a nebude statisticky zpracována, neboť ji lze získat přesně a je deterministicky zjistitelná.

Vlastní testování výkonnosti bude spočívat ve vytvoření velkého datového souboru, který bude 5× natažen do kostky pomocí plného natažení a pomocí natažení přírůstků, přičemž bude měřen čas. Měření bude probíhat jak ve variantě umístění databázového serveru a OLAP serveru na jednom počítači, tak ve variantě na více počítačích. Dále bude velký datový soubor rozdělen do několika částí, které budou nataženy do datového skladiště, přičemž bude testování opět opakováno 5× a v několika kombinacích rozložení komponent na několika počítačích.

Následně budou zhodnoceny výsledky testů a bude diskutován význam využití zrcadla zdrojového systému. Dále bude provedeno srovnání výkonu plného natažení a natažení přírůstků.

8.1 Použité počítače

8.1.1 PentiumM

Prvním počítačem použitým pro testování bude notebook Hewlett-Packard[1] nx6110, pro potřeby práce dále označovaný jako PentiumM. Tento notebook obsahuje procesor Intel® Pentium® M processor 1.73 GHz[2], jeden a půl gigabajtu operační paměti DDR taktované na 333 MHz. Pevný disk osazený v tomto notebooku je FUJITSU[3] MHT2060A s rozhranním Fast Ultra ATA/100 a rychlostí otáčení ploten 4 200 otáček za minutu. Síťová karta zabudovaná v notebooku je Broadcom[4] BCM4401-B0 100Base-TX.

[1] Hewlett-Packard je registrovanou obchodní známkou firmy Hewlett-Packard Development Company L. P.

[2] Intel a Pentium jsou registrované obchodní známky firmy Intel Corporation

[3] FUJITSU je registrovaná obchodní známka firmy Fujitsu Limited

[4] Broadcom je registrovaná obchodní známka firmy Broadcom Corporation

Operační systém na tomto notebooku je Ubuntu[5] 8.04.2 LTS Desktop, běhové prostředí Java HotSpot™Client VM (build 10.0-b23, mixed mode, sharing) a databáze PostgreSQL 8.3.7 zkompilovaná pro platformu i486-pc-linux-gnu.

8.1.2 Core2Duo

Druhým počítačem použitým pro testování bude notebook Lenovo[6] R61, pro potřeby práce dále označovaný jako Core2Duo. Tento notebook obsahuje procesor Intel®Core™2 Duo CPU T7250 2.00 GHz, tři gigabajty operační paměti DDR2 taktované na 667 MHz. Pevný disk osazený v tomto notebooku je HITACHI[7] HTS54161 s rozhranním Serial ATA a rychlostí otáčení ploten 5 400 otáček za minutu. Síťová karta zabudovaná do notebooku je Intel Corporation 82566MM Gigabit Network Connection.

Operační systém na tomto notebooku je Ubuntu 8.04.2 LTS Desktop, běhové prostředí Sun Java HotSpot™64-Bit Server VM (build 10.0-b23, mixed mode) a databáze PostgreSQL 8.3.7 zkompilovaná pro platformu x86_64-pc-linux-gnu.

8.1.3 Celeron

Třetím počítačem použitým pro testování bude starší neznačkový stolní počítač, pro potřeby práce označovaný jako Celeron. Tento počítač obsahuje procesor Intel®Celeron™CPU 2.66 GHz, jeden gigabajt operační paměti DDR taktované na 333 MHz pracující v režimu dual-channel. Pevný disk osazený v tomto počítači je Western Digital[8] Caviar WD2000JB s rozhranním Fast Ultra ATA/100 a rychlostí otáčení ploten 7 200 otáček za minutu. Síťová karta zabudovaná na základní desce je Realtek RTL-8139.

Operační systém na tomto notebooku je Ubuntu 8.04.2 LTS Desktop, běhové prostředí Sun Java HotSpot™Client VM (build 10.0-b23, mixed mode) a databáze PostgreSQL 8.3.7 zkompilovaná pro platformu x86_64-pc-linux-gnu.

8.2 Metodika měření

Všechna měření budou provedena opakovaně (a to vždy pětkrát) tak, aby výsledky měření mohly být statisticky zpracovány. U výsledků měření provedeme výpočet aritmetického průměru $\overline{x}$ a směrodatné odchylky s_x dle vztahů 8.1 a 8.2.

$$\overline{x} = \frac{1}{n} \sum x_i \tag{8.1}$$

$$s_x = \sqrt{\frac{1}{n-1} \sum (x_i - \overline{x})^2} \tag{8.2}$$

kde x_i je i-tý prvek souboru x a n je počet prvků v souboru. Tyto vztahy jsou všeobecně známy; viz [13]. Za předpokladu, že měření má tvar normálního (Gaussova) rozdělení, lze na hladině významnosti $\alpha = 5\%$ dle [2] tvrdit, že skutečná hodnota leží v rozsahu $\overline{x} \pm 2 \cdot s_x$, tento rozsah bude uváděn.

[5] Ubuntu je registrovanou obchodní známkou firmy Caconical Limited
[6] Lenovo je registrovaná obchodní známka firmy Legend (Beijing) Limited
[7] HITACHI je registovanou obchodní známkou firmy Kabuchiki Kaisha Hitachi Seisakusho
[8] Western Digital je registovanou obchodní známkou firmy Western Digital Technologies, Inc.

8.3 Měření propustnosti

Jako první měření bylo provedeno měření propustnosti pevného disku. Propustností pevného disku je souvislý datový tok, který se podaří číst z pevného disku. Pro měření bylo použito nástroje *hdparm* a výsledky jsou uvedeny v tabulce 8.1, údaje jsou v megabajtech za sekundu (MBps). Hdparm patří mezi základní systémové nástroje pro správu pevných disků.

Označení počítače	PentiumM	Core2Duo	Celeron
1. pokus	33,84	45,61	36,38
2. pokus	33,94	45,72	36,35
3. pokus	33,86	45,86	36,45
4. pokus	33,91	45,74	36,31
5. pokus	33,90	45,84	36,33
průměrná hodnota $\overline{x}$	33,89	45,75	36,36
směrodatná odchylka s_x	0,04	0,10	0,05

Tabulka 8.1: Výsledky měření propustnosti pevného disku

Propustnost operační paměti byla zjištěna pomocí programu *Memtest86*, který ji zjišťuje při testování funkce operační paměti. Propustnost operační paměti je souvislý datový tok, kterým se podaří číst data z operační paměti. Výsledky obou testů propustnosti jsou uvedeny v tabulce 8.2.

Označení počítače	PentiumM	Core2Duo	Celeron
Pevný disk	33,89 ± 0,08 MBps	45,7 ± 0,2 MBps	36,36 ± 0,11 MBps
Operační paměť	1110 MBps	1558 MBps	2040 MBps

Tabulka 8.2: Výsledky měření propustnosti použitých počítačů

8.4 Výkonnostní testy

Pro testování výkonnosti operací s daty v datovém skladišti byla zvolena metoda měření času, za který je daná operace provedena. Provádět se pro účely testování bude úprava dat v zrcadle zdrojového systému, plné natažení dat do datového skladiště a natažení přírůstků dat do datového skladiště. Pro zjištění aktuálního času bylo využito standardní volání Java Platform `System.nanoTime()` a výsledný čas byl vypočítán jako rozdíl aktuálního času před operací a po operaci. Nebyla prováděna korekce času volání zjištění aktuálního času.

První test byl realizován vytvořením souboru o velikosti třicet šest tisíc záznamů, jeho importem do zrcadla zdrojového systému, následným importem celého zrcadla plným natažením do kostky a natažením přírůstků z protokolu o změnách zrcadla zdrojového systému do druhé kostky. Výsledky testů po statistickém zpracování pomocí vzorců 8.1 a 8.2 jsou uvedeny v tabulce 8.3.

Výsledky měření času importu a natažení velkého souboru pro přírůstkové a plné natažení ukazují na velmi patrný rozdíl. Tento rozdíl je způsoben dobrou optimalizací plného natažení a téměř žádnou optimalizací přírůstkového natažení, které je navíc složitější o jeden

Označení počítače	PentiumM	Core2Duo	Celeron
Import do zrcadla	19,8 ± 0,7 s	13,1 ± 0,4 s	44,2 ± 0,4 s
Přírůstkové natažení	31,1 ± 0,8 s	19,7 ± 0,6 s	68,4 ± 0,2 s
Plné natažení	9,5 ± 2,2 s	5,1 ± 1,5 s	11,9 ± 0,4 s

Tabulka 8.3: Výsledky měření času importu a natažení velkého souboru

rozhodovací krok. Proto byl proveden test s rozdělením souboru na deset částí po třech tisících šesti stech záznamech, aby bylo zjištěno zda natažení přírůstků má reálný vliv. A případně zjištění podmínek, kdy má natahování přírůstků vliv. Výsledky tohoto testu jsou uvedeny v tabulkách 8.4, 8.5 a 8.6.

Označení počítače	PentiumM	Core2Duo	Celeron
1. část	1,78 ± 0,06 s	1,22 ± 0,20 s	4,29 ± 0,09 s
2. část	1,95 ± 0,03 s	1,25 ± 0,04 s	4,38 ± 0,06 s
3. část	1,92 ± 0,02 s	1,27 ± 0,14 s	4,50 ± 0,02 s
4. část	2,03 ± 0,04 s	1,28 ± 0,09 s	4,47 ± 0,05 s
5. část	2,04 ± 0,09 s	1,33 ± 0,29 s	4,56 ± 0,05 s
6. část	2,06 ± 0,08 s	1,49 ± 0,79 s	4,51 ± 0,06 s
7. část	2,06 ± 0,12 s	1,29 ± 0,11 s	4,58 ± 0,08 s
8. část	2,27 ± 0,31 s	1,51 ± 0,88 s	4,56 ± 0,08 s
9. část	2,41 ± 0,41 s	1,82 ± 1,21 s	4,70 ± 0,09 s
10. část	2,36 ± 0,26 s	1,32 ± 0,10 s	4,64 ± 0,06 s
všechny části dohromady	20,5 ± 0,5 s	14,0 ± 1,4 s	45,2 ± 0,2 s

Tabulka 8.4: Výsledky měření času importu do zrcadla velkého souboru rozděleného na deset částí

Při pohledu na data v tabulce 8.4 je patrné, že čas importu do zrcadla zdrojového systému je závislý na naplnění zrcadla. Pomocí lineární regrese byla získána lineární rovnice popisující data z tabulky 8.4:

$$f_{PentiumM}(x) = 62,23 \cdot x + 1746,25[ms] \tag{8.3}$$

$$f_{Core2Duo}(x) = 38,32 \cdot x + 1169,13[ms] \tag{8.4}$$

$$f_{Celeron}(x) = 35,89 \cdot x + 4322,32[ms] \tag{8.5}$$

kde x je pořadové číslo části. Z koeficientů vztahů 8.3, 8.4 a 8.5 je patrné, že lineární koeficient se velmi blíží směrodatné odchylce a tudíž je zanedbatelný. Díky této skutečnosti lze tvrdit, že čas importu dat do zrcadla zdrojového systém má konstantní časovou složitost. Při srovnání dat v tabulkách 8.3 a 8.4 je také patrné, že rozdělení souboru zvýší časovou náročnost importu, i když nevýznamně.

Z tabulky 8.5 je patrné, že čas natažení přírůstků do kostky je závislý na naplnění kostky. Pomocí lineární regrese byla získána lineární rovnice popisující data z tabulky 8.5:

$$g_{PentiumM}(x) = 38,08 \cdot x + 3614,3[ms] \tag{8.6}$$

Označení počítače	PentiumM	Core2Duo	Celeron
1. část	3,38 ± 0,10 s	2,18 ± 0,30 s	7,68 ± 0,18 s
2. část	3,83 ± 0,12 s	2,27 ± 0,17 s	7,47 ± 0,05 s
3. část	3,80 ± 0,01 s	2,26 ± 0,16 s	7,54 ± 0,36 s
4. část	3,81 ± 0,08 s	2,23 ± 0,21 s	7,42 ± 0,04 s
5. část	3,85 ± 0,09 s	2,21 ± 0,17 s	7,50 ± 0,33 s
6. část	4,00 ± 0,36 s	2,20 ± 0,10 s	7,43 ± 0,07 s
7. část	3,86 ± 0,17 s	2,23 ± 0,18 s	7,50 ± 0,31 s
8. část	3,86 ± 0,12 s	2,36 ± 0,56 s	7,51 ± 0,31 s
9. část	3,90 ± 0,09 s	2,58 ± 0,85 s	7,46 ± 0,09 s
10. část	3,95 ± 0,33 s	2,27 ± 0,28 s	7,52 ± 0,44 s
všechny části dohromady	38,2 ± 0,9 s	22,8 ± 1,2 s	75,0 ± 0,8 s

Tabulka 8.5: Výsledky měření času natažení přírůstku velkého souboru rozděleného na deset částí

$$g_{Core2Duo}(x) = 20,90 \cdot x + 2162,46[ms] \tag{8.7}$$

$$g_{Celeron}(x) = -8,45 \cdot x + 7548,42[ms] \tag{8.8}$$

kde x je pořadové číslo části. Z koeficientů vztahů 8.6, 8.7 a 8.8 je patrné, že lineární koeficient se velmi blíží směrodatné odchylce a tudíž je zanedbatelný. Díky tomu lze tvrdit, že čas natažení přírůstků do kostky má konstantní časovou složitost. Při srovnání dat v tabulkách 8.3 a 8.5 je také patrné, že rozdělení souboru zvýší časovou náročnost importu, i když zcela nepatrně.

Označení počítače	PentiumM	Core2Duo	Celeron
1. část	0,85 ± 0,06 s	0,47 ± 0,01 s	1,30 ± 0,10 s
2. část	1,38 ± 0,30 s	0,72 ± 0,08 s	2,01 ± 0,04 s
3. část	1,96 ± 0,15 s	1,07 ± 0,10 s	3,01 ± 0,05 s
4. část	2,60 ± 0,44 s	1,49 ± 0,17 s	4,00 ± 0,48 s
5. část	3,17 ± 0,56 s	1,83 ± 0,17 s	4,87 ± 0,15 s
6. část	3,75 ± 0,49 s	2,13 ± 0,10 s	5,79 ± 0,19 s
7. část	4,66 ± 0,51 s	2,73 ± 0,32 s	7,14 ± 0,39 s
8. část	5,43 ± 0,54 s	2,95 ± 0,23 s	8,10 ± 0,37 s
9. část	5,99 ± 0,60 s	3,44 ± 0,39 s	9,30 ± 0,53 s
10. část	7,55 ± 1,36 s	4,05 ± 0,77 s	10,42 ± 0,52 s

Tabulka 8.6: Výsledky měření času plného natažení velkého souboru rozděleného na deset částí

Z tabulky 8.6 je patrné, že čas plného natažení dat do kostky je závislý na obsahu zrcadla. Pomocí lineární regrese byla získána lineární rovnice popisující data z tabulky 8.6:

$$h_{PentiumM}(x) = 707,50 \cdot x - 157,69[ms] \tag{8.9}$$

$$h_{Core2Duo}(x) = 391,85 \cdot x - 66,28[ms] \tag{8.10}$$

$$h_{Celeron}(x) = 1020,41 \cdot x - 10,52[ms] \tag{8.11}$$

kde x je pořadové číslo části. Z koeficientů vztahů 8.9, 8.10 a 8.11 je patrné, že kostantní koeficient se velmi blíží směrodatné odchylce a tudíž jej lze zanedbat. Díky tomu lze tvrdit, že čas natažení přírůstků do kostky má lineární časovou složitost. Při srovnání dat v tabulkách 8.3 a 8.6 je také patrné, že rozdělení souboru sníží časovou náročnost importu, i když zcela nepatrně.

Pokud položíme rovny rovnice 8.6 a 8.9, lze odvodit existenci momentu, kdy se začíná vyplácet natahování přírůstků.

$$h_{PentiumM}(x) = g_{PentiumM}(x) \tag{8.12}$$

$$707,50 \cdot x - 157,69 = 38,08 \cdot x + 3614,3 \tag{8.13}$$

$$x = 5,63 \tag{8.14}$$

Analogicky lze získat hodnotu x pro zbylé dva počítače. Výsledky jsou vypsány a statisticky zhodnoceny v tabulce 8.7.

Označení počítače	hodnota x
PentiumM	5,63
Core2Duo	6,01
Celeron	7,35
průměrná hodnota $\overline{x}$	6,33
směrodatná odchylka s_x	0,96
relativní směrodatná odchylka s_r	14 %

Tabulka 8.7: Výsledky výpočtu momentu rovnosti času natažení přírůstků a plného natažení

Z hodnoty s_r je patrné, že moment je stabilní, a tutíž že bude existovat u všech souborů. Lze také dle [9] s 98 % pravděpodobností tvrdit, že pokud bude objem přírůstku alespoň 7,2-krát menší než objem dat v datovém skladišti, je použití natažení přírůstků výhodnější.

8.4.1 Oddělení serverové části TOPZ od databáze

První pokus o zrychlení systému bylo oddělení serverové části od databázového serveru. Pro tento účel byla použita síťová infrastruktura o komunikační rychlosti 100 Mbps. Reálná propustnost sítě byla změřena pětkrát pomocí softwaru *iperf*, který je pro toto měření přímo určen. Měření iperf provádí nad protokolem TCP/IP.

Reálná síťová propustnost mezi počítači PentiumM a Core2Duo byla 11,9 ± 0,1 Mbps.

Z porovnání tabulek 8.3 a 8.8 je patrné, že jediné zrychlení nastalo u plného natažení dat do kostky. Toto je způsobeno tím, že natažení probíhá pouze v databázi. U ostatních testů došlo k výraznému zpomalení, což lze přičíst použití spojení s propustností pouze necelých 12 MBps. Propustnost je ve srovnání s propustností pevných disků v těchto dvou počítačích třetinová (v případě počítače PentiumM), eventuálně čtvrtinová (v případě počítače Core2Duo).

Označení počítače s databázovým serverem	PentiumM	Core2Duo
Označení počítače se serverovou částí TOPZ	Core2Duo	PentiumM
Import do zrcadla	37,7 ± 0,7 s	37,5 ± 0,3 s
Přírůstkové natažení	62,9 ± 0,8 s	53,9 ± 0,6 s
Plné natažení	8,6 ± 1,1 s	4,6 ± 1,1 s

Tabulka 8.8: Výsledky měření času importu a natažení velkého souboru s oddělením databázového serveru od serverové části TOPZ

Z důvodu zrychlení pouze plného natažení dat do datového skladiště nebyla provedena další měření. Oddělení databázového serveru a serverové části aplikace TOPZ nejsou doporučena při propojení 100 Mbps síťovou infrastrukturou jako možnost zrychlení!

8.4.2 Závislost na výkonu pevného disku

Druhý pokus o zrychlení systému byl návrh eliminace pevného disku a přesunutí databáze nad RAM–disk. RAM-disk je emulace pevného disku nad pamětí. Test byl proveden na počítači PentiumM a Core2Duo. Maximální očekávané zrychlení lze odvodit z poměru propustností pevných disků a operačních pamětí z tabulky 8.2. Výsledky byly zpracovány statisticky.

Označení počítače	PentiumM	Core2Duo
Import do zrcadla	18,9 ± 0,2 s	12,8 ± 0,3 s
Přírůstkové natažení	30,1 ± 0,4 s	18,7 ± 0,7 s
Plné natažení	8,5 ± 1,5 s	4,6 ± 0,7 s

Tabulka 8.9: Výsledky měření času importu a natažení velkého souboru s použitím RAM-disku

U výsledků testu s rozdělením souboru na deset částí po třech tisících šesti stech záznamech jsou v tabulce 8.10 uvedeny vztahy získané lineární regresí.

Označení počítače	PentiumM	Core2Duo
Import do zrcadla f	30,72 x + 1721,57	14,64 x + 1144,43
Přírůstkové natažení g	31,98 x + 3574,94	8,68 x + 2135,05
Plné natažení h	574,02 x + 9,81	323,47 x + 56,15

Tabulka 8.10: Výsledky měření času importu a natažení velkého souboru rozděleného na deset částí s použitím RAM–disku

S použitím vzorce 8.15 pro výpočet zrychlení systému byl z průměrných časů vypočíteno zrychlení času β při použití operační paměti místo pevného disku. Dále byl s pomocí vzorce 8.16 vypočtena maximální hodnota zrychlení systému z hodnot propustností operační paměti a pevného disku δ. Tyto koeficienty zrychlení β a δ_{max} jsou uvedeny v tabulce 8.12.

$$\beta = 1 - \frac{t}{t_0} \tag{8.15}$$

$$\delta_{max} = \frac{N}{N_0} - 1 \tag{8.16}$$

Označení počítače	PentiumM	Core2Duo
Maximální zrychlení	3200 %	3300 %

Tabulka 8.11: Koeficienty maximálního zrychlení při použití RAM–disku místo pevného disku

Datový soubor	velký soubor vcelku		velký soubor rozdělený na deset částí	
Označení počítače	PentiumM	Core2Duo	PentiumM	Core2Duo
Import do zrcadla	4,6 %	2,3 %	1,4 %	2,1 %
Přírůstkové natažení	3,2 %	5,1 %	1,1 %	1,3 %
Plné natažení	10,5 %	9,8 %	22,7 %	17,5 %

Tabulka 8.12: Koeficienty zrychlení při použití RAM–disku místo pevného disku

Průměrná hodnota zrychlení dle vzorce 8.1 je 7 % a směrodatná odchylka 7 %. Z hodnot v tabulce 8.12 lze snadno usoudit, že zrychlení je statisticky nevýznamné a několika násobně menší než maximální hodnoty koeficientů z tabulky 8.11.

Příčinou nevýznamného zrychlení datového skladiště při použití RAM–disku je pravděpodobně vysoká sofistikovanost mezipaměti pevného disku, která využívá operační paměť. Díky využití mezipaměti pevného disku se propustnost pevného disku blíží operační paměti, což způsobuje nevýznamné zrychlení při použití RAM–disku.

Kapitola 9

Možná rozšíření a pokračování výzkumu

9.1 Implementace lepšího zobrazení dat v kostce

Tato práce se specializovala na demonstraci kritických míst z pohledu urychlení zobrazení dat od jejich vzniku ve zdrojovém systému po natažení do datového skladiště. V tomto prototypu aplikace TOPZ nebylo účelem vytvoření vhodné prezentace dat v kostce. Aplikace TOPZ umožnuje zobrazení pouze pomocí relační tabulky. V implementovaném systému doporučuji změnu a vylepšení zobrazování obsahu kostky.

9.2 Úprava TOPZ pro distribuovaný provoz systému

Při dalším řešení tohoto problému by bylo vhodné zaměřit se na problematiku zjištění zrychlení systému při přechodu na distribuovaný model. Oddělením zrcadla zdrojového systému od vlastního datového skladiště by mohlo dojít k významnému snížení zatížení datového skladiště a tudíž zrychlení systému jako celku. Je však nutno mít na zřeteli problém spočívající v možném zpomalení způsobeném rychlostí sítě mezi počítači.

9.3 Zrychlení operací Roll–up a Filter

Dále by bylo vhodné zaměřit se na analýzu četnosti použití transformačních operací. Pro optimalizaci výkonu celého systému by mohlo být vhodné zejména zrychlení operací Roll–up a Filter. Pro zrychlení by mohlo být například využito u operace Filter toho, že operace Filter zmenší počet řádků v pohledu, což nemusí nutně znamenat přepočítání celého pohledu, a vede ke zrychlení operace. U operace Roll–up dochází k sesumování řádků, což v některých případech může znamenat pouze přepočítání řádků pohledu, takže není nutné provést sumování dat nad celým datovým skladištěm, což urychlí provedení operace.

Kapitola 10

Závěr

Předložená práce analyzuje důvody pro které jsou využívána datová skladiště při získávání informací z dat. Zároveň je diskutován i problém rychlosti zpracování těchto dat se zřetelem na maximální aktuálnost výsledných výstupů.

V úvodu práce je rozebrán aktuální stav znalostí o funkci datových skladišť a o problematice využití datových skladišť pro získávání informací z dat.

Dále jsou nastíněny požadavky na aplikaci a návrh řešení prototypu aplikace TOPZ, která je určena pro testování významu zrcadla zdrojového systému jako možnosti zlepšení práce s datovým skladištěm. Zlepšením práce s datovým skladištem může být myšleno zrychlení průchodu dat datovým skladištěm či zkrácení doby reakce na vznik nových dat.

V práci je popsána architektura, vývoj a implementace prototypu aplikace TOPZ. Následně jsou uvedeny výsledky měření výkonnosti natahování dat do kostky a provedeno statistické vyhodnocení naměřených dat. Vyslovený předpoklad vhodnosti využívání zrcadla zdrojového systému a natahování přírůstků se ukázal být správný, což bylo jednoznačně prokázáno testy. Z provedených měření bylo zjištěno, že pokud je změněno nebo přidáno ve zdrojovém systému více než 15 % dat, vyplatí se tato data natahovat do datového skladiště pomocí plného natažení dat. Pokud je změněno nebo přidáno méně než 15 % dat vyplatí se dle experimetálně zjištěných dat, prezentovaných a vyhodnocených v části Testování, využít natažení pouze přírůstků.

Závěrem je možno konstatovat, že prezentovaný systém TOPZ splňuje všechny požadavky na něho kladené.

Literatura

[1] Date, C. J.: *Thirty Years of Relational: The First Three Normal Forms, The beginnings of relational dependency theory.* 1999-03-30.

[2] Eckschalager, K., Horsák, I., Kodejš, Z.: *Vyhodnocování analytických výsledků a metod.* SNTL, 1980.

[3] Gosling, J., McGilton, H.: *The Java Language Environment.* May 1996.

[4] Harold, E., Means, W. S.: *XML v kostce.* Computer Press, 2002, iSBN 80-7226-712-4.

[5] Kupčík, J.: *OLAP technologie a mobilní klienti.* FIT VUT v Brně, 2007.

[6] Lacko, L.: *Databáze: datové sklady, OLAP a dolování dat s příklady v Microsoft SQL Serveru s Oracle.* Computer Press, 2003, iSBN 80-7226-969-0.

[7] Meloun, M., Militký, J.,Hill, M.: *Počítačová analýza vícerozměrných dat v příkladech.* Academia, 2005, iSBN 80-200-1335-0.

[8] Messmer, H. P.,Dembowski, K.: *Velká kniha hardware.* Computer Press, 2005, iSBN 80-251-0416-8.

[9] Mittenecker, E.: *Plánování a statistické vyhodnocování experimentů.* SPN, 1968.

[10] Momjian, B.: *PostgreSQL Praktický průvodce.* Computer Press, 2003, iSBN 80-7226-954-2.

[11] Oppel, A.: *Databáze bez předchozích znalostí.* Computer Press, 2006, iSBN 80-251-1199-7.

[12] Pospíšil, P.: Konzultant firmy SAP AG. Ústní sdělení.

[13] Rektorys, K. a spolupracovníci: *Přehled užité matematiky.* SNTL, 1963.

[14] WWW stránky: Historie relačních databází. `http://www.root.cz/clanky/historie-relacnich-databazi/`, [Online; navštíveno 29.3.2009].

[15] WWW stránky: Rtolap - Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Rtolap&oldid=245361016`, [Online; navštíveno 17.12.2008].

[16] WWW stránky: Ukládání počítačových dat. `http://wikipedia.infostar.cz/c/co/computer_storage_1.html`, [Online; navštíveno 4.1.2009].

[17] WWW stránky: Von Neumann architecture - Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Von_Neumann_architecture&oldid=269717243`, [Online; navštíveno 12.2.2009].

[18] Zendulka, J. a kolektiv: *Studijní opora: Získávání znalostí z databází*. FIT VUT v Brně, 2006.

[19] Češka, M., Vojnar, T., Smrčka, A.: *Studijní opora: Teoretická informatika*. FIT VUT v Brně, 2007.

Seznam příloh

Dodatek A

Obsah přiloženého CD

Na přiloženém CD je umístěno následující:

- zdrojová data pro tisk zprávy v adresáři `text`,
- zdrojový kód aplikace TOPZ v adresáři `src`,
- zkompilovanou aplikaci TOPZ včetně obsahu dvou testovacích vlastností v adresáři `topz`,
- knihovny nutné pro běh aplikace v adresáři `topz/lib`.

Dodatek B

Licenční smlouva JDBC ovladače PostgreSQL

Dodatek C

Manuál pro instalaci TOPZ

Základními prerekvizitami pro běh aplikace TOPZ jsou JVM ve verzi 6.0 od společnosti Sun Microsystems Inc. a PostgreSQL ve verzi 8.3.

Po nainstalování prerekvizit bude aplikován následující postup:

1. Na počítači založíme v databázovém serveru databázi s názvem `olap` a povolíme uživateli `postgres` přístup k této databázi bez hesla.
2. Provedeme spuštení `orbd` serveru.
3. Následně spuštěním `java -jar benchmark.jar` vytvoříme obsah databáze včetně testovacích dat.
4. Dále spustíme server příkazem `java -jar server.jar`.
5. A nakonec příkazem `java -jar client.jar` spustíme GUI pro nahlédnutí do datového skladiště.