UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Systém pro tvorbu obchodních nabídek

Bc. Petr Zelený

Diplomová práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Petr Zelený**
Osobní číslo:	**I11424**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Systém pro tvorbu obchodních nabídek**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce je vytvoření funkční internetové aplikace zaměřené na snadnou a automatizovanou tvorbu cenových nabídek pro profesionální použití ve firmách. Aplikace bude využívat systémy veřejné správy pro ověřování údajů a solventnosti zákazníků.
Aplikace bude umožňovat tyto funkcionality:

1. Registrace uživatelů a jejich přístup do aplikace dle uživatelských práv.
2. Správa firem a zákazníků (jejich validace prostřednictvím systému veřejné správy).
3. Správa nabízených služeb a produktů (správa cen a závislostí mezi jednotlivými službami a produkty).
4. Správa šablon nabídky (vlastní logo, kontaktní údaje a text nabídky).
5. Možnost přehledného tiskového výstupu nabídky určené pro zaslání zákazníkovi (včetně automatického odeslání elektronickou poštou).
6. Zobrazení statistik.

V úvodní části je nutné provést analýzu podobných řešení a porovnání s nově navrhovaným systémem, který bude tvořit předmět této práce. Úvodní část musí obsahovat analýzu navrhovaného řešení, která bude obsahovat popis použitých technologií, návrh databáze a aplikačního řešení.
V práci bude navržen postup pro provedení optimalizace vybraných SQL dotazů s následnou analýzou a porovnáním výkonu nalezených alternativ exekučních plánů s původním dotazem.
Aplikace bude naprogramována pomocí vhodné technologie, například jazyka PHP 5 (s využitím Java Scriptu), Java EE a další. Aplikace bude spolupracovat volitelně s databází MySQL 5 a Oracle XE 11. Komunikace s databází bude řešena použitím vhodné oddělovací vrstvy.
Součástí práce bude porovnání rychlosti aplikace na obou databázových platformách.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. LACKO, Luboslav. Oracle. Správa, programování a použití databázového systému. Brno: Computer Press, 2007.
2. GROFF, James R. a WEINBERG, Paul N. SQL kompletní průvodce. Brno: Computer Press, 2005.
3. VRÁNA, Jakub. 1001 tipů a triků pro PHP. Brno: Computer Press a.s., 2010.
4. BRYLA, Bob; LONEY, Kevin. Mistrovství v Oracle Database 11g. Jiří Huf. Vydání první. Brno: Computer Press, a.s., 2009.
5. NOWICKI, Steven D. a LECKY-THOMSON, Ed. PHP 6. Programujeme profesionálně. Brno: Computer Press, 2009.
6. DRUSKA, P. CSS a XHTML - tvorba dokonalých webových stránek krok za krokem, Grada 2006.
7. LONEY, K., THERIAULT, M. Mistrovství v Oracle. Praha, Computer Press, 2002.
8. GATINA, Petr. Programování v jazyku Java (1) ? Úvod [online]. 9. 7. 2004 [cit. 2011-04-19]. Dostupné z www:
9. BOSÁK, Petr. Úvod do programovacího jazyka Java [online]. 24. 4. 2006 [cit. 2011-04-19]. Dostupné z www:
10. WELLING, L., THOMSON, L. MySQL: Průvodce základy databázového systému. 1. vyd. Brno: CP Books, 2005. 255 s. ISBN 80-251-0671-3.
11. DUBOIS, Paul. MySQL profesionálně. Brno: Mobil Media, 2003. 1071 s. ISBN 80-86593-41-X.
12. www.oralce.com

Vedoucí diplomové práce: **Ing. Miloslav Macháček, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2012

Prohlášení autora

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

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 17. 05. 2013 Bc. Petr Zelený

Poděkování

Rád bych poděkoval panu Ing. Miloslavu Macháčkovi, Ph.D. za cenné rady, připomínky a samotné vedení mé práce. Dále bych chtěl poděkovat své rodině a svým blízkým, kteří mi po celou dobu studia byli oporou.

ANOTACE

Tato práce se zabývá vytvořením CRM aplikace umožňující snadnou a rychlou tvorbu obchodních nabídek. Teoretická část obsahuje přehled a analýzu stávajících řešení, která se zabývají obdobnou problematikou. Předmětem praktické části je vytvoření samotné aplikace s možností spolupráce se dvěma různými databázovými platformami. Součástí praktické části je i provedení rozborů exekučních plánů konkrétních SQL dotazů spolu s jejich následnou optimalizací.

KLÍČOVÁ SLOVA

crm, obchodní nabídka, marketing, informační systém, databázová aplikace, webová aplikace

TITLE

Business offer system

ANOTATION

This project deals with creating of CRM system for easy and fast offer creation. The theoretical part of this work contains analysis of existing solutions, which deal with similar issues. The goal of practical part is to develop an application that will work with the two different database platforms. Another goal of practical part is subsequent optimization of SQL queries, together with an analysis of query execution plans.

KEYWORDS

crm, business offer, marketing, information system, database application, web application

OBSAH

ÚVOD **13**
1 CRM SYSTÉMY **14**
2 ANALÝZA STÁVAJÍCÍCH ŘEŠENÍ **15**
2.1 Analýza aplikace Offeris 15
2.1.1 Cena systému 16
2.1.2 Moduly programu 16
2.1.3 Uživatelské prostředí 17
2.1.4 Funkčnost aplikace 17
2.1.5 Shrnutí 18
2.2 Analýza aplikace Nabídky Plus 19
2.2.1 Cena systému 19
2.2.2 Moduly programu 19
2.2.3 Uživatelské prostředí 19
2.2.4 Funkčnost aplikace 20
2.2.5 Shrnutí 21
2.3 Analýza aplikace RodutData Cenová Nabídka 21
2.3.1 Cena systému 21
2.3.2 Moduly programu 22
2.3.3 Uživatelské prostředí 22
2.3.4 Funkčnost aplikace 22
2.3.5 Shrnutí 23
2.4 Shrnutí analýzy 24
3 NÁVRH SYSTÉMU **26**
3.1 Funkční požadavky na aplikaci 26
3.1.1 Zákazníci 27
3.1.2 Produkty 28
3.1.3 Obchod 29
3.1.4 Import/export a synchronizace 29
3.1.5 Pošta 30
3.1.6 Nastavení 31
3.1.7 Statistiky 31
3.2 Návrh datového modelu 32

3.2.1 Rozdíly mezi datovými typy ... 33
3.2.2 Zvolené datové typy pro návrh datového modelu ... 33
3.2.3 Rozdílné řešení automatického generování sekvencí primárního klíče ... 34
3.2.4 Rozdílné restrikce při tvorbě databázových pohledů ... 36
3.2.5 Rozdílné omezení počtu řádků ve výpisu ... 36
3.2.6 Schéma datového modelu ... 37
3.3 Návrh struktury aplikace ... 37
4 VYUŽITÉ TECHNOLOGIE ... 38
4.1 Databázové technologie ... 38
4.1.1 Databázový server Oracle XE 11g ... 38
4.1.2 Databázový server MySQL 5 ... 39
4.2 Programovací prostředky ... 39
4.2.1 Jazyk PHP 5.3 ... 39
4.2.2 JavaScript, jQuery ... 40
4.2.3 Přídavné knihovny ... 40
5 OPTIMALIZACE SQL DOTAZŮ ... 42
5.1 Obecná pravidla optimalizace databázových dotazů ... 42
5.1.1 Pravidla návrhu struktury databáze ... 42
5.1.2 Pravidla tvorby SQL příkazů ... 43
5.2 Přístupové cesty k datům v databázi Oracle ... 44
5.2.1 Úplné prohledávání ... 44
5.2.2 Přímý přístup podle adresy ROWID ... 45
5.2.3 Indexové prohledávání ... 46
5.2.4 Principy spojení tabulek ... 48
5.3 Přístupové cesty v databázi MySQL ... 51
5.3.1 Úplné prohledávání ... 52
5.3.2 Indexové prohledávání ... 52
5.4 Databázové objekty využívané pro optimalizaci ... 52
5.4.1 Indexy v databázi Oracle ... 52
5.4.2 Indexy v databázi MySQL ... 54
6 PRAKTICKÝ POSTUP OPTIMALIZACE SQL DOTAZŮ ... 55
6.1 Naplnění databáze testovacími daty ... 55
6.2 Provedení standardních scénářů ... 56

6.2.1 Scénář pro testování databázových dotazů 57
6.3 Optimalizace dotazů pro databázový server Oracle 59
6.3.1 Optimalizace dotazu pro výpis seznamu nabídek – dotaz č. 1.1 61
6.3.2 Optimalizace dotazu pro výpočet statistiky na hlavní straně – dotaz č. 1.2 63
6.3.3 Optimalizace dotazu pro výpis komunikace – dotaz č. 1.3 65
6.3.4 Shrnutí optimalizace databázových dotazů na platformě Oracle 70
6.4 Rozbory exekučních plánů MySQL 71
6.4.1 Optimalizace dotazu pro výpis seznamu produktů – dotaz č. 2.1 72
6.4.2 Optimalizace dotazu pro zobrazení seznamu nabídek – dotaz č. 2.2 75
6.4.3 Optimalizace dotazu pro zobrazení seznamu komunikace – dotaz č. 2.3 77
6.4.4 Optimalizace dotazu pro zobrazení seznamu firem – dotaz č. 2.4 79
6.4.5 Optimalizace dotazu pro zobrazení seznamu osob – dotaz č. 2.5 82
6.4.6 Shrnutí optimalizace databázových dotazů na platformě MySQL 85
7 POROVNÁNÍ RYCHLOSTI DATABÁZOVÝCH PLATFOREM 87
8 ZÁVĚR 91
9 SEZNAM LITERATURY 93
10 PŘÍLOHY 95

SEZNAM ZKRATEK

ADO	ActiveX Data Objects
API	Application Programming Interface
ARES	Administrativní registr ekonomických subjektů
CRM	Customer Relationship Management
CSV	Comma Separated Values
DDL	Data Definition Language
DRCP	Database Resident Connection Pooling
EAN	European Article Numbering
EE	Enterprise Edition
ERD	Entity Relationship Diagram
GLP	General Public License
HTTP	Hypertext Transfer Protocol
MVC	Model View Controller
OLAP	Online Analytical Processing
SQL	Structured Query Language
UML	Unified Modeling Language
UP	Unified Process
XE	Express Edition
XML	Extensible Markup Language

SEZNAM OBRÁZKŮ

Obrázek 1 – Graf hodnocených vlastností systémů 25
Obrázek 2 – Případ užití modulu Firmy a Osoby 28
Obrázek 3 – Případ užití modulu Import dat 30
Obrázek 4 – Případ užití modulu Import Google 30
Obrázek 5 – Aktéři systému 31
Obrázek 6 – Explicitní využití úplného prohledávání 45
Obrázek 7 – Explicitní využití prohledávání dle adresy ROWID 46
Obrázek 8 – Prohledávání rozsahu indexu 47
Obrázek 9 – Prohledávání rozsahu indexu v sestupném pořadí 47
Obrázek 10 – Spojení vnořenými cykly 49
Obrázek 11 – Spojení typu hash join 50
Obrázek 12 – Spojení typu merge join 50
Obrázek 13 – Anti spojení 51
Obrázek 14 – Struktura B-tree indexu 53
Obrázek 15 – Histogram doby provádění operací pro server Oracle 58
Obrázek 16 – Histogram doby provádění operací pro server MySQL 58
Obrázek 17 – Statistika alternativních exekučních plánů dotazu 1.1 B 62
Obrázek 18 – Statistika alternativních exekučních plánů dotazu 1.2 64
Obrázek 19 – Exekuční plán dotazu 1.3 A 67
Obrázek 20 – Exekuční plán dotazu 1.3 B 68
Obrázek 21 – Statistika testování alternativních exekučních plánů dotazu 1.3 69
Obrázek 22 – Graf rychlostí původních a optimalizovaných dotazů pro server Oracle 70
Obrázek 23 – Exekuční plán dotazu 2.1 A 73
Obrázek 24 – Exekuční plán dotazu 2.1 B 74
Obrázek 25 – Exekuční plán dotazu 2.2 A 76
Obrázek 26 – Exekuční plán dotazu 2.2 B 76
Obrázek 27 – Exekuční plán dotazu 2.3 A 78
Obrázek 28 – Exekuční plán dotazu 2.3 B 79
Obrázek 29 – Exekuční plán dotazu 2.4 A 80
Obrázek 30 – Exekuční plán dotazu 2.4 B 82
Obrázek 31 – Exekuční plán dotazu 2.5 A 83
Obrázek 32 – Exekuční plán dotazu 2.5 B 85
Obrázek 33 – Graf prováděcích časů původních a upravených SQL dotazů 86
Obrázek 34 – Graf celkových časů SQL dotazů před optimalizací s použitím cache 88
Obrázek 35 – Graf průměrů časů SQL dotazů před optimalizací s použitím cache 88
Obrázek 36 – Graf celkových časů SQL dotazů po optimalizaci s použitím cache 89
Obrázek 37 – Graf průměrů časů SQL dotazů po optimalizaci s použitím cache 89
Obrázek 38 – Graf celkové náročnosti optimalizovaných dotazů bez použití cache 90
Obrázek 39 – Graf průměrné náročnosti optimalizovaných dotazů bez použití cache 90

SEZNAM TABULEK

Tabulka 1 – Cenová politika systému Offeris 16
Tabulka 2 – Cenová politika aplikace Nabídky Plus 19
Tabulka 3 – Porovnání hodnocených vlastností systémů 24
Tabulka 4 – Rozdíly v názvech datových typů 33
Tabulka 5 – Zvolené datové typy pro příslušné databázové platformy 34
Tabulka 6 – Počty vygenerovaných řádků jednotlivých tabulek 56
Tabulka 7 – Testovací scénář 57
Tabulka 8 – Časová statistika dotazu č. 1.1 63
Tabulka 9 – Časová statistika dotazu č. 1.2 65
Tabulka 10 – Časová statistika dotazu č. 1.3 69
Tabulka 11 – Průměrné časy provádění SQL dotazů pro server Oracle 70
Tabulka 12 – Časová statistika dotazu č. 2.1 75
Tabulka 13 – Časová statistika dotazu č. 2.2 77
Tabulka 14 – Časová statistika dotazu č. 2.3 79
Tabulka 15 – Časová statistika dotazu č. 2.4 82
Tabulka 16 – Časová statistika dotazu č. 2.5 85
Tabulka 17 – Průměrné časy provádění SQL dotazů pro server MySQL 86

ÚVOD

Informační systémy CRM, tedy systémy pro řízení vztahů se zákazníky, jsou v dnešní době poměrně rozšířené. Tyto systémy jsou využívány především ve středních a velkých společnostech, u kterých se jedná o správu velkého množství zákazníků. Efektivní správa většího množství zákazníků (řádově stovky a tisíce) je bez použití vhodného systému značně komplikovaná či nemožná.

Tato práce se zabývá návrhem aplikace, která umožní nejen správu zákazníků, ale především možnost rychlé tvorby obchodních nabídek a jejich následné rozesílání jednotlivým zákazníkům. Největším přínosem tohoto systému je podstatné zrychlení tvorby profesionálních obchodních nabídek, dále pak snížení chybovosti a v neposlední řadě i sjednocení vizuální podoby nabídek v rámci jedné firmy. Uvedené přínosy by měly vést ke zvýšení úspěšnosti nabídek a ke zvýšení obratu společností. Při návrhu aplikace bylo dbáno na využití služeb třetích stran, díky kterým je možné do značné míry usnadnit práci se systémem, a zvýšit tak uživatelský komfort. Jedná se především o propojení aplikace s českým registrem ARES, díky kterému je možné automatické načítání validních informací o ekonomických subjektech. Dále se jedná o propojení s aplikačním rozhraním společnosti Google, které umožňuje import stávajících kontaktů z existujících emailových účtů.

Zaměření aplikace je cíleno jak na živnostníky a malé firmy, tak i na větší společnosti. Aplikaci je možné provozovat na dvou rozdílných databázových platformách. První databázovou platformou je databázový server Oracle, druhou variantou je pak databázový server MySQL. Díky možnosti propojení s databází MySQL je zajištěn velice levný provoz, jelikož je možné aplikaci provozovat na běžném webhostingovém prostoru. Větší firmy mohou využít podpory databáze Oracle a implementovat tak celý systém do jejich stávající centrální databáze.

Další částí této práce je návrh postupu provádění optimalizace databázových dotazů spolu s praktickou realizací optimalizace. Cílem optimalizace je zajištění dostatečné rychlosti a stability aplikace i v případech, kdy databáze uchovává značné množství dat. Vzhledem k zaměření aplikace je možné předpokládat, že po delší době provozu dojde ke kumulaci databázových dat, tudíž je třeba zajistit dobrou použitelnost a rychlou odezvu i v tomto případě.

1 CRM SYSTÉMY

Pod pojmem CRM systém označujeme souhrnně aplikace, které umožňují řízení vztahů se zákazníky. Využití těchto systémů by mělo vést k zefektivnění obchodních procesů ve firmě a rovněž ke zlepšení vztahů s firemními zákazníky. Kompletní CRM systém bývá velice obsáhlou aplikací, která pokrývá následující oblasti obchodních procesů [1],[2]:

- správa zákazníků,
- správa obchodních nabídek,
- fakturace,
- reklamní kampaně,
- komunikace,
- získávání statistik a prognóz,
- podpora rozhodování.

Vzhledem k tomu, že vytvoření komplexního CRM systému je velice náročné, byla jako předmět této práce zvolena aplikace, která pokrývá pouze určitou část firemních procesů. Konkrétně se jedná o následující procesy:

- správa zákazníků,
- správa obchodních nabídek,
- správa emailových reklamních kampaní,
- komunikace.

V následující kapitole je popsána analýza stávajících řešení, která se zaměřují na obdobnou část firemních procesů jako zamýšlená aplikace. Jedná se tedy o aplikace zaměřené především na tvorbu obchodních nabídek.

2 ANALÝZA STÁVAJÍCÍCH ŘEŠENÍ

Samostatných systémů a programů určených pro tvorbu obchodních nabídek není příliš velké množství. Většinou je tato oblast součástí rozsáhlých CRM systémů. Jak již bylo uvedeno v předchozí kapitole, vytvoření kompletního CRM systému není předmětem této práce. Proto byly mezi porovnávané aplikace zařazeny programy, které se specializují především na tvorbu obchodních nabídek. Tyto programy lze rozdělit do dvou hlavních skupin. První skupinou jsou univerzální programy, které umožňují vytvořit cenovou nabídku z jakéhokoli oboru podnikání. Druhou skupinou jsou pak programy zaměřené na konkrétní oblast podnikání. Do druhé skupiny patří hlavně podpůrné programy kalkulací staveb, návrhů interiéru a podobně. Mezi tyto programy lze zařadit například:

1. **Kalkulace** – program pro provádění kalkulací montáže garážových vrat. Program je dostupný z www.kodl.biz/kalkulace.htm.
2. **euroCalc** – program pro kalkulace a řízení staveb. Podrobné informace a zkušební přístup do aplikace je možné získat na adrese www.callida.cz/eurocalc-3.

Mezi programy, které umožňují obecné vytváření obchodních nabídek lze zařadit následující:

1. **Offeris** – podrobné informace a zkušební přístup do aplikace je možné získat na adrese www.offeris.com/cz.
2. **Nabídky Plus** – podrobné informace a zkušební verzi aplikace je možné získat na adrese www.lanconsult.cz/nabidky-plus.html.
3. **RodutData Cenová nabídka** – podrobné informace a zkušební verzi aplikace je možné získat na adrese www.rodutdata.cz.

V následujících kapitolách je popsána analýza aplikací určených pro tvorbu obecných obchodních nabídek spolu s vyhodnocením jejich silných a slabých stránek.

2.1 Analýza aplikace Offeris

Offeris je profesionální program určený pro tvorbu obchodních nabídek, který vyvíjí slovenská společnost TABI corp., s.r.o. Systém je dostupný jak v on-line verzi, tak i jako desktop aplikace.

2.1.1 Cena systému

Cena systému závisí především na počtu uživatelů, kteří se systémem mohou pracovat. Konkrétní informace o cenách jsou uvedeny v následující tabulce[1].

Tabulka 1 – Cenová politika systému Offeris

Zdroj: [3]

Typ licence	Počet uživatelů	Cena (bez DPH)	Poznámka
Offeris Basic	max 3	650 EUR	Nezahrnuje náklady na instalaci, školení a hosting.
Offeris Business	max 10	950 EUR	Nezahrnuje náklady na instalaci, školení a hosting.
Offeris Premium	nad 10	1200 EUR	Nezahrnuje náklady na instalaci, školení a hosting.
Offeris Enterprise	individuální	individuální	

2.1.2 Moduly programu

Program je řešen klasickou modulární strukturou zajišťující logické oddělení souvisejících procesů. Aplikace obsahuje následující moduly:

- **Cenové nabídky** – tento modul umožňuje samotné vytváření obchodních nabídek a kalkulaci cen. Dále je zde možné generovat výstupy programu, kterými jsou výtisky příslušných nabídek. K jednotlivým nabídkám je možné připojovat úkoly.
- **Firmy, osoby** - tento modul komplexně pokrývá správu zákazníků.
- **Sortiment** – modul pro správu produktů.
- **Katalog** – modul umožňuje rozřazení jednotlivých produktů z předchozího modulu *Sortiment*.
- **Fakturace** – modul umožňuje správu přijatých i vydaných faktur.
- **Obchodní případy** – modul umožňuje seskupení souvisejících dokumentů dle jednotlivých firem.

[1] Uvedené ceny sytému byly platné k datu 4. 5. 2013

2.1.3 Uživatelské prostředí

Aplikace nabízí poměrně komfortní uživatelské prostředí, které ve většině případů odpovídá současným moderním aplikacím podobného typu. Níže jsou uvedeny hlavní body, které byly v souvislosti s uživatelským prostředním zaznamenány.

- Jedná se o online aplikaci vytvořenou prostřednictvím technologie Java EE.
- Aplikace působí moderním dojmem. Zobrazení jednotlivých modulů je poměrně přehledné. Design jednotlivých prvků je jednoduchý, typický pro kancelářské aplikace.
- Odezvy na některé akce se zobrazují v dialogových oknech (využití *jQuery Dialog*).
- Aplikace nabízí možnost nastavení zobrazení jednotlivých sloupců tabulek. Díky tomu je možné uživatelské přizpůsobení vzhledu.
- Některé funkce by mohly být řešeny pomocí technologie AJAX. Zvýšil by se tak uživatelský komfort při práci s aplikací.
- Design by mohl být přehlednější.

2.1.4 Funkčnost aplikace

V této části budou popsány jednotlivé moduly aplikace z hlediska jejich funkčnosti a použitelnosti.

Správa uživatelů

Správa uživatelů je rozdělena do dvou částí. První částí je správa firem, druhou částí pak správa osob. Jednotlivé osoby jsou propojeny s firmami. K firmám je možné přiřazovat příslušné cenové hladiny.

Sortiment

Tento modul se skládá z jednotlivých produktů. Každý produkt je možné rozšířit o produktové typy. K dispozici je možnost cenotvorby ke každému produktu, v rámci které je možné doplnit například náklady spojené s prací apod.

Katalog

Katalog umožňuje organizovat jednotlivé produkty do běžné stromové struktury. Katalog je možné vytisknout a také exportovat do webu či jiné aplikace.

Fakturace

Modul umožňující standardní možnosti fakturace (zálohové faktury, vydané faktury). Fakturu je možné vystavit přímo z nabídky. Modul umí evidovat i faktury přijaté.

Obchodní případy

Tento modul slouží k organizaci veškerých vytvořených dokumentů, které je možné přiřazovat k příslušným firmám. Modul také umožňuje zobrazení přehledových statistik.

Cenové nabídky

Hlavní modul systému, který umožňuje generovat cenové nabídky. Nabídka je přiřazena určité firmě a jsou do ní zahrnuty požadované položky. Při tisku nabídky je možno volit z předpřipravených tiskových šablon.

2.1.5 Shrnutí

Níže jsou uvedeny klíčové vlastnosti analyzovaného systému:

- Jedná se o profesionální, poměrně drahý systém. Přestože grafika je na úrovni současných webových aplikací, je zde prostor pro zlepšení použitelnosti a přehlednosti.
- Systém zahrnuje v podstatě celý obchodní proces.
- Systém neobsahuje podporu importů a exportů kromě tisku nabídky a katalogu.
- Systém neobsahuje možnost synchronizace se službami třetích stran.

Silné stránky

Mezi silné stránky aplikace patří následující body:

- Jedná se o profesionální a komplexní program.
- Funkcionalita programu pokrývá celý obchodní proces.
- Možnost snadného vytvoření elektronického obchodu z nabízeného sortimentu.
- Velké množství tiskových sestav.

Slabé stránky

Mezi slabé stránky systému byly zařazeny následující body:

- Chybí synchronizace se systémy třetích stran.
- Chybí možnost importu/exportu.
- Některé moduly jsou nepřehledné a málo sugestivní.

2.2 Analýza aplikace Nabídky Plus

Jedná se o profesionální komerční aplikaci určenou pro tvorbu obchodních nabídek, kterou vyvíjí česká společnosti *LAN Consult, spol. s r.o*. Aplikace je dostupná pouze pro desktop. Spolupráce s databází je v tomto případě řešena pomocí technologie Microsoft ADO.

2.2.1 Cena systému

Cena systému[2] se liší v závislosti na dostupných funkcích a modulech. Konkrétní informace o cenách jsou uvedeny v následující tabulce.

Tabulka 2 – Cenová politika aplikace Nabídky Plus Zdroj: [4]

Typ licence	Počet uživatelů	Cena (bez DPH)	Poznámka
LITE	1	1 480 Kč	Neobsahuje moduly Adresář a Ceník.
PROFI	1	3 950 Kč	
Každý další uživatel	1	950 Kč	

2.2.2 Moduly programu

Program je opět rozdělen do několika modulů, jejichž přehled je uveden v následujícím seznamu:

- **Nabídky** – umožňuje vytváření a správu obchodních nabídek.
- **Adresář** – eviduje informace o zákaznících.
- **Ceník** – umožňuje správu produktů a jejich cen.

2.2.3 Uživatelské prostředí

Po otestování aplikace byly zjištěny následující skutečnosti popisující chování uživatelského prostředí:

- Grafické prostředí desktop aplikace působí velmi zastarale a neodpovídá vzhledu aktuálních aplikací.
- Aplikace neobsahuje standardní menu a nereaguje na běžné klávesové zkratky.
- Veškeré moduly se otevírají do dialogových oken, tudíž není možné otevřít současně dva moduly najednou.

[2] Uvedené ceny systému byly platné k datu 4. 5. 2013.

2.2.4 Funkčnost aplikace

Adresář

Správa zákazníků je řešena poměrně jednoduchou formou. Přesto umožňuje evidenci všech podstatných údajů. K jednotlivým zákazníkům je možné připojovat kontakty a příslušné cenové hladiny, od kterých se odvíjí výsledná cena produktů v nabídce.

Nabídky

Tvorba nabídky je poměrně propracovaná a umožňuje rychlé vytvoření nabídky i pro zákazníka, který zatím není evidován. Při tvorbě nabídky je možné využívat předem připravené šablony. Přínosné jsou hromadné možnosti úprav všech produktů obsažených v nabídce. Nabídku nelze vytisknout přímo z jejího editačního formuláře, ale je nutný návrat o jednu obrazovku zpět. Nabízené možnosti a předvolby tisku jsou plně dostačující. V tomto modulu chybí možnost odeslat nabídku automaticky emailem.

Ceník

Tento modul umožňuje správu produktů a jejich řazení do skupin (obdoba kategorií). Chybí zde možnost evidovat dodavatele zboží, dále pak není umožněno vytvářet produktové typy. U produktů nelze zaznamenávat historii cen. Modul neumožňuje vytvářet souvislosti mezi jednotlivými produkty.

Další funkce

Program má velmi dobře propracovanou část importů dat, prostřednictvím kterých je možné importovat kontakty firem a ceník. Import podporuje klasické formáty xls, csv, dále pak dBase a databázi Access. Při nastavení importu je možné mapovat jednotlivé sloupce importovaného souboru do databáze, tudíž lze provádět import i ze souborů, které nemají stejné pořadí sloupců. Jednotlivé definice pravidel pro import je možné uložit, což ušetří práci při opětovném importu dat ze stejného typu souboru.

2.2.5 Shrnutí

Níže jsou uvedeny klíčové vlastnosti analyzovaného systému:

- Grafické prostředí desktop aplikace působí velmi zastarale a neodpovídá vzhledu aktuálních aplikací.
- Aplikace neobsahuje standardní menu a nereaguje na běžné klávesové zkratky.
- Veškeré moduly se otevírají do dialogových oken, tudíž není možné otevřít současně dva moduly najednou.
- Propracovaná možnost importu dat.
- Systém je vhodný spíše pro živnostníky nebo malé firmy.

Silné stránky

Mezi silné stránky programu patří následující body:

- Malý, rychlý program.
- Dobrá podpora importu dat.
- Možnost sdílení databáze v síti.

Slabé stránky

Mezi slabé stránky programu patří následující body:

- Vizuální podoba aplikace.
- Špatná uživatelská použitelnost aplikace.
- Program nepokryje celý obchodní proces.
- Chybí podpora přímého odesílání nabídek emailem.

2.3 Analýza aplikace RodutData Cenová Nabídka

Program Cenová nabídka je jakousi nadstavbou pro kancelářský balík Microsoft Office. Konkrétně využívá tabulkového procesoru Microsoft Excel.

2.3.1 Cena systému

Program je možné zakoupit s klasickou licencí pro jeden počítač. Nutností je předchozí instalace kancelářského balíku Microsoft Office, který není v ceně licence. Cena jedné licence

činí 490 Kč[3] [4]. Jedná se tedy o poměrně levný program, nicméně jeho nízká cena také odpovídá jeho funkčnosti.

2.3.2 Moduly programu

Tento program se neskládá z modulů v pravém slova smyslu. Jednotlivé formuláře (listy programu Microsoft Excel) jsou seskupeny do určitých logických celků, nicméně o modulární struktuře programu se zde hovořit nedá.

Struktura programu je dělena do následujících celků:

- **Katalog položek** – umožňuje správu produktů pro tvorbu nabídky.
- **Cenová nabídka** – umožňuje samotné sestavení nabídky.
- **Odběratelé** – jednoduchý seznam zákazníků.
- **Databáze nabídek** – jednoduchý seznam vytvořených nabídek.
- **Databáze smluv** – jednoduchý seznam vytvořených smluv.
- **Databáze faktur** – jednoduchý seznam faktur.

2.3.3 Uživatelské prostředí

Z hlediska testování uživatelského prostředí byly učiněny následující závěry:

- Jak již bylo zmíněno, jedná se o nadstavbu kancelářského programu Microsoft Excel, tudíž uživatelské prostředí tvoří v podstatě jednotlivé listy tohoto kancelářského programu, které jsou doplněny o různé ovládací prvky.
- Jednotlivá pole formulářů lze vyplňovat buď přímo – jako buňky listu, nebo je možné použít dialogové okno s příslušnými formulářovými poli.
- Vzhledem k tomu, že se nejedná o samostatný program, je ovládání poměrně komplikované.

2.3.4 Funkčnost aplikace

Databáze odběratelů

Jedná se pouze o jednoduchý seznam, který umožňuje evidenci názvu odběratele, adresy, IČ, DIČ a kontaktních údajů. Není zde umožněno přidávat více kontaktních osob k jedné firmě, doplňovat více kontaktů k osobě apod.

[3] Uvedená cena systému byla platná k datu 4. 5. 2013.

Katalog položek

Opět se jedná o jednoduchý seznam. Počet položek je omezen na 42 000. Položky je možné rozdělit do kategorií, jejichž počet je omezen na 15 hlavních kategorií a dalších 7 podkategorií. Chybí zde možnosti vytváření produktových typů, zaznamenávání cenové historie, připojení obrázků k produktu, vytvoření formátovaného popisu a další pokročilejší funkce.

Cenová nabídka

Tato sekce je propracovanější oproti předchozí. Umožňuje vkládání vyplněných položek produktů z databáze. Nabídku je možné napojit na konkrétního dodavatele. Tato sekce rovněž umožňuje přidat k nabídce textovou část. Dále je možné připojit k nabídce fotografie. Nabídku je možné tisknout, exportovat pro odeslání emailem a exportovat do databáze.

Fakturace

Sekce umožňuje vytvořit z nabídky příslušnou fakturu (zálohovou i vydanou).

Smlouvy

Tato sekce umožňuje vytvářet smlouvy. Výhodou je, že program obsahuje několik předdefinovaných šablon pro konkrétní druhy smluv.

2.3.5 Shrnutí

Níže je uveden seznam klíčových vlastností analyzovaného systému:

- Jedná se o softwarové řešení pro méně náročné uživatele. Systém je využitelný především pro samostatné živnostníky, jelikož zde není umožněna kooperace více uživatelů.
- Systém obsahuje určitá omezení, například počty produktů, počty kategorií apod.
- Chybí zde možnosti importů a exportů, tudíž není prakticky možné využít nějakou stávající databázi kontaktů.
- Systém neobsahuje možnost synchronizace se službami třetích stran.

Silné stránky

Mezi silné stránky aplikace patří:

- Nízká cena.
- Ovládání, které je shodné se sadou Microsoft Office, může být dobře použitelné pro stávající uživatele tohoto kancelářského balíku.

Slabé stránky

Mezi slabé stránky aplikace patří:

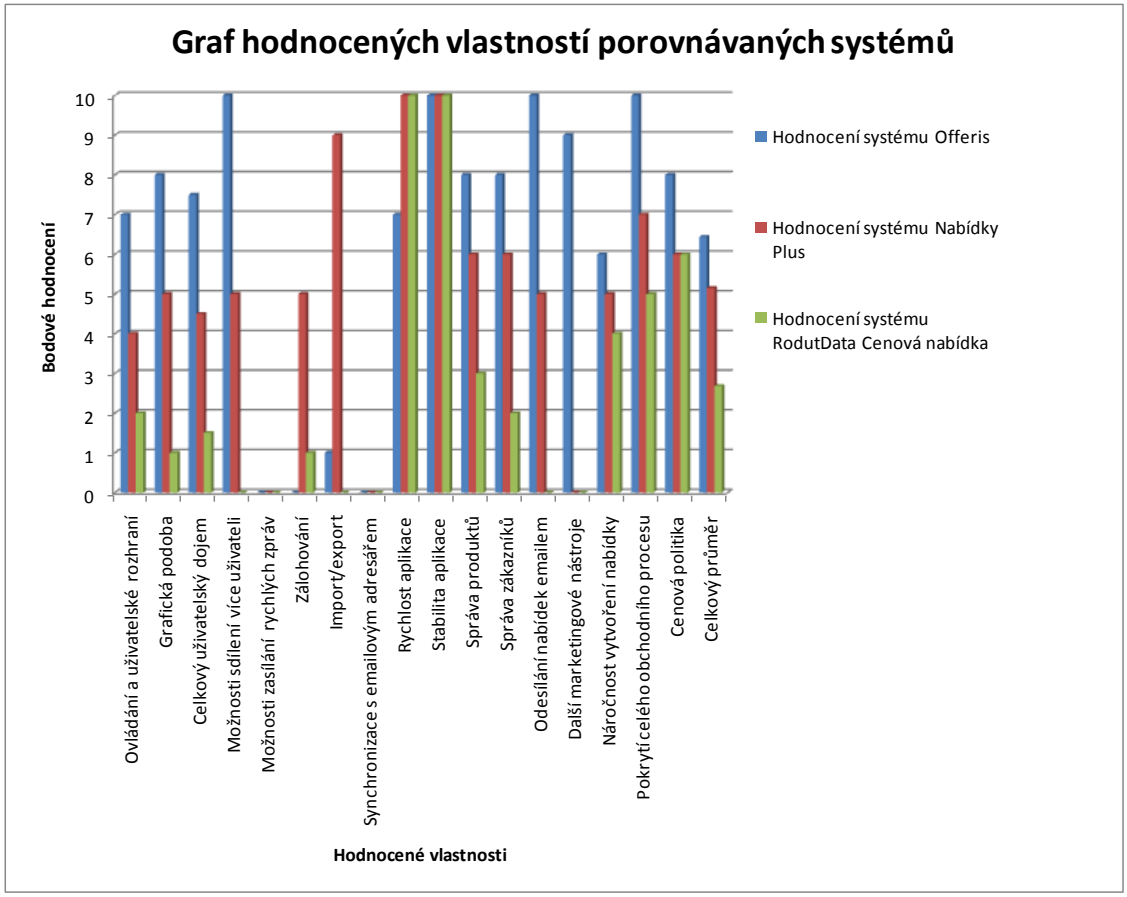
- Chybí synchronizace se systémy třetích stran.
- Chybí možnost importu/exportu.
- Nepřehledné ovládání.
- Omezení počtu položek a kategorií.
- Chybí možnost vytvoření šablony nabídky.
- Chybí možnost uživatelského nastavení tiskových výstupů.
- Nutnost zakoupit kancelářský balík Microsoft Office.

2.4 Shrnutí analýzy

V této kapitole je provedeno vzájemné porovnání programů, které byly analyzovány v předchozí části. Následující tabulka obsahuje hodnocení příslušných programů z hlediska jednotlivých vlastností. Výsledky jsou pak přehledně zobrazeny v grafu.

Tabulka 3 – Porovnání hodnocených vlastností systémů

Porovnání vlastností systémů (maximum - 10 bodů u každé hodnocené vlastnosti)				
	Hodnocená vlastnost	Hodnocení systémů		
		Offeris	Nabídky Plus	Rodut Data Cenová nabídka
Grafika a UI	Ovládání a uživatelské rozhraní	7	4	2
	Grafická podoba	8	5	1
	Celkový uživatelský dojem	7,5	4,5	1,5
Kooperace mezi uživateli	Možnosti sdílení více uživateli	10	5	0
	Možnosti zasílání rychlých zpráv	0	0	0
Sdílení a využívání dat	Zálohování	0	5	1
	Import/export	1	9	0
	Synchronizace s emailovým adresářem	0	0	0
Rychlost a stabilita	Rychlost aplikace	7	10	10
	Stabilita aplikace	10	10	10
Obchodní procesy	Správa produktů	8	6	3
	Správa zákazníků	8	6	2
	Odesílání nabídek emailem	10	5	0
	Další marketingové nástroje	9	0	0
	Náročnost vytvoření nabídky	6	5	4
	Pokrytí celého obchodního procesu	10	7	5
Cenová politika	Cenová politika	8	6	6
	Celkový průměr	6,44	5,15	2,68



Obrázek 1 – Graf hodnocených vlastností systémů

3 NÁVRH SYSTÉMU

V této kapitole je popsán samotný návrh celého systému. Pro přehlednost byl návrh rozdělen na následující části:

- **Specifikace funkčnosti aplikace** – popisuje funkční požadavky vzešlé z provedené analýzy stávajících řešení.
- **Návrh datového modelu** – popisuje postup při tvorbě datového modelu pro obě databázové platformy.
- **Návrh struktury aplikace** – popisuje obecné koncepty a přístupy využité při implementaci systému.

Jednotlivé body jsou podrobně rozepsány v následující části. Aplikace byla vyvíjena pod pracovním názvem *DEALER*. Tento název je tudíž často využíván ve jménech projektů a ve zdrojových kódech aplikace.

3.1 Funkční požadavky na aplikaci

V této kapitole jsou popsány klíčové vlastnosti systému, které byly zvoleny v návaznosti na analýze stávajících řešení. Jelikož se jedná o poměrně rozsáhlý systém, je třeba jednotlivé funkcionality systému rozdělit do následujících sekcí:

- zákazníci,
- produkty,
- obchod,
- import/export a synchronizace,
- pošta,
- nastavení.

V další části jsou jednotlivé moduly stručně popsány z hlediska funkcí, které budou poskytovat. Kompletní dokumentace návrhu systému, včetně všech případů užití a jejich scénářů, je k dispozici na přiloženém CD. Pro tvorbu dokumentace byla využita metodika UP a modelovací nástroj Enterprise Architect verze 9.

3.1.1 Zákazníci

Je žádoucí, aby prostřednictvím této sekce byly evidovány firmy včetně kontaktních osob, se kterými je vedeno obchodní jednání. Sekce tedy bude rozdělena na dva moduly: firmy a kontaktní osoby.

Kontaktní osoba reprezentuje osobu, se kterou je vedeno obchodní jednání. V současné době se stále častěji dostávají do popředí jiné komunikační kanály, než pouze emailová komunikace. Proto je třeba klást zvýšenou pozornost evidenci těchto alternativních komunikačních kanálů. V systému budou uchovávány především následující informace o kontaktních osobách:

- jméno, příjmení, titul,
- oslovení,
- doručovací adresa,
- kontaktní údaje (možnost přidat jakýkoli kontakt – email, telefon, ICQ, Skype atd.).

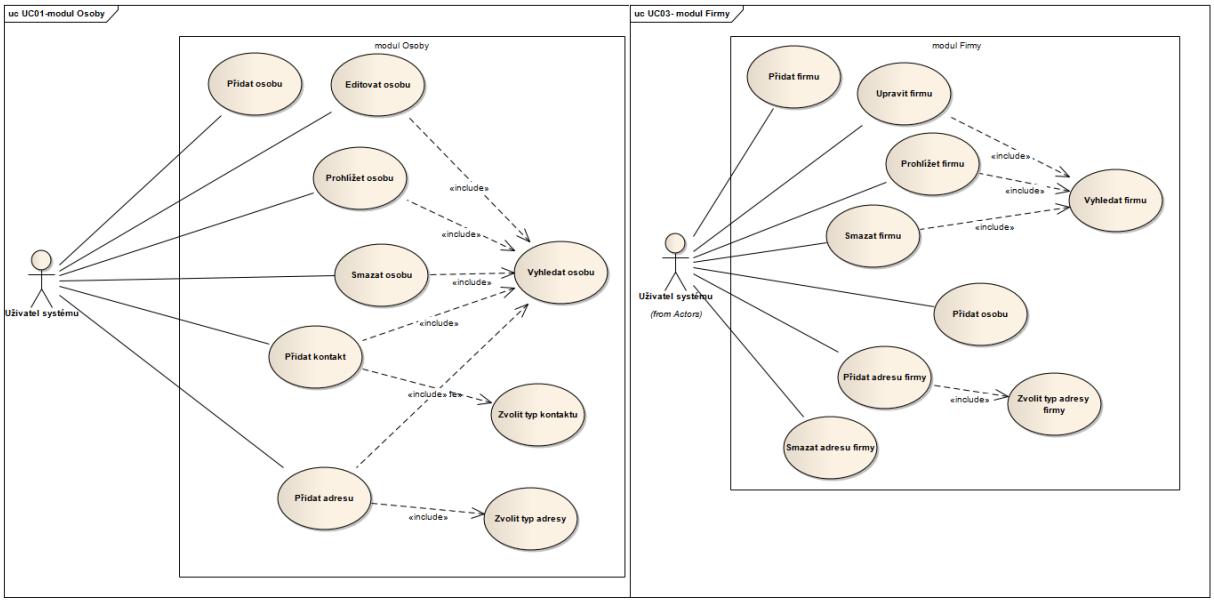
Firma reprezentuje fyzickou či právnickou osobu, s jejímiž zaměstnanci jsou vedena obchodní jednání. Systém musí evidovat příslušné údaje, které jsou potřebné pro formální vytvoření nabídky, případně smlouvy na nabízené služby. V systému budou uchovávány především následující informace o firmě:

- název, typ společnosti,
- IČ, DIČ,
- adresa sídla, doručovací adresa,
- kontaktní osoby (hlavní kontaktní osoba),
- komunikace s kontaktními osobami,
- cenová hladina.

Pro urychlení práce s modulem *Firmy*, bude tento modul obsahovat funkci načtení potřebných informací prostřednictvím rejstříku firem ARES. Informace o ekonomickém subjektu budou automaticky načítány v závislosti na uvedeném IČ.

Vzhledem k tomu, že počet záznamů firem v tomto modulu může být velmi vysoký (řádově tisíce), je třeba implementovat možnost podrobné filtrace údajů dle různorodých kritérií.

Na následujícím obrázku jsou uvedeny dva základní případy užití sekce zákazníci (konkrétně se jedná o modul Firmy a Osoby).



Obrázek 2 – Případ užití modulu Firmy a Osoby

3.1.2 Produkty

Modul primárně slouží k evidenci jednotlivých nabízených produktů. Produkt bude možné zařazovat do různých kategorií a katalogů. Pro každý produkt bude možné nadefinovat několik dodavatelů, kteří se mohou lišit cenou. Modul bude umožňovat vytváření produktových typů. O produktu budou uchovávány především následující informace:

- název, EAN,
- krátký popis, dlouhý popis,
- dodavatel, výrobce,
- nákupní cena, prodejní cena, DPH, cenová historie,
- galerie obrázků, připojené soubory,
- parametry produktu,
- produktové typy, související produkty.

Diagram případů užití tohoto modulu je z důvodu jeho značné velikosti uveden v příloze A.

3.1.3 Obchod

Sekce *Obchod* bude pokrývat celý proces tvorby nabídky. Nabídka je hlavním produktem obchodníka, může být vytvořena ze šablony, nebo vyplněním jednotlivých polí příslušného formuláře. Nabídka bude napojena na firmu a její kontaktní osoby, kterým může být zaslána.

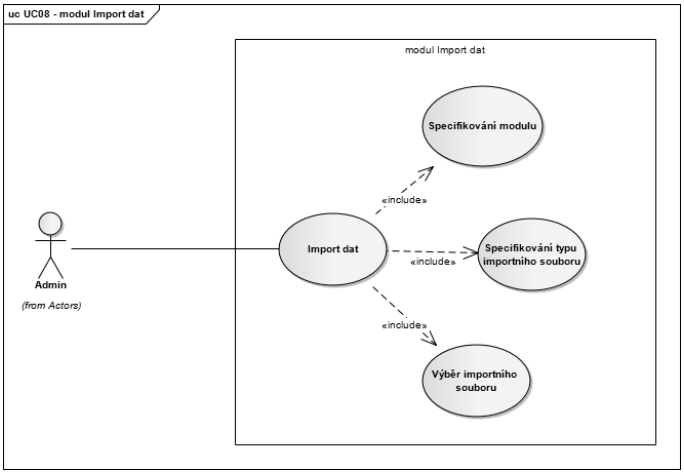
Součástí nabídky bude seznam zařazených produktů, případně seznam připojených slev na jednotlivé produkty nebo na celkovou nabídku. Nabídku lze snadno vytisknout v závislosti na nastavení požadovaného formátu výstupu do pdf. Vytvořený výtisk je pak možné zaslat zákazníkům vedeným v systému.

Součástí sekce *Obchod* bude i možnost editace šablon nabídek, uchovávání komunikace k jednotlivým nabídkám a evidence souvisejících úkolů. Diagram případů užití hlavního modulu této sekce je kvůli své značné velikosti uveden v příloze B.

3.1.4 Import/export a synchronizace

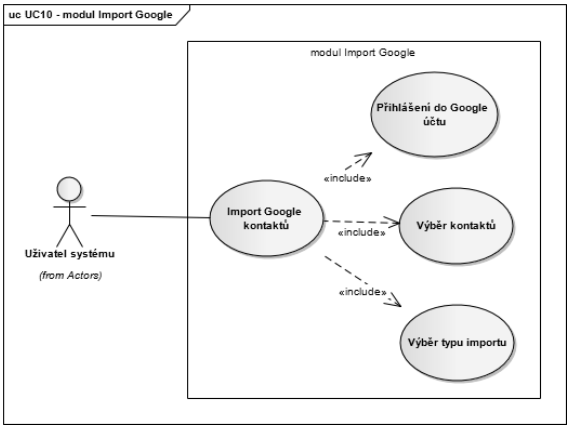
Tato sekce slouží k univerzálnímu provádění výměny dat mezi jednotlivými aplikacemi nebo databázemi pomocí souborů typu xml a csv. Vzhledem k tomu, že podpora výměny dat u stávajících řešení nebyla příliš dobrá, byl na tuto část kladen zvýšený důraz. Systém umožňuje následující druhy importu a exportu:

- **Export tabulkových výpisů do formátu CSV** kompatibilního s kancelářským programem Microsoft Office. Tento druh exportu provádí export tabulky tak, jak ji vidí uživatel systému na své obrazovce. Cizí klíče tabulky jsou tedy nahrazeny příslušnou hodnotou z propojené tabulky. Tento druh exportu je vhodný například k sestavování reportů pro management firmy, kdy je tabulkový editor Microsoft Excel velice oblíbenou formou prezentace dat.
- **Import/export kompletních tabulek jednotlivých modulů do formátu CSV**, který provádí import/export dat uložených v databázové tabulce příslušného modulu. Struktura dat je poté totožná s daty příslušné tabulky. Tento import/export je vhodný buď jako forma zálohování příslušného modulu, nebo jako prostředek pro rychlé naplnění databáze z již existujících dat.
- **Import/export kompletních tabulek jednotlivých modulů do XML.** Tento druh importu je obdobný jako předchozí, nicméně data jsou exportována do univerzálního formátu XML. Exportovaná data je možné dále strojově zpracovávat a univerzálně využít. Na následujícím obrázku 3 je znázorněn případ užití modulu pro import dat.



Obrázek 3 – Případ užití modulu Import dat

- **Import kontaktů z účtu společnosti Google**. Tento druh importu byl vytvořen pro snadné a rychlé zahájení práce se systémem tak, aby uživatelé mohli využít svých kontaktů, které mají u společnosti Google uloženy a nemuseli je ručně do systému přenášet. Případ užití importu z Google účtu je uveden na následujícím obrázku.



Obrázek 4 – Případ užití modulu Import Google

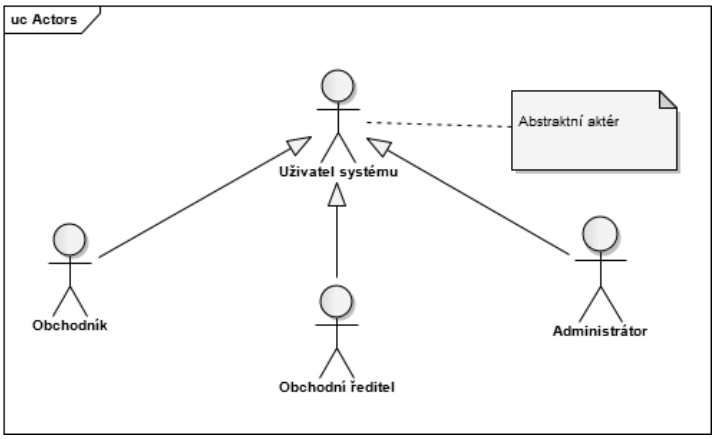
3.1.5 Pošta

Tato sekce slouží k evidenci a správě emailové komunikace, která je prostřednictvím aplikace vedena s jednotlivými zákazníky. Sekce se skládá z následujících modulů:

- **Modul Emaily** – slouží pro uchovávání veškerých emailových zpráv, které byly v rámci systému vytvořeny.
- **Modul Newsletter** – slouží pro vytváření hromadných emailových sdělení, které je možné rozesílat po menších dávkách tak, aby nedocházelo k problémům se spamovými filtry.

3.1.6 Nastavení

Sekce nastavení slouží především ke správě uživatelů, kteří do aplikace přistupují. V neposlední řadě je pak tato sekce využívána pro správu hodnot jednotlivých číselníků, které aplikace využívá. Na následujícím obrázku jsou uvedeny uživatelské role dostupné v systému.



Obrázek 5 – Aktéři systému

Z předchozího obrázku vyplývá, že jsou v systému dostupné následující role:

- **Obchodník** – základní druh uživatelského oprávnění. Umožňuje přístup k těm záznamům systému, které danému uživateli náleží nebo k záznamům, které jsou mezi uživateli sdíleny.
- **Obchodní ředitel** – uživatel s tímto oprávněním může přistupovat ke všem záznamům v systému s výjimkou nastavení parametrů, které vyžaduje odborné znalosti.
- **Administrátor** – nejvyšší oprávnění v systému, které umožňuje pracovat s veškerými záznamy v systému včetně všech nastavení.

Kompetence jednotlivých oprávnění je možné upravovat na míru příslušné organizaci, která bude systém provozovat. Nicméně je nutné provést tuto úpravu přímo ve zdrojovém kódu aplikace, nikoli v administraci.

3.1.7 Statistiky

Vzhledem k tomu, že statistiky jsou vždy úzce spjaty s příslušným modulem nebo sekcí, bylo jako vhodné řešení zvoleno uvádění statistických údajů přímo v detailu jednotlivých záznamů (například v další záložce formuláře konkrétní firmy apod.).

Obecné a přehledové statistiky jsou navíc zobrazeny na úvodní stránce aplikace. Jedná se především o statistický pohled na vytvořené nabídky za dané období, dále pak přehledy úkolů a přehledy rozpracovaných nabídek.

3.2 Návrh datového modelu

Samotnému návrhu datového modelu byla věnována velká pozornost především kvůli možnosti pozdější optimalizace databázových dotazů. Při návrhu byla dodržována následující kritéria:

- Dodržování třetí normální formy.
- Volba odpovídajících datových typů.
- Správné využití primárních a cizích klíčů.

Při návrhu nebyly vytvářeny žádné speciální indexy, kromě indexů automaticky vytvořených u sloupců primárních a cizích klíčů. Indexy byly doplněny až při optimalizaci databázových dotazů.

Pro návrh datového modelu byl využit nástroj *Oracle SQL Developer Data Modeler*. Předností tohoto programu je snadný návrh relačních modelů pro databáze Oracle a jejich následný export do DDL. Nevýhodou je chybějící podpora exportu datového modelu pro databázi MySQL.

Výsledný DDL skript byl v případě MySQL upraven ručně tak, aby odpovídal syntaxi tohoto databázového serveru. V následujících částech je uvedeno několik hlavních rozdílů mezi formalismem obou databází. Tato kapitola si neklade za cíl porovnávat komplexní strukturu a způsob práce obou databází, pouze popisuje hlavní odlišností z hlediska syntaxe SQL příkazů a způsobu návrhu datového modelu.

3.2.1 Rozdíly mezi datovými typy

Rozdíly mezi základními (běžně používanými) datovými typy nejsou nikterak markantní. Většinou se jedná pouze o různé varianty názvů těchto typů. Větší rozdíly se projeví při využívání specifických datových typů určených například pro uchovávání velkých binárních dat. Níže je uvedena tabulka porovnání (mapování) základních datových typů databází MySQL a Oracle.

Tabulka 4 – Rozdíly v názvech datových typů

Zdroj: [6]

Datové typy MySQL	Datové typy Oracle	Datové typy MySQL	Datové typy Oracle
BIGINT	NUMBER(19, 0)	MEDIUMBLOB	BLOB, RAW
BIT	RAW	MEDIUMINT	NUMBER(7, 0)
BLOB	BLOB, RAW	MEDIUMTEXT	CLOB, RAW
CHAR	CHAR	NUMERIC	NUMBER
DATE	DATE	REAL	FLOAT (24)
DATETIME	DATE	SET	VARCHAR2
DECIMAL	FLOAT (24)	SMALLINT	NUMBER(5, 0)
DOUBLE	FLOAT (24)	TEXT	VARCHAR2, CLOB
DOUBLE PRECISION	FLOAT (24)	TIME	DATE
ENUM	VARCHAR2	TIMESTAMP	DATE
FLOAT	FLOAT	TINYBLOB	RAW
INT	NUMBER(10, 0)	TINYINT	NUMBER(3, 0)
INTEGER	NUMBER(10, 0)	TINYTEXT	VARCHAR2
LONGBLOB	BLOB, RAW	VARCHAR	VARCHAR2, CLOB
LONGTEXT	CLOB, RAW	YEAR	NUMBER

3.2.2 Zvolené datové typy pro návrh datového modelu

Z požadavků na systém vyplývá, že v datovém modelu bude třeba uchovávat data popisující následující obecné údaje:

- identifikátory řádků,
- krátké textové řetězce (do 256 znaků),
- dlouhé textové řetězce (HTML popisy produktů, texty nabídek apod.),
- datum,
- datum a čas,

- reálná čísla (ceny, sazby DPH apod.),
- celá čísla.

V následující tabulce jsou uvedeny příslušné datové typy, které byly zvoleny pro danou databázi.

Tabulka 5 – Zvolené datové typy pro příslušné databázové platformy

Druh uchovávaných údajů	Datový typ MySQL	Datový typ Oracle	Poznámka
Identifikátory řádků	INT, BIGINT	NUMBER (38,0)	Typicky se jedná o sloupce primárních klíčů (ID).
Krátké textové řetězce	VARCHAR(n)	VARCHAR2(n)	
Dlouhé textové řetězce	MEDIUMTEXT	CLOB	Oracle podporuje i datový typ LONG pro dlouhé řetězce, nicméně se doporučuje využívat datové typy ze skupiny LOB.
Datum	DATE	DATE	
Datum a čas	DATETIME	DATE	
Reálná čísla	DECIMAL(m,n)	NUMBER(m,n)	
Celá čísla	INT	NUMBER(n,0)	

3.2.3 Rozdílné řešení automatického generování sekvencí primárního klíče

U většiny tabulek, kde je primární klíč tvořen jednoznačným číselným identifikátorem, je nutné zajistit automatické generování unikátní sekvence celých čísel. Každá z databázových platforem řeší tento problém jiným způsobem.

Databázový server MySQL využívá pro automatické generování posloupností sloupce s atributem AUTO_INCREMENT. Syntaxe použití tohoto sloupce je uvedena níže na příkladu vytvoření tabulky osoba.

```
CREATE TABLE OSOBA
(
ID    INT AUTO_INCREMENT NOT NULL,
JMENO VARCHAR(30)
)ENGINE=InnoDB AUTO_INCREMENT=5;
```

U tabulek typu InnoDB, InnoDBD, MyISAM a HEAP je možné počáteční hodnotu čítače nastavit explicitně pomocí volby AUTO_INCREMENT=n. Čítač tabulky typu ISAM začíná standardně na hodnotě 1.

K vygenerování nové hodnoty sloupce dojde v následujících případech:

- Do sloupce je vložena hodnota NULL.
- Do sloupce je vložena hodnota 0.
- Hodnota pro sloupec není explicitně uvedena.

Automaticky vloženou hodnotu identifikátoru je následně možné zjistit pomocí příkazu LAST_INSERT_ID(), který vrátí poslední automaticky vygenerovanou hodnotu sekvence.

Databázový server Oracle využívá odlišného přístupu. Pro zajištění automatického vkládání unikátních hodnot je třeba vytvořit další dva databázové objekty. Sekvenci, která bude generovat unikátní posloupnost čísel a tigger, který zajistí vložení vygenerovaného čísla do tabulky. Níže je uvedena syntaxe jednotlivých databázových objektů.

```
CREATE TABLE OSOBA
    (
     ID INTEGER NOT NULL,
     JMENO VARCHAR2 (30)
    )
;
CREATE SEQUENCE OSOBA_ID_SEQ
    NOCACHE
    ORDER;

CREATE OR REPLACE TRIGGER OSOBA_ID_TRG
BEFORE INSERT ON OSOBA
FOR EACH ROW
WHEN (NEW.ID IS NULL)
BEGIN
    SELECT OSOBA_ID_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;
END;
```

Výše uvedený kód je sice složitější oproti použití sloupce AUTO_INCREMENT, nicméně dovoluje podstatně větší kontrolu nad procesem přidělování jedinečných identifikátorů. V těle triggeru je možné specifikovat libovolný kód, na základě kterého bude přidělování pracovat.

Automaticky vloženou hodnotu je možné zjistit přímým dotazem na aktuální stav čítače (current value) příslušné sekvence. Příslušný příkaz je uveden níže.

```
SELECT NAZEV_SEKVENCE.CURRVAL AS ID FROM DUAL;
```

3.2.4 Rozdílné restrikce při tvorbě databázových pohledů

Databáze MySQL nedovoluje při tvorbě databázového pohledu využít vnořené dotazy. Situaci je třeba vyřešit jiným vhodným spojením tabulek. Příklad řešení takového pohledu je uveden níže.

Pohled pro databázi Oracle

```
CREATE VIEW ACTUAL_PRICES AS
SELECT P1.PRODUCTS_ID, P1.SUPPLIERS_ID, P1.PRODUCTS_TYPES_ID, P1.MVAL AS
VALID_FROM, P2.SELL_PRICE, P2.PURCH_PRICE, VAT.VALUE
FROM (SELECT PRODUCTS_ID, SUPPLIERS_ID, PRODUCTS_TYPES_ID, MAX(VALID_FROM)
AS MVAL FROM PRICES P1
GROUP BY PRODUCTS_ID, SUPPLIERS_ID, PRODUCTS_TYPES_ID) P1
JOIN PRICES P2 ON P1.MVAL=P2.VALID_FROM
JOIN VAT ON P2.VAT_ID=VAT.ID;
```

Provedená úprava pro databázi MySQL

```
CREATE VIEW ACTUAL_PRICES AS
SELECT P1.PRODUCTS_ID, P1.SUPPLIERS_ID, P1.PRODUCTS_TYPES_ID,
MAX(P1.VALID_FROM) AS VALID_FROM , P2.SELL_PRICE , P2.PURCH_PRICE , VALUE
FROM PRICES P1 JOIN PRICES P2
ON P1.PRODUCTS_ID=P2.PRODUCTS_ID AND
P1.VALID_FROM = P2.VALID_FROM
JOIN VAT ON P2.VAT_ID=VAT.ID
GROUP BY P1.PRODUCTS_ID, P1.SUPPLIERS_ID, P1.PRODUCTS_TYPES_ID
```

3.2.5 Rozdílné omezení počtu řádků ve výpisu

Pro přehledný tabulkový výstup dat v administraci je třeba zajistit stránkování dat vrácených databázovým serverem. V tomto případě jsou opět přístupy obou databázových serverů značně rozdílné.

Databázový server MySQL využívá v SQL dotazu klauzule limit se dvěma číselnými parametry – *limit x, y*.

- **Parametr x** – definuje číslo řádku, od kterého budou výsledky databázového dotazu vráceny.
- **Parametr y** – definuje počet řádků, který bude navrácen.

Databázový server Oracle umožňuje využít pro omezení počtu vrácených řádků pseudo sloupec ROWNUM. Hodnota ROWNUM je přiřazena každému řádku při jeho zpracování databázovým serverem a indikuje tak pořadí, ve kterém byl tento řádek navrácen [21].

Pro omezení počtu vrácených řádků se využívá konstrukce, kdy je původní SQL dotaz použit jako vnořený. Následně jsou podmínkou WHERE specifikována čísla řádků, která chceme získat. Příklad použití sloupce ROWNUM je uveden níže.

```
SELECT * FROM
(SELECT * FROM NAZEV_TABULKY)
WHERE ROWNUM BETWEEN (10, 20);
```

3.2.6 Schéma datového modelu

Kompletní schéma datového modelu je kvůli své velikosti (obsahuje 52 tabulek) uvedeno na přiloženém CD. V přílohách na CD je možné nalézt model pro databázi Oracle i model pro databázi MySQL. Fragment diagramu datového modelu je také umístěn v textové příloze E.

3.3 Návrh struktury aplikace

Struktura aplikace byla navrhována dle konceptu (architektury) MVC. Jedná se tedy o architekturu rozdělenou na vrstvy, které mezi sebou komunikují, ale jsou na sobě minimálně závislé. Architektura MVC se skládá z následujících vrstev [7]:

- **Model** – reprezentuje data objektu, se kterými aplikace pracuje.
- **View** – reprezentuje vizuální podobu dat. Zajišťuje tedy převod informací obsažených v modelu do nějaké srozumitelné formy, která je zobrazena uživateli.
- **Controller** – má za úkol reagovat na jednotlivé události. Controller obsahuje logiku aplikace a zajišťuje operace s daty, které jsou zapouzdřeny modelem.

Velkou výhodou této architektury je minimální závislost jednotlivých úrovní. Díky tomu, že je zobrazovací část oddělena od logiky aplikace, je možné snadno transformovat data modelu do požadované formy, aniž by bylo nutné výrazněji zasahovat do aplikace. Toho je u webových aplikací možné úspěšně využít například při tvorbě šablon, které sice pracují nad totožnou logikou, nicméně jejich výstupy jsou odlišné.

4 VYUŽITÉ TECHNOLOGIE

Volba technologií pro vytvoření aplikace se řídila především funkčními požadavky na systém, které vznikly při analýze stávajících řešení. To znamená, že bylo nutné zajistit pokrytí všech případů užití včetně propojení s API třetích stran a napojení systému na webové služby. Technologie byly rovněž voleny tak, aby provozní náklady na aplikaci byly přijatelné i pro menší podniky a živnostníky. Obecně je možné říci, že z pohledu programátora bylo nutné zvolit databázovou platformu, která bude spravovat poměrně rozsáhlý datový model, dále pak vhodný programovací jazyk, ve kterém bude vytvořena logika aplikace běžící na straně serveru a v neposlední řadě vhodný skriptovací jazyk, který umožní interaktivní funkce na straně klienta. V další části budou jednotlivé technologie rozebrány podrobněji.

4.1 Databázové technologie

Specifikace databázových technologií byla poměrně jasně dána zadáním této práce. Dle zadání má aplikace podporovat propojení s databázovým serverem Oracle XE 11g a databázovým serverem MySQL 5.

Možnost volby mezi výše uvedenými databázemi umožní široké využití aplikace na běžných serverech, kde je databáze MySQL nabízena jako součást standardních webhostingových služeb. Naproti tomu je zde také možnost implementovat aplikaci do databáze Oracle, což ocení především větší společnosti, které tuto databázi využívají pro své stávající systémy.

4.1.1 Databázový server Oracle XE 11g

Databázový server Oracle 11 je v současné době nejnovější databázový server této společnosti. Verze XE (Express Edition) je pak varianta, která je distribuována zdarma. Vzhledem k tomu, že standardní databáze Oracle jsou placené, mají parametry XE edice značná omezení. Omezení edice XE jsou následující [8]:

- Využití pouze jednoho procesoru.
- Využití maximálně 1 GB paměti RAM.
- Maximální velikost uživatelských dat 11 GB.

Oproti předchozí verzi databáze Oracle XE 10 je zde patrná značná změna v limitu na celkovou velikost uživatelských dat. Předchozí edice byla omezena velikostí 4 GB. Aktuální limit ve výši 11 GB dovoluje tuto databázi využívat i pro poměrně rozsáhlé aplikace. Tato verze přináší velké množství novinek a společnost Oracle ji prezentuje jako nový

standard bezplatné verze. Za zmínku stojí také nové vylepšené připojení (DRCP) do databáze určené pro webové aplikace, které by mělo poskytnout mnohem větší rychlost.

4.1.2 Databázový server MySQL 5

Pro vývoj aplikace byl konkrétně použit databázový server MySQL 5.5.8. MySQL je databázový systém, který je v současnosti vlastněn rovněž společností Oracle. Tento systém je však distribuován pod licencí GLP, tudíž je možné používat ho v mnoha případech zcela zdarma [9].

Současné verze tohoto systému nabízí poměrně sofistikované funkce databázového serveru při zachování rychlé odezvy. Databázové systémy jsou velice oblíbené mezi webovými vývojáři především díky příznivé licenční politice a výše uvedeným kladným vlastnostem. Oproti Databázi Oracle 11 zde však není podpora pokročilých analytických funkcí, tudíž v případě požadavku na transparentní spolupráci aplikace s oběma databázemi je nutné tuto situaci řešit rozdílnými databázovými dotazy v závislosti na konkrétní databázi, která je aktuálně využívána.

4.2 Programovací prostředky

4.2.1 Jazyk PHP 5.3

Jazyk PHP je původně skriptovacím jazykem, který byl vyvinut v roce 1995. V dnešní době je k dispozici ve své verzi 5, která již dobře podporuje velké množství vlastností objektově orientovaných jazyků, jako jsou například třídy, rozhraní, dědičnost apod. Díky podpoře objektově orientovaného programování je možné vytvářet rozsáhlé aplikace s dobře čitelným a strukturovaným kódem. Rovněž rozšiřitelnost těchto aplikací dosahuje při správném návrhu velice dobré úrovně. Syntaxe jazyka vychází ze syntaxe C++ a je poměrně snadná na porozumění. Jazyk PHP je dynamicky typovaný, to znamená, že typ proměnné není vázán k jejímu názvu, ale k její hodnotě [10].

Jazyk PHP je nejčastěji využíván pro tvorbu webových aplikací, které běží na straně serveru. Velice často bývá kombinován s databází MySQL či PostgreSQL. Tato kombinace s sebou nese minimální finanční náročnost zvláště ve chvíli, kdy je jako operační systém využita bezplatná distribuce Linuxu spolu s webovým serverem Apache.

4.2.2 JavaScript, jQuery

JavaScript je objektově orientovaný programovací jazyk, který je využíván především na straně klienta. Prostřednictvím JavaScriptu je možné značně zvýšit použitelnost webových aplikací. V současné době je mezi vývojáři webových aplikací velice populární používání JavaScriptové knihovny jQuery, která vyniká dobrou podporou napříč webovými prohlížeči. Dalším velmi oblíbeným frameworkem je jQuery UI, které mimo jiné obsahuje takzvané widgety, což jsou připravené interaktivní grafické prvky (například kalendář, dialogová okna, záložkové panely a podobně)[11].

4.2.3 Přídavné knihovny

Vzhledem k tomu, že aplikace obsahuje poměrně náročné funkcionality, bylo vhodné využít stávajících knihoven, které implementaci těchto funkcionalit značně usnadňují. V aplikaci byly použity následující knihovny.

Dibi – jedná se o knihovnu vytvořenou v jazyce PHP, která slouží jako oddělovací databázová vrstva usnadňující práci s různými druhy databází. Hlavními výhodami použití této knihovny je například sjednocení práce s datem, sjednocení omezení počtu výsledků databázového dotazu, automatická podpora konvencí apod. Tato knihovna je dobře prověřená širokou základnou uživatelů a je například využívána i ve známem webovém frameworku Nette. Samozřejmostí je ochrana před technikou napadení prostřednictvím SQL injection[4]. Ochrana spočívá v kontrole proměnných vkládaných do dotazu vzhledem k typu příslušného datového sloupce. Dále dochází k odfiltrování potenciálně nebezpečných znaků z kódu SQL dotazu. Níže je uveden příklad jednoduchého databázového dotazu spolu se zpracováním jeho výsledků [12].

```
//provedení SQL dotazu
$result = dibi::query(SELECT ID, NAME FROM TABLE);

//získání výsledků ve formě asociativního pole
$res = $result->fetchPairs('ID', 'NAME');
```

Předchozí skript nejprve provede databázový dotaz uvedený jako argument statické metody query. Následně je pomocí metody *fetchPairs* vráceno asociativní pole, kde jako klíče budou použita příslušná ID záznamů, jako hodnoty pak jejich názvy. V další ukázce kódu je uveden

[4] Jedná se o způsob napadení databázové aplikace pomocí vkládání cizího SQL kódu do příkazů, které aplikace využívá. Děje se tak prostřednictvím neošetřených vstupních polí (především formulářů, případně hodnot parametrů předávaných v URL).

příklad vkládání konkrétních hodnot proměnných do příslušného dotazu. Rovněž je z ukázky patrné, jakým způsobem je možné pracovat s množinou výsledků.

```
//provedení SQL dotazu
$result = dibi::fetchAll('
SELECT USERS.ID as ID, PASSWORD, COMPETENCE.NAME as COMP, COMP_VALUE from
USERS
JOIN COMPETENCE on USERS.COMPETENCE_ID = COMPETENCE.ID WHERE EMAIL=%s',
$login);

//získání počtu vrácených řádků
 if (count($result) == 0) {
    return 'Uživatelské jméno, nebo heslo není správné.';
   }
//získání hodnot jednotlivých sloupců
  foreach ($result as $r) {
          $id = $r['ID'];
          $pass = $r['PASSWORD'];
          $competence = $r['COMP'];
 }
```

Nette – jedná se o velmi populární webový framework, používaný především v České republice. Framework nabízí kompletní sadu tříd pro snadné vytváření webových aplikací. V této práci však byla využita pouze jediná třída tohoto frameworku, konkrétně se jednalo o třídu Message, která slouží pro snadné odesílání emailových zpráv[13].

TCPDF – jedná se o robustní knihovnu určenou pro tvorbu PDF dokumentů. Knihovna umožňuje nepřeberné možnosti nastavení formátu výstupu, tudíž jejím výstupem může být v podstatě libovolný PDF dokument [14].

Google Api PHP Client – tato knihovna usnadňuje připojení k aplikačnímu programovému rozhraní společnosti Google. Díky ní je možné poměrně snadno přistupovat k jednotlivým službám, které společnost Google veřejně nabízí. Jedná se například o Google Kalendář, Google Kontakty, Google+ a další [15]. Příklad připojení k API společnosti Google je uveden níže.

```
$client = new Google_Client();
        $client->setClientId($clientID);
        $client->setClientSecret($clientSecret);
        $client->setRedirectUri($redURI);
        $client->setScopes($scope);
        $client->setDeveloperKey($devKey);

if (isset($_GET['code'])) {
      $client->authenticate();
      $_SESSION['token'] = $client->getAccessToken();
      echo 'http://'. $_SERVER['HTTP_HOST'] . $_SERVER['PHP_SELF'];
}
```

5 OPTIMALIZACE SQL DOTAZŮ

Cílem optimalizace SQL dotazů, respektive aplikací, které využívají databázi, bývá především zlepšení následujících dvou kritérií [16]:

- **Propustnost systému** – toto kritérium vystihuje počet operací za jednotku času, které je schopen systém vykonat. Z jiného pohledu se jedná o počet současně obsloužených uživatelů systému. Je tedy zřejmé, že cílem optimalizace je zvýšení propustnosti systému.
- **Odezva systému** – tento parametr udává čas, který je potřebný k obdržení výsledků databázového dotazu od serveru. V tomto případě je našim cílem při optimalizaci snížení doby odezvy na co nejmenší hodnotu.

Výše uvedené parametry nezávisí pouze na struktuře databázových dotazů, ale jsou závislé také na celkové struktuře aplikace, nastavení databázového serveru, síťové komunikaci apod. Ladění aplikace by mělo probíhat v několika krocích, mezi které patří například vyhodnocení návrhu aplikace, optimalizace kódu SQL dotazů, optimalizace výkonu databázového serveru a ladění přenosu dat po síti [17 s. 245–246].

Vzhledem k tomu, že aplikace zatím není nasazena v produkčním prostředí, zaměřuje se tato práce především na ladění kódu SQL a využití databázových objektů určených pro optimalizaci SQL dotazů. Testování odezvy systému na lokálním stroji by v tomto případě nemělo praktický význam.

5.1 Obecná pravidla optimalizace databázových dotazů

Aby bylo možné dosáhnout co největší rychlosti vykonávání databázových dotazů, je důležité dodržovat obecná pravidla týkající se jednak návrhu databáze, ale také tvorby samotných SQL dotazů. Popis těchto pravidel je uveden v následujících kapitolách.

5.1.1 Pravidla návrhu struktury databáze

Při návrhu struktury tabulek je třeba dbát na použití správných datových typů, které budou odpovídat datům v příslušných sloupcích. Jedná se například o nesprávné využití řetězce pro uchovávání data. Toto nevhodné použití má za následek nutnost využití různých konverzních funkcí při porovnávání apod. Využívání konverzních funkcí stojí určitý výpočetní čas, tudíž může dojít ke zpomalení databázového systému. Pro datum je vhodné použít příslušný datový typ DATE nebo DATETIME. Obecně platí pravidlo, že pro příslušný

sloupec bychom měli vybírat ten nejpřesnější datový typ vzhledem ke druhu dat, která bude sloupec uchovávat [18 s. 260–270]. Díky použití správného datového typu se můžeme spolehnout na to, že v příslušném sloupci budou validní data. Validitu můžeme kontrolovat pomocí omezení CHECK.

Další častou chybou návrhu databáze bývá využívání příliš velkých datových typů. Například použití datového typu VARCHAR2(4000) nebo VARCHAR(4000) pro všechny sloupce tabulky, které uchovávají řetězec, přestože takto velkého rozsahu nemohou sloupce nikdy dosáhnout. Použití nevhodné velikosti datového typu může vést k zvětšování velikosti tabulky, a tím pádem i ke zvýšení počtu operací s diskem [18 s. 279]. To se v praxi může negativně projevit na rychlosti databázového systému. Rovněž provádění výpočtů bude u delších sloupců probíhat pomaleji než u sloupců krátkých.

5.1.2 Pravidla tvorby SQL příkazů

Při vytváření rychlých SQL příkazů nám jde především o minimalizaci vyhledávací cesty, kterou server použije pro vyhledání dat a dále pak o minimalizaci režie, kterou databázový server musí vykonat (například konverze jednotlivých sloupců při nevhodném spojování tabulek). V následujícím seznamu jsou uvedena obecná doporučení k tvorbě efektivních SQL příkazů [19]:

- Porovnávat sloupce stejných datových typů. V případě, že musí databázový server provádět konverzi různých datových typů, nebude moci využít stávající indexy, čímž zpravidla dojde k prodloužení vykonávání dotazu.
- Podmínky v klauzuli *where* řadit tak, aby na začátku byly podmínky, po kterých bude z možné množiny výsledků vyřazen co největší počet řádků.
- Tam, kde je to žádoucí, deklarovat sloupce s hodnotou *NOT NULL*. Databáze tak nebude muset brát v potaz speciální funkce pro práci s hodnotami *NULL*.
- Vyjmenovat v dotazu názvy požadovaných sloupců
    - Síťová komunikace nebude namáhána přenosem nepotřebných dat.
    - Databázový server nemusí zjišťovat seznam názvů sloupců.
    - Pokud požadujeme pouze hodnoty indexovaných sloupců, budou pro získání dat použity příslušné indexy.
- Omezit používání operátoru *LIKE*. V případech, kdy je to možné, je vhodné nahradit porovnáním na rovnost.

- Indexované sloupce v podmínce where využívat samostatně. Toto doporučení neplatí pro function based indexy.
- V případě složitých vnořených dotazů je vhodné využívat klauzuli with.
- Řazení výsledků provádět podle indexovaných sloupců. Případně pro sloupce, které jsou využívány při řazení, doplnit příslušné indexy.
- Omezit použití klauzulí IN a NOT IN.
- Zvolit správné pořadí spojování tabulek a vhodné sloupce, přes které budou tabulky spojeny. Cílem je využití indexovaných sloupců pro spojení a minimalizace počtu řádků, které budou spojovány.

5.2 Přístupové cesty k datům v databázi Oracle

Přístupová cesta označuje způsob, kterým databázový server získá požadovaná data. Podle způsobu přístupu a prohledávání tabulek rozlišujeme následující přístupové cesty [20]:

- Úplné prohledávání (full table scan).
- Indexové prohledávání (index scan).
- Přímý přístup podle adresy ROWID, nebo hash algoritmu.

5.2.1 Úplné prohledávání

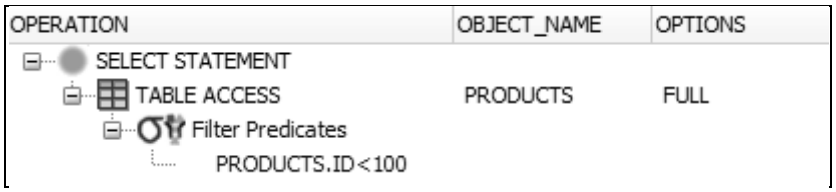
Při tomto typu prohledávání jsou postupně načítány všechny řádky příslušné tabulky a následně jsou na ně aplikovány podmínky v klauzuli where. Jednotlivé bloky jsou načítány sekvenčně. Pro urychlení načítání je vhodné načítat data v rámci jedné vstupně výstupní operace po více blocích, což je možné ovlivnit pomocí parametru DB_FILE_MULTIBLOCK_READ_COUNT.

K použití metody úplného prohledávání dochází zpravidla v následujících případech [20]:

- Databázový server nemá k dispozici vhodný index pro aplikaci indexového vyhledávání.
- Výsledkem databázového dotazu bude velké množství řádků. V tomto případě zvolí optimalizátor úplné prohledávání, přestože má k dispozici vhodné indexy.
- Jedná se o malou tabulku, jejíž záznamy mohou být přečteny v rámci jedné vstupně výstupní operace.

Pro explicitní využití úplného pohledávání můžeme v SQL kódu příkazu využít následující hint[5] */*+ *FULL(NÁZEV_TABULKY)* */*. Níže je uveden konkrétní příklad vynucení úplného prohledávání pro tabulku PRODUCTS. Vzhledem k tomu, že tabulka obsahuje cca 12 000 záznamů a restrikce vybírá pouze 100 řádků (méně než 1 %), optimalizátor standardně využije indexového prohledávání. Při použití hintu bude explicitně využito úplné prohledávání tabulky, jak je uvedeno na následujícím obrázku 6.

```
SELECT
/*+ FULL(PRODUCTS)*/ PRODUCTS.ID, PRODUCTS.NAME
FROM PRODUCTS
WHERE
PRODUCTS.ID<100;
```



Obrázek 6 – Explicitní využití úplného prohledávání

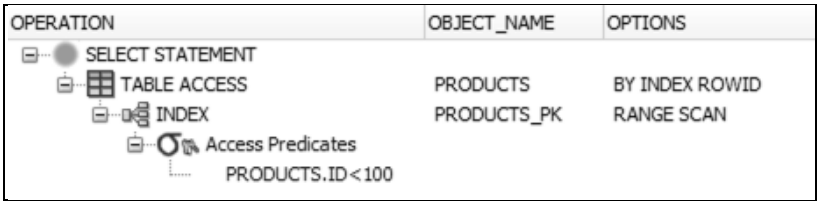
5.2.2 Přímý přístup podle adresy ROWID

ROWID je jedinečný identifikátor řádku, který v sobě obsahuje informaci o datovém souboru s příslušným záznamem, datové stránce a konkrétním záznamu v dané datové stránce [20]. Přístup pomocí adresy ROWID bude nejrychlejší metodou v případě, že chceme získat jeden konkrétní řádek. Díky znalosti příslušného ROWID získáme ihned přesné umístění hledaného řádku.

Optimalizátor použije přístup pomocí ROWID zpravidla jako další krok po hledání prostřednictvím indexu. Prohledáním indexu velice rychle získá příslušná ROWID hledaných řádků a následně je velice rychle načte. Explicitní využití přístupu dle ROWID můžeme vyvolat použitím hintu */*+ *ROWID (NÁZEV_TABULKY)* */*. Praktické použití uvedeného hintu spolu s obrázkem exekučního plánu je ukázáno na následujícím příkladu procházení tabulky PRODUCTS.

[5] Hint slouží jako pomocná instrukce, která optimalizátoru databáze říká, jaký zvolit postup při provádění dané operace.

```
SELECT
/*+ ROWID(PRODUCTS)*/ PRODUCTS.ID, PRODUCTS.NAME
FROM PRODUCTS
WHERE
PRODUCTS.ID<100;
```



Obrázek 7 – Explicitní využití prohledávání dle adresy ROWID

5.2.3 Indexové prohledávání

Jak již z názvu vyplývá, indexové prohledávání využívá pro urychlení přístupu pomocnou datovou strukturu index. Podrobnější informace o typech a strukturách indexů jsou uvedeny v kapitole 5.4.1. Indexové prohledávání můžeme rozdělit na následující typy [22]:

- Prohledávání jedinečného indexu (index unique scan).
- Prohledávání rozsahu indexu (index range scan).
- Sestupné prohledávání rozsahu indexu (index range scan descending).
- Prohledávání s vynecháním sloupců (index skip scan).
- Úplné prohledávání indexu (full scan).
- Rychlé úplné prohledávání indexu (fast full index scan).
- Spojení indexů (index joins).

V následující části budou jednotlivé druhy indexového prohledávání podrobněji popsány. Příslušné informace byly čerpány z optimalizační příručky *Oracle® Database Performance Tuning Guide* [22].

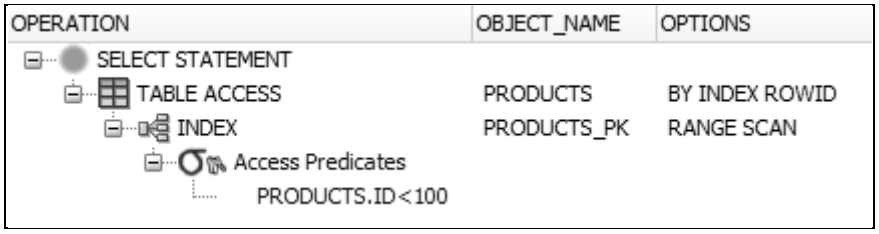
Prohledávání jedinečného indexu – po prohledání indexu bude navrácen nejvýše jeden řádek. Toto prohledávání bude použito v případě, kdy optimalizátor zjistí, že hodnoty daného indexu jsou unikátní (například omezení UNIQUE nebo PRIMARY KEY).

Prohledávání rozsahu indexu – v tomto případě se využívá vlastnosti, kdy listy ve stromové struktuře indexu tvoří seřazený seznam. Při prohledávání se zjistí rozsah, který tvoří dolní hranici rozsahu. Následně je zahájeno prohledávání seznamu listů indexu ve vzestupném pořadí, dokud není nalezen list, který rozsah ohraničuje shora.

Díky tomu, že listy indexu jsou v podstatě seřazený seznam, může server v určitých případech (řazení podle hodnoty indexovaného sloupce) vynechat řazení výsledků a vrátit jednotlivé záznamy v pořadí, které je dáno posloupností v indexu.

Na následujícím příkladu je jasně patrné, že databázový server neprovádí operaci řazení, ale využívá toho, že data z indexu načítá ve vhodném pořadí.

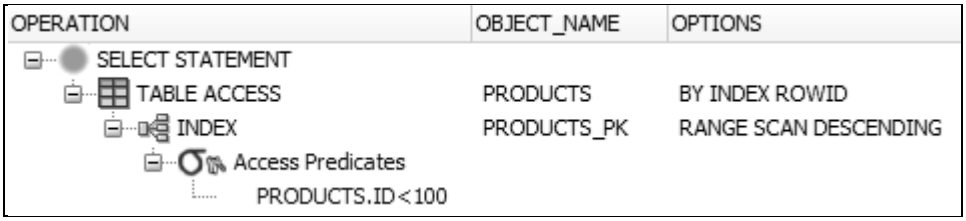
```
SELECT
PRODUCTS.ID, PRODUCTS.NAME
FROM PRODUCTS
WHERE
PRODUCTS.ID<100
ORDER BY PRODUCTS.ID ASC;
```



Obrázek 8 – Prohledávání rozsahu indexu

Sestupné prohledávání rozsahu indexu – postup prohledávání je obdobný jako v předchozím případě s tím rozdílem, že jsou listy indexu načítány v sestupném pořadí. Využití sestupného prohledávání je patrné z následujícího příkladu, kdy jsou data požadována v sestupném pořadí dle indexovaného sloupce.

```
SELECT
PRODUCTS.ID, PRODUCTS.NAME
FROM PRODUCTS
WHERE
PRODUCTS.ID<100
ORDER BY PRODUCTS.ID DESC;
```



Obrázek 9 – Prohledávání rozsahu indexu v sestupném pořadí

Prohledávání s vynecháním sloupců – tento typ prohledávání je v databázovém serveru dostupný od verze Oracle 9. Prohledávání s vynecháním sloupců je využíváno v případě

dotazů na sloupce, které jsou obsaženy ve složeném indexu. Ve starších verzích nedošlo k využití indexu v případě, že první sloupec klauzule where nebyl rovněž prvním sloupcem složeného indexu. Při prohledávání s vynecháním sloupců je možné využít indexového vyhledávání i v případě, že je v klauzuli where uveden sloupec, který není vstupním sloupcem celého indexu.

Úplné prohledávání indexu – v tomto případě je zpracováno pouze minimální množství větví indexu, které je potřebné k nalezení prvního listu. Následně dojde k procházení seznamu listů příslušného indexu. Data jsou opět načítána v seřazeném pořadí, takže v příhodných situacích může dojít k vynechání operace řazení.

Úplné rychlé prohledávání indexu – oproti úplnému prohledávání je tento přístup využíván v případě, kdy jsou všechny požadované sloupce součástí některého indexu. Data tak mohou být získána pouze ze struktury indexu a nemusí docházet k přístupu do tabulky. Díky tomu, že jsou bloky struktury načítány více blokovým čtením, je tento přístup rychlejší než úplné prohledávání indexu. V tomto případě však nedochází k načítání dat v seřazeném pořadí, tudíž není možné vyhnout se operaci řazení.

Spojení indexů – k využití spojení indexů může dojít v případě, že pro každý sloupec specifikovaný v SQL dotazu existuje příslušný index. Aby bylo možné vyhnout se přístupu do tabulky, dojde ke spojení indexů a požadované hodnoty jsou získány přímo ze struktury indexu.

5.2.4 Principy spojení tabulek

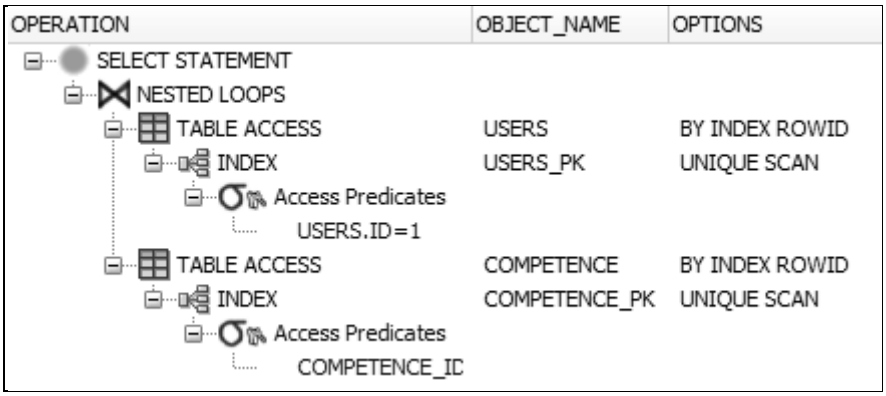
V praxi si málokdy vystačíme se získáním dat pouze z jedné tabulky, proto jsou tabulky spojovány a data získávána napříč několika tabulkami. Pro spojení tabulek, respektive různých databázových objektů, jsou využívány následující principy [22]:

- Spojení vnořenými cykly (nested loop join).
- Spojení hash (hash join).
- Spojení sloučením po seřazení (sort merge join).
- Kartézské spojení (cartesian join).
- Anti spojení (anti join).
- Úplné vnější spojení (full outer join).

V následující části budou jednotlivé druhy spojení podrobněji popsány. Příslušné informace byly čerpány z optimalizační příručky *Oracle® Database Performance Tuning Guide* [22].

Spojení vnořenými cykly – v tomto případě dojde k postupnému procházení řádků první tabulky a pro každý její řádek je pomocí indexu prozkoumána druhá tabulka, ze které jsou získány shodné řádky. Tato technika bývá často využívána, pokud chceme rychle získat první řádek výsledků dotazu, což je vidět na následujícím příkladu.

```
SELECT
EMAIL, COMPETENCE.NAME
FROM
USERS JOIN COMPETENCE ON COMPETENCE_ID=COMPETENCE.ID
WHERE USERS.ID=1;
```

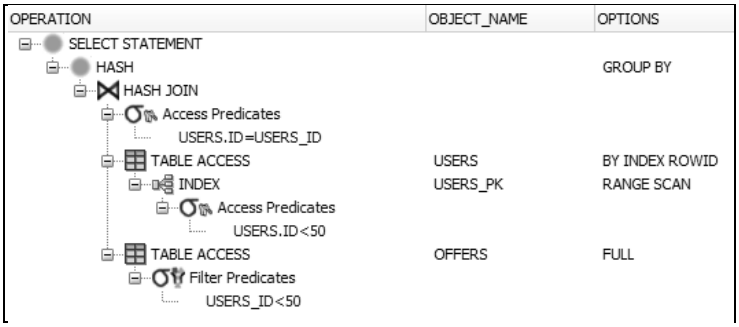


Obrázek 10 – Spojení vnořenými cykly

Spojení hash – toto spojení se využívá v případě, že je nutné připojit k tabulce velké množství řádků. Databázový server vybere menší ze dvou tabulek (menší množinu řádků, která mohla vzniknout aplikací omezení klauzule where), a z této tabulky vytvoří hashovanou tabulku. Hashovací funkce je aplikována na klíčový sloupec, tudíž naleznutí příslušného řádku v tabulce podle klíče bude velice rychlé.

Následně se provede úplné prohledání větší tabulky, kdy se pro každý řádek vyhledává v hash tabulce příslušný záznam. Nevýhodou tohoto spojení je značná časová režie při vytváření hash tabulky. Následné spojování řádků je již velice rychlé. Typické využití hash spojení je patrné z následujícího příkladu, kde tabulka USERS bude po aplikaci podmínky where obsahovat 50 řádků, kdežto tabulka OFFERS obsahuje cca 10 000 řádků.

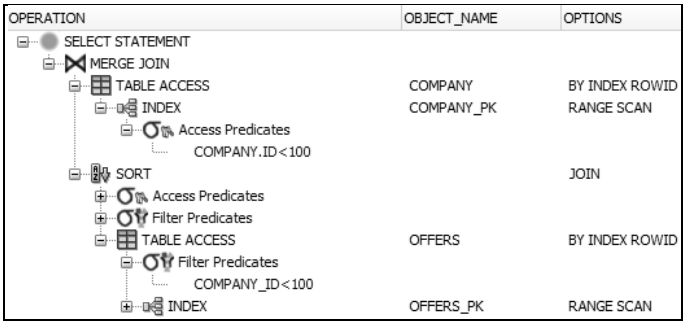
```
SELECT
EMAIL, COUNT(*) as POCET
FROM
USERS, OFFERS
WHERE
USERS.ID=USERS_ID AND USERS_ID<50
GROUP BY EMAIL;
```



Obrázek 11 – Spojení typu hash join

Spojení sloučením po seřazení – jak již z názvu vyplývá, v tomto případě dojde nejprve k seřazení řádků obou vstupních řad a následně až k samotnému spojení. Toto spojení je většinou pomalejší než hash join, nicméně existují situace, kdy bude výhodnější. Spojení sloučením po seřazení bude rychlé v případě, že vstupní sady dat již máme seřazeny (vyhneme se prvotnímu řazení), dále toto spojení může být výhodné v momentě, kdy požadujeme výsledky seřazené (řazení se tak jako tak nevyhneme). Posledním případem, kdy tento typ spojení poskytuje dobré výsledky, je spojování pomocí nerovnic (operátory <, <=, >, >=). Situace, při které databázový server zvolí sloučení po seřazení, je patrná z následujícího příkladu.

```
SELECT
COMPANY.NAME AS NAZEV_FIRMY, OFFERS.NAME AS NABIDKA
FROM
COMPANY, OFFERS
WHERE
COMPANY.ID=COMPANY_ID AND OFFERS.ID<=200 AND COMPANY.ID<100
ORDER BY COMPANY.ID
```



Obrázek 12 – Spojení typu merge join

Kartézské spojení – ke kartézskému spojení dochází v případě, že v dotazu nejsou uvedeny žádné podmínky spojení. Dojde tedy ke spojení všech řádků jedné tabulky se všemi řádky druhé tabulky. Pokud jsou počty řádků obou tabulek velké, může být tato operace značně náročná, jelikož počet výsledných kombinací je dán součinem počtů řádků obou tabulek.

Anti spojení – tento typ spojení bude využit pro získání řádků jedné tabulky, které se v druhé tabulce nenachází. Příklad využití anti spojení v kombinaci s hash spojením je uveden níže.

```
SELECT
PRODUCTS.NAME
FROM
PRODUCTS
WHERE
PRODUCTS.ID NOT IN
(SELECT PRODUCTS_ID FROM OFFERS_PRODUCTS WHERE COUNT<10);
```

Obrázek 13 – Anti spojení

Vnější spojení tabulek je jakýmsi rozšířením vnitřního spojení, kdy výsledek obsahuje kromě řádků splňujících podmínku spojení i řádky z jedné, druhé, nebo obou tabulek, které podmínku nesplňují. Pro vnější spojení tabulek můžeme využít následující metody[22]:

- nested loop outer join,
- hash join outer join,
- sort merge outer join,
- full outer join.

5.3 Přístupové cesty v databázi MySQL

Databázový server MySQL nabízí obdobné přístupy, používané při vyhledávání příslušných záznamů, jako databázový server Oracle. V některých případech, obzvláště u indexového vyhledávání, je počet různých typů metod databáze MySQL nižší než v Oracle. Níže jsou stručně uvedeny možnosti přístupu k datům v databázovém serveru MySQL.

5.3.1 Úplné prohledávání

Princip této techniky spočívá v postupném načítání řádků příslušné tabulky, na které jsou aplikovány restrikce uvedené v SQL dotazu. Technika je obdobná jako u databáze Oracle (podrobněji popsána v kapitole 5.2.1).

5.3.2 Indexové prohledávání

Prohledávání rozsahu indexu (Range Access Method) – tato technika využívá vlastností B-tree indexu, který uchovává své listy v seřazené posloupnosti [23]. Princip fungování je opět obdobný jako u databáze Oracle (podrobněji popsáno v kapitole 5.2.3).

Metoda spojení indexů (Index Merge method) – tato metoda je využívána v případě, že podmínka where obsahuje více omezení, která specifikují různé rozsahy hodnot. Metoda spojení indexů je typicky používána při vytváření průniků nebo sjednocení výsledků dotazů [24].

5.4 Databázové objekty využívané pro optimalizaci

Databázové servery využívají pro zvýšení výkonu různé objekty, které usnadňují vyhledávání dat v tabulkách. Jedná se především o indexy, materializované pohledy nebo dělené tabulky. V následující části jsou podrobněji popsány typy indexů, používané v rámci příslušných databázových platforem.

5.4.1 Indexy v databázi Oracle

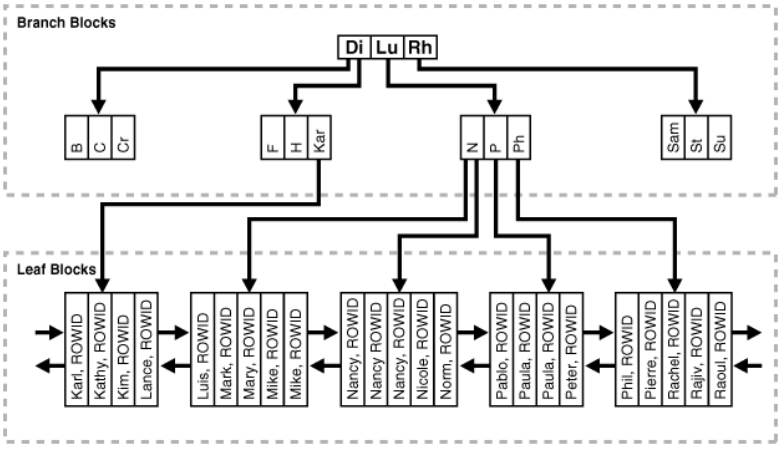
Jednotlivé řádky jsou v databázi uloženy v náhodném pořadí, tudíž databázový server musí prohledávat jednotlivé řádky postupně, než nalezne příslušné záznamy odpovídající zadanému dotazu. Průměrně musí databázový server projít 50 % všech záznamů, což může být velmi zdlouhavé zvláště v případě, kdy se jedná o rozsáhlou tabulku a počet řádků, které vyhovují podmínce dotazu je malý [26 s. 309–311].

Využití indexů nad vhodnými sloupci tabulky značně zrychluje přístup k jednotlivým řádkům v případě, že celkový počet řádků, které chceme získat je malý (do 5% celkového počtu řádků tabulky).

Indexy, dostupné v databázi Oracle 11, můžeme z hlediska principu jejich fungování rozdělit do následujících kategorií:

- B-tree indexy – jedná se o nejčastěji využívaný typ indexu, kdy jsou data uchovávána v B-stromech. Tento typ uchovávání dat využívají také takzvané funkční indexy.
- B-tree cluster indexy.
- Hash cluster indexy.
- Reverse key indexy.
- Bitmap indexy.
- Bitmap join indexy.

Na obrázku 14 je uvedena struktura B-tree indexu databázového serveru Oracle. Z obrázku je patrné, že index je možné rozdělit na dvě části. První z nich jsou bloky větví (branch blocks), které slouží pro traverzování strukturou stromu. Druhou částí jsou pak bloky listů (leaf blocks), které nesou potřebné informace o ROWID indexovaného řádku.



Zdroj: převzato z [27]

Obrázek 14 – Struktura B-tree indexu

Dále jsou uvedeny podrobnější informace, které se týkají dvou nejpoužívanějších druhů indexů v relační databázi.

B-tree index databáze Oracle uchovává na nejnižší úrovni jednotlivé listy, které nesou informaci o identifikátoru ROWID (jedinečný identifikátor bloku, kde se nachází příslušný řádek) a hodnotě příslušného sloupce. K listům se přistupuje traverzováním stromové struktury přes jednotlivé uzly ve stromě (uchovávají především informaci o rozsahu hodnot v listech příslušného podstromu a reference na ostatní uzly stromové struktury). Použití

tohoto typu indexu je vhodné u sloupců, jejichž variabilita je vysoká. K jejich častému využití dochází například u sloupců primárních klíčů.

Bitmapové indexy jsou určeny pro použití u tabulek, ve kterých nedochází k častým změnám dat. Často jsou tedy používány v datových skladech nebo jiných OLAP databázích. Na rozdíl od B-tree indexu se tento typ hodí především pro sloupce s nízkou variabilitou hodnot.

Princip bitmapového indexu spočívá ve vytvoření dvojrozměrného bitového pole, kde každému indexovanému řádku databáze odpovídá jeden sloupec. Počet řádků bitového pole odpovídá počtu unikátních hodnot, které indexovaný sloupec obsahuje. Pro každý řádek je pak v příslušné buňce bitového pole vyplněna hodnota 1 nebo 0 v závislosti na tom, jaká je skutečná hodnota indexovaného sloupce v aktuálním řádku. Při vyhledávání pak databáze provede dekompresi bitmapy, čímž získá odpovídající množinu řádků tabulky.

Je uváděno, že použití bitmapových indexů má velmi pozitivní vliv na výkon v případě, že indexovaný sloupec obsahuje maximálně sedm různých hodnot. V případě, kdy je počet různých hodnot v rozmezí 8 – 100, klesá výkonnost úměrně množství unikátních hodnot. Při počtu různých hodnot větším než 100 výkonnost databázových dotazů rapidně klesá [28]

5.4.2 Indexy v databázi MySQL

Princip fungování indexů je v tomto případě obdobný jako u databáze Oracle. Server MySQL používá následující druhy indexů [29]:

- **B-tree indexy** – jedná se o klasické indexy uchovávané v B-tree stromech. Jejich použití urychluje vykonávání dotazů při porovnávání hodnot sloupců prostřednictvím operátorů =, <, >. Index může být využit i při vhodném použití operátoru LIKE, dále je možné využít jej i pro řazení výsledků.
- **R-tree indexy** – tyto indexy jsou využívány pro takzvané prostorové datové typy, tedy typy, které v sobě uchovávají určitou kolekci dat vymezující nějakou oblast (např. typ POLYGON).
- **Hash indexy** – využití těchto indexů je vhodné při porovnávání hodnot sloupců pomocí operátorů rovnosti nebo nerovnosti, což je dáno principem fungování hashovacích tabulek, kdy je pro jednu hodnotu sloupce spočítána příslušná hodnota hashovaní funkce. Výsledek se použije jako klíč pro získání hledaného řádku. Oproti B-tree indexům nelze tyto indexy využít pro řazení výsledků, rovněž je nelze použít s operátorem *LIKE*.

6 PRAKTICKÝ POSTUP OPTIMALIZACE SQL DOTAZŮ

V této kapitole je popsán postup, který byl zvolen pro optimalizaci databázových dotazů. Dále je zde uvedena praktická aplikace zvoleného postupu pro optimalizaci konkrétních databázových dotazů. Vzhledem k tomu, že předmětem této práce měla být především optimalizace databázových dotazů z hlediska jejich exekučních plánů, nikoli optimalizace nastavení databázového serveru na příslušném hardwaru, byly všechny databázové dotazy testovány při základní[6] konfiguraci příslušného serveru. Konfigurační soubory obou databázových serverů jsou uvedeny na přiloženém CD.

6.1 Naplnění databáze testovacími daty

Jelikož aplikace zatím nebyla používána v reálném provozu, je nutné zajistit naplnění databáze testovacími daty. Databáze byla naplněna daty prostřednictvím generátoru pseudonáhodných dat, který byl napsán v jazyce PHP. Generátor byl koncipován tak, aby ve většině případů generoval smysluplná data (jména osob, adresy apod.). K naplnění tabulky produktů byl využit XML import z existujícího elektronického obchodu s velkým množstvím produktů. Díky tomu byly produkty naplněny reálnými daty včetně cen a obrázků.

Data byla generována do databáze Oracle. Následně byl proveden export těchto dat do databáze MySQL. Pro zajištění exportu dat byl využit program *Navicat Premium*[7], který umožňuje transfery dat mezi různými databázovými platformami. Migraci dat je možné provádět přímo mezi dvěma databázovými servery, nebo je možné exportovat data do souboru, který je následně využit pro import. Vzhledem k tomu, že velikost souboru s testovanými daty byla značná, datový transfer probíhal přímo na úrovni databázových serverů. Díky tomu se neobjevily problémy s limity nástroje *PHP MyAdmin* (běžně používaný správce databáze MySQL) pro nahrávání velkých souborů prostřednictvím protokolu http.

Přibližné počty řádků v jednotlivých databázových tabulkách jsou uvedeny v následující přehledové tabulce. V tabulce nejsou uvedeny počty řádků pro tabulky standardních číselníků (počty těchto řádků jsou z principu v řádu jednotek či desítek).

[6] V případě databáze MySQL byl využit výrobcem přednastavený konfigurační soubor, který je určen pro velké databázové systémy.
[7] Zkušební verze programu je dostupná z www.navicat.com/products/navicat-premium.

Tabulka 6 – Počty vygenerovaných řádků jednotlivých tabulek

Tabulka	Počet řádků	Tabulka	Počet řádků
ADDRESS	13812	**PERSONS_ADDR**	6487
CATEGORIES	519	**PERSONS_CONT**	6488
COMMUNICATION	90202	**PRICES**	11962
COMPANY	2443	**PRODUCTS**	11960
CONTACTS	6492	**PRODUCTS_SUPPLIERS**	11962
EMAILS	5012	**RECIPIENTS**	28699
IMAGES	11969	**TASKS**	5234
OFFERS	9109	**USERS**	100
PERSONS	6488		

V případě potřeby je možné počty řádků získat dotazem do systémového katalogu. Pro databázový server Oracle může dotaz vypadat následovně.

```
SELECT TABLE_NAME as TABULKA,
TO_NUMBER(EXTRACTVALUE(XMLTYPE(DBMS_XMLGEN.GETXML('SELECT COUNT(*) CNT FROM
'||TABLE_NAME)),'/ROWSET/ROW/CNT')) AS POCET_RADKU
FROM USER_TABLES where TABLE_NAME  not like 'MD_%' and TABLE_NAME  not like
'APEX%' and TABLE_NAME  not like 'MIGR%';
```

V případě databázového serveru MySQL je možné počty řádků v tabulkách získat níže uvedeným dotazem.

```
SELECT TABLE_NAME as TABULKA, TABLE_ROWS AS POCET_RADKU
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'dealer';
```

6.2 Provedení standardních scénářů

Po naplnění daty byl u aplikace simulován běžný provoz. Tato simulace byla provedena na základě připraveného scénáře, který pokrývá běžné použití testované aplikace. V průběhu provádění testů byly zaznamenávány jednotlivé databázové dotazy včetně informací o době provádění příslušného databázového dotazu. Na základě těchto informací byly vybrány ty databázové dotazy, které nejvíce zpomalovaly práci se systémem. Vybrané dotazy byly v následujícím kroku podrobeny optimalizaci s cílem zvýšit jejich rychlost.

6.2.1 Scénář pro testování databázových dotazů

Pro získání statistiky byl zvolen následující scénář, který vystihuje nejčastější operace[8] prováděné v rámci systému. Tento scénář byl několikrát proveden nad oběma databázovými servery a prováděcí časy jednotlivých SQL dotazů byly zaznamenány. Uvedené časy, potřebné k vykonávání databázového dotazu, byly měřeny pomocí nástroje knihovny dibi, který je určen pro logování. Tyto časy tedy odrážejí chování databáze při reálném běhu aplikace a mohou se lišit od časů získaných specifickými softwarovými nástroji používanými při testování alternativních exekučních plánů.

Tabulka 7 – Testovací scénář

Scénář testování databázových dotazů			
Krok	**Operace**	**ID**	**Parametry**
1	Přihlášení		Uživatelské jméno: zeleny.petr@kontaktkatalog.cz
2	Zobrazení úvodní obrazovky se statistickými informacemi		
3	Zobrazení jedné rozpracované nabídky	5719	
4	Zobrazení seznamu nabídek		
5	Filtrování seznamu nabídek		Stav: odeslaná; firma: iXXnSVNiNXL s.r.o.
6	Zobrazení nabídky	3931	
7	Vytisknutí nabídky	3931	
8	Zobrazení seznamu produktů		
9	Filtrování seznamu produktů		Kategorie: houpačky; název: Kohout houpací
10	Zobrazení konkrétního produktu	3085	
11	Zobrazení seznamu firem		
12	Filtrování seznamu firem		Ulice: Střelecká 819; město: Zásmuky
13	Zobrazení konkrétní firmy	1703	
14	Zobrazení seznamu osob		
15	Filtrování seznamu osob		Příjmení: Trubáková
16	Zobrazení konkrétní osoby	5937	
17	Zobrazení seznamu kontaktů		
18	Filtrování seznamu kontaktů		Typ: email
19	Zobrazení konkrétního kontaktu	1501	
20	Zobrazení seznamu emailů		
21	Filtrování seznamu emailů		Předmět: UAsDc
22	Zobrazení konkrétního emailu	2490	
23	Zobrazení seznamu úkoů		
24	Filtrování seznamu úkolů		Stav: Urgentní
25	Zobrazení konkrétního úkolu	4135	

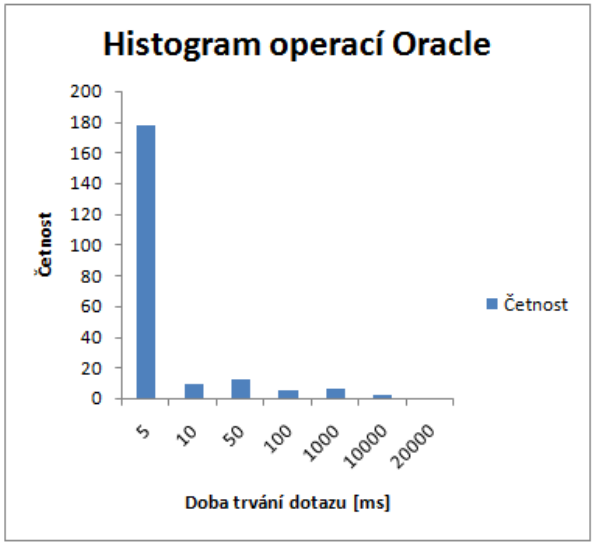
Provedením výše uvedeného scénáře vznikla statistika, která pokrývá 212 databázových dotazů[9]. Ze zaznamenaných dob provádění jednotlivých databázových dotazů byly získány průměrné hodnoty a vytvořeny jejich histogramy.

[8] Zmiňovaný scénář obsahuje pouze databázové dotazy pro výběr dat. Dotazy vkládání, úpravu a mazání nebyly do scénáře zahrnuty, jelikož cílem testování bylo odhalit náročné databázové dotazy, kterými jsou právě výběry dat z většího množství spojených tabulek.

[9] Některé databázové dotazy se mohou ve scénáři opakovat. Například výpisy hodnot číselníků pro pole filtrů. Každý krok scénáře se může skládat z většího množství databázových dotazů.

Histogram doby provádění operací nad databázovým serverem Oracle

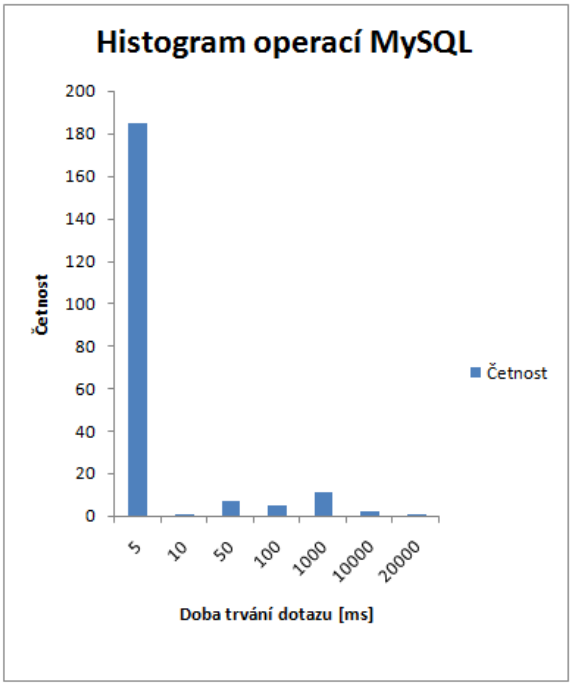
Doba trvání dotazu [ms]	*Četnost*
5	178
10	9
50	12
100	5
1000	6
10000	2
20000	0



Obrázek 15 – Histogram doby provádění operací pro server Oracle

Histogram doby provádění operací nad databázovým serverem MySQL

Doba trvání dotazu [ms]	*Četnost*
5	185
10	1
50	7
100	5
1000	11
10000	2
20000	1



Obrázek 16 – Histogram doby provádění operací pro server MySQL

Z histogramu vyplývá, že většina databázových dotazů je vykonávána velmi rychle (v rámci jednotek, nebo desítek milisekund)[10], nicméně se zde objevilo i několik dotazů, jejichž zpracování probíhá v rámci jednotek i desítek sekund. Jedná se především o statistické dotazy, výpisy přehledů nabídek a výpisy komunikace. Tyto náročné dotazy budou předmětem praktické optimalizační části, kde dojde k podrobnému rozebrání jejich exekučních plánů.

6.3 Optimalizace dotazů pro databázový server Oracle

Po provedení testovacích scénářů byly vybrány příslušné SQL dotazy, jejichž doba provádění byla značně vysoká. Jedná se o následující dotazy.

1. **Zobrazení seznamu nabídek** včetně kalkulace celkových cen, zisků a nákladů. Dotaz je dále uváděn jako dotaz č. 1.1.
2. **Výpočet statistiky nabídek pro hlavní stránku**. Dotaz je dále uváděn jako dotaz č. 1.2.
3. **Výpis komunikace.** Dotaz je dále uváděn jako dotaz č. 1.3.

SQL kódy jednotlivých dotazů jsou uvedeny na přiloženém CD, které je součástí této práce. Při optimalizaci dotazů pro databázový server Oracle byl nejprve analyzován exekuční plán získaný pomocí programu Oracle SQL Developer. V případě, že se z exekučního plánu podařilo rozpoznat problematické pasáže, byl kód dotazu ručně upraven, nebo byly například do databáze doplněny indexy či jiné databázové objekty. Dalším krokem bylo generování alternativních exekučních plánů. Vzhledem k tomu, že generování alternativních exekučních plánů je náročnou operací, byl pro tuto část zvolen specializovaný software, který je v praxi běžně používán. Konkrétně se jedná o program *Quest SQL Optimalizer*[11]. Aby mohl tento program navrhovat a testovat alternativy exekučních plánů, je třeba zajistit, že uživatelský účet používaný pro testování bude mít přístupy k pohledům *SYS.V_$SQLAREA, SYS.V_$SQLTEXT, SYS.V_$SESSION, SYS.V_$OPEN_CURSOR, SYS.V_$SQL_PLAN*. V případě, že účet nebude mít dostatečné oprávnění, skončí veškerá testování chybou.

Generování alternativ exekučních plánů bylo prováděno pro dotaz s minimálními restriktivními podmínkami (ve většině případů se jednalo o omezení záznamů podle ID přihlášeného uživatele). Případné další podmínky mají na rychlost většinou pozitivní vliv

[10] Jedná se především o standardní výpisy z jednotlivých tabulek a číselníků, které nevyžadují spojení několika velkých tabulek.
[11] Zkušební verze programu je dostupná z www.quest.com/sql-optimizer-for-oracle.

(dochází k omezení počtu řádků vstupujících do spojení tabulek, případně může být použito rychlé indexové vyhledávání).

Statistika rychlosti pro každý databázový dotaz, která byla měřena pomocí nástroje Oracle SQL Developer, obsahuje následující sady měřených časů:

- **Čas provádění dotazu při využití paměti cache a pomocných bufferů** – v tomto případě byly časy měřeny za běžného provozu aplikace. Databázový server tedy mohl využívat data uložená v paměti cache a v pomocných bufferech. Tato měření odrážejí chování aplikace za běžného provozu. Rychlost se v tomto případě může měnit v závislosti na parametru *Cache hit ratio*. Tento parametr vystihuje, do jaké míry dochází k načítání potřebných dat z paměti cache. V tabulkách a grafech, které zaznamenávají hodnoty těchto měření, jsou příslušné sloupce uváděny s identifikátorem ve tvaru *číslo dotazu – cache*.
- **Čas provádění dotazu bez využití paměti cache** – v tomto případě byl vliv paměti cache na rychlost dotazu eliminován jejím vyprázdněním. Tohoto stavu bylo docíleno restartováním databázového serveru před každým měřením. V tabulkách a grafech, které zaznamenávají hodnoty těchto měření, jsou příslušné sloupce uváděny s identifikátorem ve tvaru *číslo dotazu*.

Pro každý dotaz, který byl podroben testování alternativních exekučních plánů, byla získána tabulka, ve které jsou zhodnoceny následující parametry popisující náročnost provádění SQL dotazu:

- **Cena exekučního plánu** (popisuje teoretickou náročnost zpracování dotazu). Ve statistikách je uváděna pod označením *Plan cost*.
- **Celkový čas provádění dotazu**, který je měřen serverem. Ve statistikách je uváděn pod označením *Elapsed time*.
- **Celkový počet bloků načtených z disku**. Ve statistikách je uváděn pod označením *Physical Reads*.
- **Celkový počet bloků načtených z operační paměti a sekundární diskové paměti**. Ve statistikách je uváděn pod označením *Logical Session Reads*.
- **Celkový počet tříděných řádků**. Ve statistikách je uváděn pod označením *Sorts*.

Ze získaných výsledků byl vybrán nejlepší alternativní plán. V následující části je proveden rozbor zvolených dotazů, který obsahuje popis SQL dotazu z hlediska jeho funkčnosti,

obrázek původního exekučního plánu spolu s jeho popisem (pokud to jeho velikost umožňuje) a analýzu alternativních exekučních plánů získaných pomocí výše popisovaného programu *Quest SQL Optimalizer*.

6.3.1 Optimalizace dotazu pro výpis seznamu nabídek – dotaz č. 1.1

Výsledkem dotazu je seznam nabídek spolu s výpočtem celkové nákupní ceny všech produktů příslušné nabídky, celkové prodejní ceny a celkové marže pro každou z vypisovaných nabídek. Databázový dotaz je tvořen spojením jedenácti tabulek, tudíž nároky na spojování jednotlivých tabulek při vysokém počtu řádků mohou být velmi velké.

Vzhledem ke značnému rozsahu zde není konkrétní SQL kód uveden. Stejně tak zde není uveden obrázek exekučního plánu. Kód SQL dotazu i obrázek exekučního plánu je možné nalézt na přiloženém CD.

Z vygenerovaného exekučního plánu bylo zřejmé, že při vykonávání dotazu dochází k prohledávání tabulek metodou úplného prohledávání, která spolu s vnějšími spojeními těchto tabulek značně zvyšuje cenu dotazu.

Návrh optimalizovaného kódu SQL

SQL dotaz byl nejprve zjednodušen tak, že z vnořeného dotazu, nad kterým je prováděna agregační funkce pro získání celkového počtu záznamů, byly vypuštěny nepotřebné podmínky a spojení tabulek, jež nemají na výsledný počet řádků vliv. Dále byl do dotazu doplněn další vnořený dotaz, jehož cílem je omezit co nejvíce počet řádků tabulky OFFERS ještě před vstupem do operace spojení tabulek. Tyto úpravy měly sice vliv na zjednodušení exekučního plánu, nicméně doba provádění dotazu se významně nezměnila. Tato situace byla dána tím, že stále docházelo k úplnému prohledávání velké tabulky OFFERS. Řešením této situace by mohlo být vytvoření složeného indexu nad tabulkou OFFERS, kdy budou indexovány sloupce ID, USERS_ID a COMPANY_ID. Tyto sloupce jsou často obsaženy v podmínce where, tudíž by při vhodném způsobu prohledávání mohl být využit nový index. Níže je uveden SQL kód pro vytvoření indexu.

```
CREATE INDEX OFFERS1_IDX on OFFERS (ID, USERS_ID, COMPANY_ID);
```

Z nově vygenerovaného exekučního plánu vyšlo najevo, že index byl opravdu využit. Tabulka OFFERS byla prohledávána pomocí indexového prohledání s vynecháním sloupců (index

skip scan). Díky tomu klesla cena exekučního plánu přibližně na jednu třetinu oproti původnímu dotazu.

Automatické generování alternativních exekučních plánů dotazu č. 1.1

Pomocí programu *Quest SQL Optimizer* byly automaticky generovány alternativní exekuční plány s cílem dalšího možného zlepšení výkonnosti dotazu. V tomto případě se podařilo vygenerovat několik exekučních plánů, jejichž výkon byl lepší než u původního dotazu 1B. Níže je uvedena tabulka s vybranými výsledky testování alternativních exekučních plánů (celkem bylo vygenerováno 133 alternativ, proto tabulka obsahuje pouze výběr několika plánů). Z obrázku vyplývá, že ne vždy je prováděcí čas příkazu úměrný ceně jeho exekučního plánu. Nejrychlejší alternativa (Alt67) má v tomto případě zhruba třikrát větší cenu, nicméně rychlost provádění je přibližně poloviční.

Scenario Name	Plan Cost	Elapsed Time	Physical Reads	Session Logical Reads	Sorts (Rows)
Alt67	1 439	00:00:00.48	0	952	134 349
Alt78	1 439	00:00:00.51	0	953	134 358
Alt77	1 437	00:00:00.56	0	953	131 801
Alt107	1 233	00:00:00.62	0	919	134 244
Alt108	1 273	00:00:00.63	0	919	137 879
Alt37	1 270	00:00:00.64	0	919	131 691
Alt109	1 272	00:00:00.64	0	919	134 354
Alt28	1 272	00:00:00.66	0	920	131 804
Alt38	1 271	00:00:00.66	0	920	134 248
Alt132	2 668	00:00:00.69	0	920	137 879
Alt30	1 272	00:00:00.70	0	920	137 878
Alt69	1 440	00:00:00.71	0	953	137 988
Alt59	557	00:00:00.73	0	953	16 024
Alt65	560	00:00:00.73	0	953	18 688
Alt75	556	00:00:00.74	0	953	16 024
Alt76	558	00:00:00.74	0	952	16 140
Alt61	557	00:00:00.76	0	953	16 024
Original	288	00:00:00.93	0	920	1 311

Obrázek 17 – Statistika alternativních exekučních plánů dotazu 1.1 B

Zmiňovaný alternativní plán Alt67 využívá hintu pro specifikaci druhu spojení jednotlivých tabulek, kdy je explicitně vyžadováno spojení typu *merge join* pro všechny tabulky uvedené v kódu příslušného hintu.

```
/*+
USE_MERGE(VAT,PRICES,UNITS,SUPPLIERS,USERS,PRICE_LEVEL,COMPANY,OFFER_STATES
,PRODUCTS_TYPES,PRODUCTS,OFFERS_PRODUCTS)
*/
```

Vyhodnocení optimalizace dotazu č. 1.1

Po rozboru původního exekučního plánu byly identifikovány příčiny, které způsobovaly prodlužování prováděcí doby SQL dotazu. Nejprve byla provedena základní optimalizace spočívající v doplnění nového indexu nad sloupci tabulky, které byly využívány pro aplikaci filtru vyhledávání. Díky doplnění indexu došlo k poměrně velké časové úspoře. Následně byl dotaz podroben automatickému generování alternativních exekučních plánů. Celkem bylo vygenerováno 133 alternativ, z nichž 17 dosáhlo lepšího prováděcího času. Z těchto alternativ byl vybrán exekuční plán s označením Alt67, který dosahoval nejlepších výsledků. Konkrétní naměřené časy jsou uvedeny v následující tabulce.

Tabulka 8 – Časová statistika dotazu č. 1.1

			Čas provádění [s]				
Dotaz		1.1 A	1.1 A - cache	1.1 B	1.1 B - cache	1.1 B Alt67	1.1 B Alt67 - cache
Číslo měření	1	2,939	2,195	1,698	0,199	1,678	0,115
	2	2,921	2,155	1,721	0,194	1,598	0,118
	3	2,950	1,876	1,710	0,198	1,652	0,117
	4	2,943	2,273	1,705	0,195	1,655	0,112
	5	2,926	2,048	1,711	0,198	1,649	0,117
Průměr [s]		**2,936**	**2,109**	**1,709**	**0,197**	**1,646**	**0,116**

U alternativního exekučního plánu došlo ke zvýšení rychlosti i přesto, že cena tohoto plánu byla vyšší, než u dotazu původního.

6.3.2 Optimalizace dotazu pro výpočet statistiky na hlavní straně – dotaz č. 1.2

Tento dotaz patří k nejsložitějším, jelikož jeho výstupem je statistika nabídek seskupená dle jednotlivých měsíců a jednotlivých stavů nabídek včetně kalkulace celkových příjmů a nákladů z příslušného agregovaného setu nabídek. Součástí statistiky jsou také celkové počty nabídek daného typu a sumarizace jejich celkového počtu produktů. Pro provádění optimalizace byla volena statistika za rok 2012, jelikož testovací data byla generována s limitem v rozmezí let 2010 – 2012.

Vzhledem k velkému rozsahu zde není konkrétní SQL kód uveden. Stejně tak zde není uveden obrázek exekučního plánu. Kód SQL dotazu i obrázek exekučního plánu je možné nalézt na přiloženém CD.

Z exekučního plánu bylo patrné, že ke spojování tabulek dochází obdobným postupem jako u předchozího příkazu č. 1.1. Hlavní rozdíl spočívá v tom, že je spojeno méně tabulek, ale nad nimi jsou prováděny agregační funkce a seskupování prostřednictvím klauzule

group by. Negativní vliv na dobu provádění příkazu mělo opět procházení tabulky OFFERS prostřednictvím operace úplného prohledávání.

Návrh optimalizovaného kódu SQL

Vzhledem k tomu, že struktura dotazu je obdobná předchozímu případu, jako řešení se opět nabízí doplnění složeného indexu nad sloupce primárních a cizích klíčů. Oproti předchozímu případu není nutné do indexu zahrnout sloupec COMPANY_ID, jelikož tento sloupec není využíván k filtrování ani spojování tabulky.

Při testování exekučního plánu bylo provedeno měření rychlosti, jak pro použití indexu bez zahrnutí sloupce COMPANY_ID, tak i s použitím indexu OFFERS1_IDX, který byl vytvořen v rámci optimalizace předchozího dotazu č. 1.1. Vzhledem k tomu, že zrychlení provádění dotazu v případě použití indexu bez sloupce COMPANY_ID bylo prakticky zanedbatelné, byl pro optimalizaci využit stávající index OFFERS1_IDX. Použití dvou podobných indexu by se negativně projevilo při vkládání záznamů do tabulky, jelikož by musely být aktualizovány dva databázové objekty.

Po vygenerování nového exekučního plánu se ukázalo, že došlo k obdobnému zrychlení jako u předchozího dotazu. Ke zrychlení došlo díky využití indexového prohledávání s vynecháním sloupců při aplikaci omezující podmínky where. V dalším kroku byl SQL kód podroben automatickému generování alternativních exekučních plánů.

Scenario Name		Plan Cost	Elapsed Time	Physical Reads	Session Logical Reads	Sorts (Rows)
Alt8	✓	1 247	00:00:00.49	0	827	130 466
Alt10	✓	1 249	00:00:00.50	0	827	130 515
Alt11	✓	1 247	00:00:00.65	0	823	130 509
Alt7	✓	368	00:00:00.71	0	826	14 872
Alt5	✓	266	00:00:00.73	0	822	1 810
Alt9	✓	369	00:00:00.77	0	827	15 448
Alt1	✓	265	00:00:00.98	0	818	1 810
Alt2	✓	809	00:00:00.99	0	1 390	1 810
Alt12	✓	266	00:00:01.16	0	827	1 771
Alt6	✓	367	00:00:01.20	0	822	14 915
Alt13	✓	368	00:00:01.25	0	826	14 876
Original	✓	266	00:00:01.50	0	822	1 810

Obrázek 18 – Statistika alternativních exekučních plánů dotazu 1.2

Z výše uvedeného obrázku vyplývá, že bylo nalezeno celkem 11 rychlejších exekučních plánů. Celkový počet vygenerovaných alternativ byl 13. Nejlepšího výkonu dosáhl alternativní plán označený jako Alt8, který využívá následující hint.

```
/*+
USE_MERGE(VAT,PRICES,PRODUCTS_TYPES,PRODUCTS,OFFERS_PRODUCTS) ORDERED
*/
```

Díky použití tohoto hintu bude explicitně vyžadováno spojení tabulek pomocí operace *merge join* v daném pořadí.

Vyhodnocení optimalizace dotazu č. 1.2

V případě tohoto dotazu se ukázala velká podobnost s dotazem předchozím. Úspěšně byl tedy využit index použitý při optimalizaci dotazu č. 1.1. Využití dalšího (specifického) indexu bylo zamítnuto z důvodu zvýšení režie při zápisu dat do tabulky. Pomocí automatizovaného programu byly vygenerovány alternativní exekuční plány, které dosahovaly lepšího prováděcího času, než původní dotaz. Níže je uvedena tabulka naměřených časů pro dotaz č. 1.2. Ve sloupci 1.2 B1 je uveden čas měřený pro index OFFERS1_IDX, ve sloupci 1.2 B2 je pak čas měřený při využití speciálního indexu pouze nad sloupci ID a USERS_ID v tabulce OFFERS.

Tabulka 9 – Časová statistika dotazu č. 1.2

		Čas provádění [s]							
Dotaz		1.2 A	1.2 A - cache	1.2 B1	1.2 B1 - cache	1.2 B2	1.2 B2 - cache	1.2 B Alt8	1.2 B Alt8 - cache
Číslo měření	1	3,186	2,322	1,449	0,037	1,421	0,039	1,457	0,097
	2	3,179	2,327	1,389	0,039	1,429	0,034	1,461	0,095
	3	3,188	2,337	1,385	0,039	1,436	0,038	1,452	0,090
	4	3,195	2,752	1,457	0,036	1,420	0,035	1,456	0,093
	5	3,191	2,599	1,481	0,035	1,426	0,033	1,450	0,093
Průměr [s]		**3,188**	**2,467**	**1,432**	**0,037**	**1,426**	**0,036**	**1,455**	**0,094**

Při praktickém testování (při běhu aplikace) alternativního plánu Alt8 se ukázalo, že dosahuje horšího času (sloupec 1.2 B Alt8 tabulky 9) než dotaz původní. Z tohoto důvodu nebyl příslušný hint do aplikace zahrnut.

6.3.3 Optimalizace dotazu pro výpis komunikace – dotaz č. 1.3

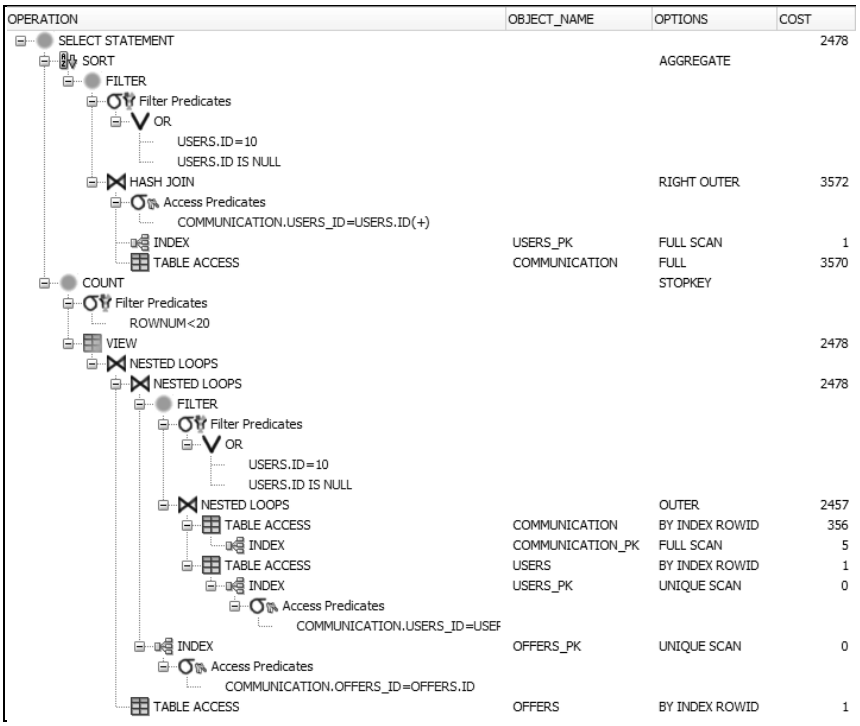
Výpis jednotlivých záznamů komunikace je velmi náročnou operací především proto, že tabulka komunikace obsahuje velké množství záznamů (v testovací sestavě cca 90 000 řádků). Výpis komunikace se skládá především ze zpráv a příslušných odpovědí, které jsou doplněny informací o uživateli, který zprávu vytvořil. Dále je zde uvedena informace o nabídce, ke které se zpráva váže.

Níže je uveden konkrétní SQL kód dotazu, v další části pak jeho exekuční plán.

```
SELECT * FROM(
SELECT
(SELECT COUNT(*) FROM  COMMUNICATION LEFT JOIN USERS ON
COMMUNICATION.USERS_ID=USERS.ID
JOIN OFFERS ON OFFERS_ID=OFFERS.ID WHERE ( USERS.ID=10 OR USERS.ID IS NULL
)) AS TC,
COMMUNICATION.ID AS ID, OFFERS_ID, USERS.SURNAME AS SURNAME, OFFERS.NAME AS
ONAME, MESSAGE,
ANSWER, COMMUNICATION.CREATED AS COMCREATED FROM COMMUNICATION LEFT JOIN
USERS ON COMMUNICATION.USERS_ID=USERS.ID
JOIN OFFERS ON OFFERS_ID=OFFERS.ID WHERE ( USERS.ID=10 OR USERS.ID IS NULL
)  ORDER BY COMMUNICATION.ID ASC
) WHERE ROWNUM < 20;
```

Z exekučního plánu dotazu na obrázku 19 vyplývá, že mezi nejnáročnější operace patří vnější spojení tabulek COMMUNICATION a USERS, které je realizováno pomocí operace vnořených cyklů. Rovněž je zde patrné, že k aplikaci filtru na sloupec USERS_ID dochází až po provedení spojení, tudíž je spojováno zbytečně velké množství řádků. K množině řádů, která vznikne popisovanou operací spojení, je v dalším kroku připojena tabulka OFFERS. Toto spojení probíhá pomocí hodnoty ROWID, získané z indexu tabulky OFFERS, tudíž tato operace bude velmi rychlá. Nad výslednou množinou je následně aplikováno omezení počtu výstupních řádků (v plánu uvedeno jako COUNT STOPKEY).

Další nákladnou operací je provedení agregační funkce COUNT ve vnořeném dotazu, která je prováděna nad spojením tabulky COMMUNICATION s hodnotami indexu tabulky USERS (index USERS_PK). V tomto případě opět probíhá spojování ještě před aplikací filtru na sloupec identifikátoru uživatele.



Obrázek 19 – Exekuční plán dotazu 1.3 A

Návrh optimalizovaného kódu SQL

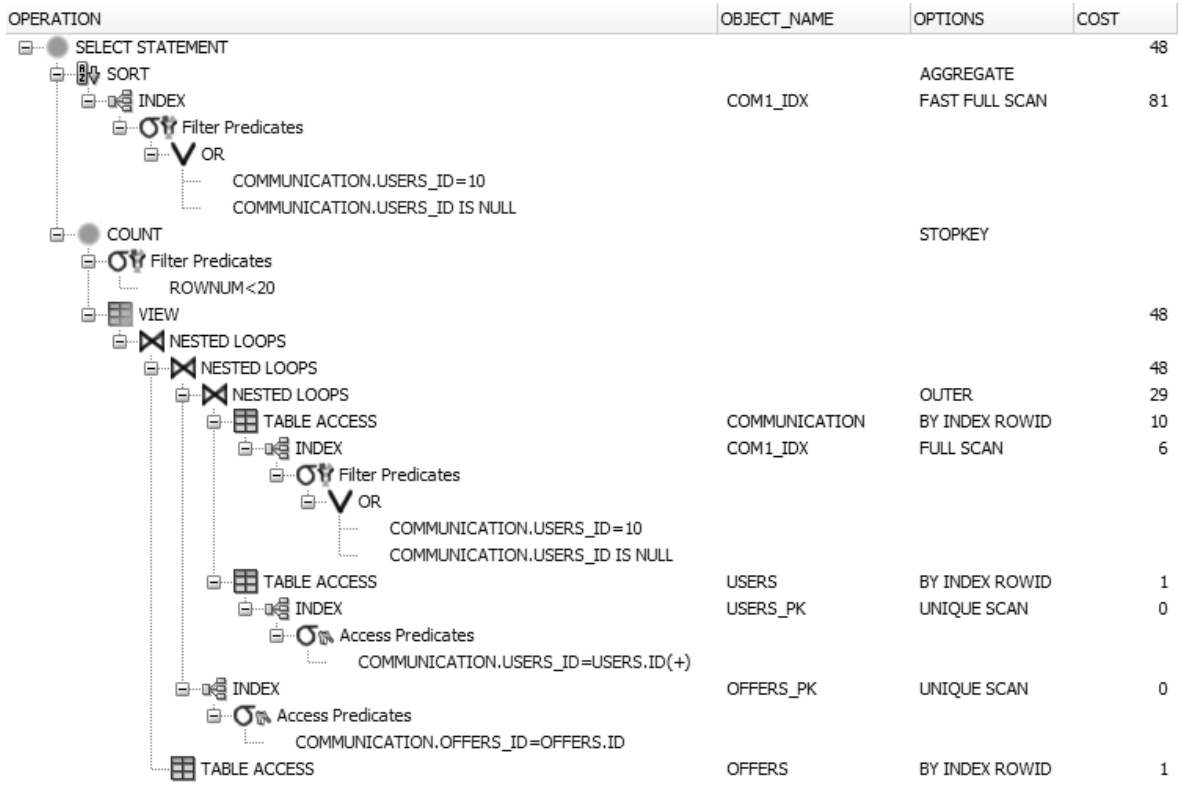
Vzhledem k tomu, že podmínka where může dle současného nastavení aplikace obsahovat pouze sloupce, které jsou primárními, nebo cizími klíči v tabulce COMMUNICATION, bylo by vhodné vytvořit nad tabulkou další složený index, který by indexoval právě všechny sloupce cizích a primárních klíčů. V tomto případě by mohlo být využito úplné, nebo úplné rychlé prohledání indexu, kdy jsou hodnoty sloupců získány velice rychle přímo ze struktury indexu. Aby mohl být nový index využit, je třeba změnit podmínku v klauzuli where tak, aby odkazovala na sloupce obsažené v indexu (nikoli na sloupce identifikátorů tabulek). Příkaz pro vytvoření indexu a výsledný dotaz bude vypadat následovně.

```
CREATE INDEX COM1_IDX ON COMMUNICATION (ID, USERS_ID, OFFERS_ID);

SELECT * FROM(
SELECT (SELECT COUNT(*) FROM  COMMUNICATION LEFT JOIN USERS ON
COMMUNICATION.USERS_ID=USERS.ID
JOIN OFFERS ON OFFERS_ID=OFFERS.ID WHERE   ( COMMUNICATION.USERS_ID=10 OR
COMMUNICATION.USERS_ID IS NULL ) ) AS TC,
COMMUNICATION.ID AS ID, OFFERS_ID, USERS.SURNAME AS SURNAME, OFFERS.NAME AS
ONAME, MESSAGE,
ANSWER, COMMUNICATION.CREATED AS COMCREATED  FROM  COMMUNICATION LEFT JOIN
USERS ON COMMUNICATION.USERS_ID=USERS.ID
JOIN OFFERS ON OFFERS_ID=OFFERS.ID WHERE( COMMUNICATION.USERS_ID=10 OR
COMMUNICATION.USERS_ID IS NULL )  ORDER BY COMMUNICATION.ID ASC
) WHERE ROWNUM < 20;
```

Na níže uvedeném obrázku 20 je zobrazen exekuční plán upraveného dotazu. Z exekučního plánu je patrné, že opravdu došlo k využití indexu COM1_IDX, na základě kterého je získána množina řádků tabulky COMMUNICATION odpovídající podmínce where. Až po aplikaci filtru uvedeného v podmínce je zahájena operace spojení tabulek COMMUNICATION a USERS. Přestože se nadále jedná o vnější spojení, je cena tohoto spojení řádově stokrát nižší, než u původního dotazu.

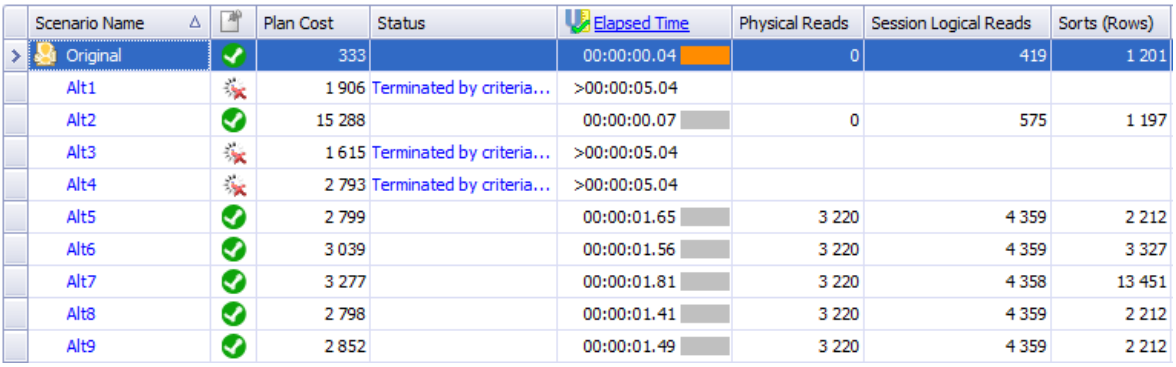
K dalšímu podstatnému zlepšení dochází při aplikaci agregační funkce COUNT na vnořený dotaz. V tomto případě je využit pouze nově vytvořený index COM1_IDX, který je prohledán metodou *fast full scan*.



Obrázek 20 – Exekuční plán dotazu 1.3 B

Automatické generování alternativních exekučních plánů dotazu č. 1.3

Pomocí programu *Quest SQL Optimizer* byly automaticky generovány alternativní exekuční plány s cílem dalšího možného zlepšení výkonu dotazu. V tomto případě se však nepodařilo nalézt alternativu, která by dosahovala lepšího výkonu než dotaz 1.3 B. Tabulka s výsledky testování alternativních exekučních plánů je uvedena na následujícím obrázku 21.



Scenario Name	Plan Cost	Status	Elapsed Time	Physical Reads	Session Logical Reads	Sorts (Rows)
Original	333		00:00:00.04	0	419	1 201
Alt1	1 906	Terminated by criteria...	>00:00:05.04			
Alt2	15 288		00:00:00.07	0	575	1 197
Alt3	1 615	Terminated by criteria...	>00:00:05.04			
Alt4	2 793	Terminated by criteria...	>00:00:05.04			
Alt5	2 799		00:00:01.65	3 220	4 359	2 212
Alt6	3 039		00:00:01.56	3 220	4 359	3 327
Alt7	3 277		00:00:01.81	3 220	4 358	13 451
Alt8	2 798		00:00:01.41	3 220	4 359	2 212
Alt9	2 852		00:00:01.49	3 220	4 359	2 212

Obrázek 21 – Statistika testování alternativních exekučních plánů dotazu 1.3

Z obrázku vyplývá, že nejrychlejším dotazem a stejně tak dotazem s nejnižší cenou je upravený dotaz 3B (na obrázku označen jako Original). Alternativní plány, které mají ve sloupci status uvedenu informaci *Terminated by criteria*, jsou takové plány, jejichž doba provádění značně přesahovala dobu potřebnou pro provedení originálního dotazu. Testování těchto plánů bylo systémem předčasně ukončeno.

Vyhodnocení optimalizace dotazu č. 1.3

Po rozboru původního exekučního plánu byly identifikovány příčiny, které způsobovaly prodlužování prováděcí doby SQL dotazu. Vhodným řešením se ukázalo vytvoření dalšího indexu nad sloupci obsaženými v podmínce where. Díky využití indexu došlo ke značnému zlepšení prováděcího času SQL dotazu. Konkrétní naměřené časy jsou uvedeny v následující tabulce. Kromě zrychlení dotazu došlo i ke snížení ceny jeho exekučního plánu, což je patrné z předchozích obrázků.

Tabulka 10 – Časová statistika dotazu č. 1.3

		Čas provádění [s]			
Dotaz		**1.3 A**	**1.3 A - cache**	**1.3 B**	**1.3 B - cache**
Číslo měření	1	2,695	2,315	0,423	0,021
	2	2,713	2,349	0,421	0,025
	3	2,698	2,298	0,435	0,022
	4	2,691	2,376	0,429	0,022
	5	2,701	2,317	0,423	0,024
Průměr [s]		**2,700**	**2,331**	**0,426**	**0,023**

Upravený dotaz byl podroben automatickému testování alternativních exekučních plánů, nicméně se v tomto případě nepodařilo dosáhnout dalšího zlepšení. Přesto je výsledný optimalizovaný čas velmi dobrý.

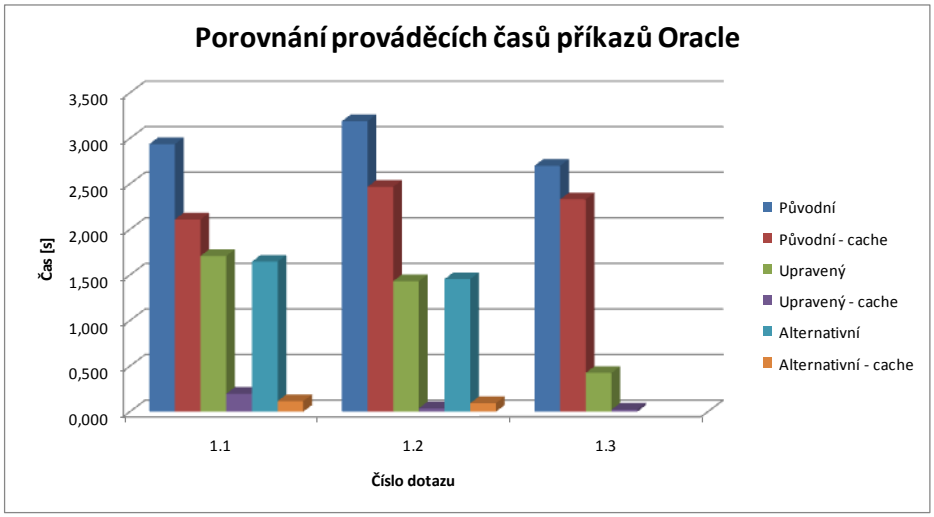
6.3.4 Shrnutí optimalizace databázových dotazů na platformě Oracle

Chování databázového serveru po naplnění větším množstvím dat bylo stabilní. V některých případech sice došlo k prodloužení časů nutných pro provádění jednotlivých dotazů, ale použitelnost aplikace byla i bez optimalizace poměrně dobrá. Po prostudování exekučních plánů byly odhaleny příčiny, které způsobovaly dlouhé prováděcí časy dotazů. Tyto příčiny se podařilo odstranit díky zjednodušení SQL kódů a díky využití nově přidaných indexů.

Další částí optimalizace bylo vygenerování alternativních exekučních plánů pomocí nástroje *Quest SQL Optimalizer*. Vzhledem k velkému množství alternativních plánu bylo použití automatizovaného programu velkým přínosem. V jednom případě se podařilo najít takový alternativní plán, který díky využití vhodného hintu dále vylepšil prováděcí čas. Níže je uvedena souhrnná statistika testovaných databázových dotazů spolu s přehledným grafem.

Tabulka 11 – Průměrné časy provádění SQL dotazů pro server Oracle

Číslo dotazu	Původní [s]	Původní - cache [s]	Upravený [s]	Upravený - cache [s]	Alternativní [s]	Alternativní - cache [s]
1.1	2,936	2,109	1,709	0,197	1,646	0,116
1.2	3,188	2,467	1,432	0,037	1,455	0,094
1.3	2,700	2,331	0,426	0,023		



Obrázek 22 – Graf rychlostí původních a optimalizovaných dotazů pro server Oracle

6.4 Rozbory exekučních plánů MySQL

Po provedení testovacích scénářů byly vybrány příslušné SQL dotazy se značně vysokou dobou provádění. Jedná se o následující dotazy:

1. **Zobrazení seznamu produktů**. Dotaz je dále uváděn, jako dotaz č. 2.1.
2. **Zobrazení seznamu nabídek** včetně kalkulace celkových cen obsažených produktů, zisků a nákladů. Dotaz je dále uváděn, jako dotaz č. 2.2.
3. **Výpis komunikace.** Dotaz je dále uváděn, jako dotaz č. 2.3.
4. **Zobrazení seznamu firem.** Dotaz je dále uváděn, jako dotaz č. 2.4.
5. **Zobrazení seznamu osob.** Dotaz je dále uváděn, jako dotaz č. 2.5.

SQL kódy jednotlivých příkazů jsou uvedeny na přiloženém CD, které je součástí této práce. Pro databázi MySQL se nepodařilo najít vhodný program, který by prováděl automatické generování alternativních exekučních plánů a byl dostupný v nekomerční licenci. K rozboru exekučních plánů byl tedy zvolen program *dbForge Studio for MySQL*[12], který disponuje funkcemi pro přehledné zobrazení exekučních plánů v grafické podobě.

Statistika rychlosti pro každý databázový dotaz obsahuje následující sady měřených časů:

- **Čas provádění dotazu při využití paměti Query cache** – v tomto případě byly časy měřeny za běžného provozu aplikace. Databázový server tedy mohl využívat data uložená v paměti cache a v pomocných bufferech (*Table cache*). Server rovněž mohl využívat předpřipravené výsledky, uložené v pomocné paměti *Query cache*[13]. V tabulkách a grafech, které zaznamenávají hodnoty těchto měření, jsou příslušné sloupce uváděny s identifikátorem ve tvaru *číslo dotazu – Query cache*.
- **Čas provádění dotazu při využití paměti Table cache** – v tomto případě byly opět časy měřeny za běžného provozu aplikace. Databázový server tedy mohl využívat data uložená v paměti cache a v pomocných bufferech. Před každým měřením však bylo provedeno vyprázdnění paměti Query cache tak, aby byl eliminován její vliv na rychlost výsledků. Vyprázdnění této paměti je možné provést pomocí příkazu RESET QUERY CACHE. V tabulkách a grafech, které zaznamenávají hodnoty těchto

[12] Zkušební verze programu je dostupná z www.devart.com/dbforge/mysql.
[13] V této paměti jsou uloženy kódy SQL dotazů spolu s odpovídajícími výsledky. Pokud databázový server obdrží požadavek na vykonání dotazu, jehož výsledky má již v paměti uloženy, budou tato data navrácena přímo z paměti cache.

měření, jsou příslušné sloupce uváděny s identifikátorem ve tvaru *číslo dotazu – Table cache*.

- **Čas provádění dotazu bez využití paměti cache** – v tomto případě byl vliv paměti cache na rychlost dotazu eliminován jejím vyprázdněním. Tohoto stavu bylo docíleno restartováním databázového serveru před každým měřením. V tabulkách a grafech, které zaznamenávají hodnoty těchto měření, jsou příslušné sloupce uváděny s identifikátorem ve tvaru *číslo dotazu*.

6.4.1 Optimalizace dotazu pro výpis seznamu produktů – dotaz č. 2.1

U databáze MySQL se jednalo o nejproblematičtější dotaz. Přestože tento SQL dotaz není nikterak složitý, jeho provádění zabralo databázi čas v řádu desítek sekund (databázový server Oracle si s totožným dotazem poradil v čase na úrovni jednotek sekund). Díky tomu bylo použití modulu *Produkty* prakticky nemožné. Již při pohledu na prováděcí čas příkazu bylo zjevné, že se s největší pravděpodobností jedná o špatnou konstrukci SQL dotazu. Níže je uveden konkrétní SQL kód dotazu a jeho popis.

```
SELECT (SELECT COUNT(*) FROM  PRODUCTS
LEFT JOIN PRODUCTS_SUPPLIERS ON PRODUCTS_SUPPLIERS.PRODUCTS_ID =
PRODUCTS.ID
LEFT JOIN SUPPLIERS ON PRODUCTS_SUPPLIERS.SUPPLIERS_ID = SUPPLIERS.ID
LEFT JOIN (SELECT * FROM ACTUAL_PRICES WHERE PRODUCTS_TYPES_ID IS NULL)
PRIC ON
(PRODUCTS.ID = PRIC.PRODUCTS_ID AND SUPPLIERS.ID = PRIC.SUPPLIERS_ID)
JOIN CATEGORIES ON PRODUCTS.CATEGORIES_ID=CATEGORIES.ID
LEFT JOIN (SELECT IMG_NAME, PRODUCTS_ID FROM IMAGES WHERE MAIN=1) IM ON
PRODUCTS.ID = IM.PRODUCTS_ID WHERE PRODUCTS.VISIBLE=1) as TC,
PRODUCTS.ID as ID, EAN, ORDER_NUM, PRODUCTS.NAME as NAME, CATEGORIES.NAME
as CNAME, SUPPLIERS.NAME as SUPNAME, SELL_PRICE,
PRODUCTS.VISIBLE as VISIBLE, IMG_NAME FROM PRODUCTS
LEFT JOIN PRODUCTS_SUPPLIERS ON PRODUCTS_SUPPLIERS.PRODUCTS_ID =
PRODUCTS.ID
LEFT JOIN SUPPLIERS ON PRODUCTS_SUPPLIERS.SUPPLIERS_ID = SUPPLIERS.ID
LEFT JOIN (SELECT * FROM ACTUAL_PRICES WHERE PRODUCTS_TYPES_ID IS NULL)
PRIC ON
(PRODUCTS.ID = PRIC.PRODUCTS_ID AND SUPPLIERS.ID = PRIC.SUPPLIERS_ID)
JOIN CATEGORIES ON PRODUCTS.CATEGORIES_ID=CATEGORIES.ID
LEFT JOIN (SELECT IMG_NAME, PRODUCTS_ID FROM IMAGES WHERE MAIN=1) IM
ON PRODUCTS.ID = IM.PRODUCTS_ID where PRODUCTS.VISIBLE=1
order by PRODUCTS.ID asc LIMIT 20
```

Z uvedeného kódu je patrné, že je pro výpis nutné spojit tabulky PRODUCTS, PRICES, PRODUCTS_SUPPLIERS, SUPPLIERS, CATEGORIES a IMAGES. Vzhledem k tomu, že cizí klíče (kromě klíče categories_id) umožňují vložení hodnoty NULL, je třeba provádět vnější spojení tabulek, aby výpis obsahoval veškeré produkty. Výsledkem spojení je pak řádek produktu s následujícími atributy: id, ean, objednávkové číslo, jméno produktu, jméno

kategorie, název dodavatele, prodejní cena, příznak zobrazení a hlavní obrázek produktu. Pro podrobnou analýzu provádění SQL dotazu byl vygenerován následující exekuční plán.

table	id	select_type	type	possible_keys	key	key_len	ref	rows	Extra
PRODUCTS	1	PRIMARY	ALL	PRODUCTS_C...				11269	Using where; ...
PRODUCTS_SUPPLIERS	1	PRIMARY	ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	Using index
SUPPLIERS	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	
CATEGORIES	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	
<derived6>	1	PRIMARY	ALL					11960	
IMAGES	6	DERIVED	ALL					9574	Using where
<derived5>	1	PRIMARY	ALL					11962	
prices	5	DERIVED	ref	PRICES_VAT_...	PRICES_PRODU...	5		5002	Using where
vat	5	DERIVED	ALL	PRIMARY				3	Using where; ...
CATEGORIES	2	SUBQUERY	index	PRIMARY	CATEGORIES_C...	5		490	Using index
PRODUCTS	2	SUBQUERY	ref	PRODUCTS_C...	PRODUCTS_CA...	4	dealer.CATEG...	12	Using where
PRODUCTS_SUPPLIERS	2	SUBQUERY	ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	Using index
SUPPLIERS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	Using index
<derived4>	2	SUBQUERY	ALL					11960	
IMAGES	4	DERIVED	ALL					9574	Using where
<derived3>	2	SUBQUERY	ALL					11962	
prices	3	DERIVED	ref	PRICES_VAT_...	PRICES_PRODU...	5		5002	Using where
vat	3	DERIVED	ALL	PRIMARY				3	Using where; ...

Obrázek 23 – Exekuční plán dotazu 2.1 A

Z exekučního plánu je patrné, že většina prohledávání tabulek probíhá metodou úplného prohledávání, což v praxi znamená, že jsou prohledávány všechny řádky od začátku až do konce tabulky. Nejnákladnějšími operacemi pak bude propojení s tabulkou IMAGES a propojení s tabulkou PRICES, které obě mají více než 10 000 řádků.

Z hlediska výsledku dotazu je třeba si uvědomit, že potřebujeme získat pouze 20 řádků (respektive n řádků, kde n je dáno konstantou stránkování v aplikaci), přesto je prohledávání a spojování tabulek prováděno nad všemi řádky. Dále však potřebujeme získat celkový počet všech řádků odpovídající podmínce filtru tabulky, pro výpočet počtu stránek. V tomto případě již není možné vyhnout se spojení příslušných tabulek v celém rozsahu, nicméně je možné spojování optimalizovat.

Návrh optimalizovaného kódu SQL

Vzhledem k tomu, že filtr příslušného výpisu v administraci filtruje pouze nad sloupci uvedenými v tabulce PRODUCTS, SUPPLIERS a CATEGORIES, je možné provést omezení počtu řádků výsledku již po spojení těchto tabulek a připojení ostatních tabulek provést až k výsledné množině výstupních řádků. Díky této úpravě klesne počet původně spojovaných řádků na pouhých 20, což se projeví značným zrychlením tohoto dotazu.

```
SELECT (SELECT COUNT(*) FROM
PRODUCTS
LEFT JOIN  PRODUCTS_SUPPLIERS ON PRODUCTS_SUPPLIERS.PRODUCTS_ID =
PRODUCTS.ID
LEFT JOIN SUPPLIERS ON PRODUCTS_SUPPLIERS.SUPPLIERS_ID = SUPPLIERS.ID
LEFT JOIN CATEGORIES ON PRODUCTS.CATEGORIES_ID=CATEGORIES.ID WHERE
PRODUCTS.VISIBLE=1  ) AS TC,
ID, EAN, ORDER_NUM,  NAME,  CNAME, SUPNAME, SELL_PRICE, VISIBLE, IMG_NAME
FROM(
SELECT
PRODUCTS.ID AS ID, EAN, SUPPLIERS.ID AS SUPID, ORDER_NUM, PRODUCTS.NAME AS
NAME,
CATEGORIES.NAME AS CNAME, PRODUCTS.CATEGORIES_ID AS CAT_ID,
SUPPLIERS.NAME AS SUPNAME, PRODUCTS.VISIBLE AS VISIBLE
FROM  PRODUCTS
LEFT JOIN  PRODUCTS_SUPPLIERS ON PRODUCTS_SUPPLIERS.PRODUCTS_ID =
PRODUCTS.ID
LEFT JOIN SUPPLIERS ON PRODUCTS_SUPPLIERS.SUPPLIERS_ID = SUPPLIERS.ID
LEFT JOIN CATEGORIES ON PRODUCTS.CATEGORIES_ID=CATEGORIES.ID
WHERE  PRODUCTS.VISIBLE=1  LIMIT 0, 20   ) T1
LEFT JOIN (SELECT IMG_NAME, PRODUCTS_ID FROM IMAGES WHERE MAIN=1)T2 ON
T1.ID =T2.PRODUCTS_ID
LEFT JOIN (SELECT * FROM ACTUAL_PRICES WHERE PRODUCTS_TYPES_ID IS NULL)
PRIC ON
(T1.ID = PRIC.PRODUCTS_ID AND T1.SUPID = PRIC.SUPPLIERS_ID) ORDER BY T1.ID
ASC;
```

Na následujícím obrázku je uveden příslušný exekuční plán upraveného dotazu, kde je jasně patrná markantní změna v počtu zpracovávaných řádků.

table	id	select_type	type	possible_keys	key	key_len	ref	rows	Extra
<derived3>	1	PRIMARY	ALL					20	
PRODUCTS	3	DERIVED	index		PRIMARY	4		11269	Using where
PRODUCTS_SUPPLIERS	3	DERIVED	ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	Using index
SUPPLIERS	3	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	
CATEGORIES	3	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	
<derived4>	1	PRIMARY	ALL					11960	
IMAGES	4	DERIVED	ALL					9574	Using where
<derived5>	1	PRIMARY	ALL					11962	
prices	5	DERIVED	ref	PRICES_VA...	PRICES_PRO...	5		5002	Using where
vat	5	DERIVED	ALL	PRIMARY				3	Using where; Using j...
PRODUCTS	2	SUBQUERY	ALL					11269	Using where
PRODUCTS_SUPPLIERS	2	SUBQUERY	ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	Using index
SUPPLIERS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	Using index
CATEGORIES	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.PROD...	1	Using index

Obrázek 24 – Exekuční plán dotazu 2.1 B

Vyhodnocení optimalizace dotazu č. 2.1

V tomto případě je patrné, že původní dotaz byl chybně navržen. Je to důsledek toho, že vytvořený systém využívá metody pro univerzální automatizovaný výpis tabulkových hodnot, který nemusí být pro všechny druhy výpisů optimální. K odhalení tohoto problému mohlo dojít až po naplnění systému větším množstvím dat, jelikož nákladnost operací není

možné na malém počtu testovacích dat prakticky zaznamenat (z hlediska rychlosti práce s aplikací).

Po vyhodnocení exekučního plánu a odhalení příčin zdlouhavého provádění dotazu bylo aplikováno pravidlo týkající se omezování počtu řádků, které budou dále spojovány a také pravidlo týkající se správného pořadí spojovaných tabulek. Níže je uvedena statistika doby provádění původního a upraveného kódu dotazu.

Tabulka 12 – Časová statistika dotazu č. 2.1

		Čas provádění [s]				
Dotaz		2.1 A	2.1 A - Table cache	2.1 B	2.1 B - Table cache	2.1 B - Query cache
Číslo měření	1	155,136	147,263	1,923	0,315	0,293
	2	154,845	145,398	1,958	0,311	0,295
	3	156,268	147,259	1,935	0,315	0,293
	4	155,698	149,035	1,899	0,317	0,299
	5	155,978	146,597	1,942	0,313	0,303
Průměr [s]		**155,585**	**147,110**	**1,931**	**0,314**	**0,297**

Na tomto výsledku je jednoznačně patrné, že chybný návrh databázového dotazu může prakticky znemožnit použití aplikace. Proto je třeba věnovat čas analýze příčin nízkého výkonu a jejich následnému odstranění.

6.4.2 Optimalizace dotazu pro zobrazení seznamu nabídek – dotaz č. 2.2

V tomto případě se, vzhledem ke složité struktuře databázového dotazu, jednalo o poměrně rychlý SQL dotaz, jehož doba provádění byla menší než 1 vteřina. Tento dotaz je však využíván i při načítání úvodní stránky v kombinaci s dalšími složitějšími dotazy. Jakákoli optimalizace z hlediska prováděcího času bude tedy vést jak ke zrychlení načítání modulu nabídek, tak i ke zrychlení načítání úvodní stránky.

SQL kód dotazu zde není uveden kvůli jeho velkému rozsahu. Tento dotaz je uveden na přiloženém CD. Výsledkem dotazu je seznam nabídek spolu s výpočtem celkové nákupní ceny všech produktů příslušné nabídky, celkové prodejní ceny a celkové marže pro každou z vypisovaných nabídek. Databázový dotaz je tvořen spojením jedenácti tabulek, tudíž nároky na spojování jednotlivých tabulek při vysokém počtu řádků mohou být velmi velké.

Z následujícího exekučního plánu je patrné, že do spojení vstupuje značné množství tabulek. Přestože výpis obsahuje pouze 20 záznamů, spojuje databázový server 784 řádků tabulky COMPANY, což je počet řádků, které vyhovují omezení klauzulí where. Až následně je připojena tabulka OFFERS.

table	id	select_type	type	possible_keys	key	key_len	ref	rows	Extra
<derived5>	1	PRIMARY	ALL					110	Using fil...
<derived6>	5	DERIVED	ALL					1321	Using te...
PRICE_LEVEL	6	DERIVED	ALL	PRIMARY				1	
COMPANY	6	DERIVED	ref	PRIMARY,COMPA...	COMPANY_P...	4	dealer.PRICE_LEVEL.ID	784	Using w...
OFFERS	6	DERIVED	ref	OFFERS_COMPAN...	OFFERS_CO...	4	dealer.COMPANY.ID	2	
USERS	6	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS.USE...	1	Using w...
OFFERS_PRODUCTS	6	DERIVED	ref	OFFERS_PRODUC...	OFFERS_PRO...	4	dealer.OFFERS.ID	8	Using w...
PRODUCTS	6	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS_PRO...	1	
PRODUCTS_TYPES	6	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS_PRO...	1	Using in...
OFFER_STATES	6	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS.OFF...	1	
SUPPLIERS	6	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS_PRO...	1	Using in...
UNITS	6	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.PRODUCTS.U...	1	Using in...
PRICES	6	DERIVED	ref	PRIMARY,PRICES...	PRICES_PRO...	4	dealer.OFFERS_PRO...	1	Using w...
VAT	6	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.PRICES.VAT_ID	1	
	2	SUBQUERY							Select ta...
<derived4>	3	DERIVED	ALL					1321	Using te...
PRICE_LEVEL	4	DERIVED	ALL	PRIMARY				1	
COMPANY	4	DERIVED	ref	PRIMARY,COMPA...	COMPANY_P...	4	dealer.PRICE_LEVEL.ID	784	Using w...
OFFERS	4	DERIVED	ref	OFFERS_COMPAN...	OFFERS_CO...	4	dealer.COMPANY.ID	2	
USERS	4	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS.USE...	1	Using w...
OFFERS_PRODUCTS	4	DERIVED	ref	OFFERS_PRODUC...	OFFERS_PRO...	4	dealer.OFFERS.ID	8	Using w...
PRODUCTS	4	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS_PRO...	1	
PRODUCTS_TYPES	4	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS_PRO...	1	Using in...
OFFER_STATES	4	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS.OFF...	1	
SUPPLIERS	4	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS_PRO...	1	Using in...
UNITS	4	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.PRODUCTS.U...	1	Using in...
PRICES	4	DERIVED	ref	PRIMARY,PRICES...	PRICES_PRO...	4	dealer.OFFERS_PRO...	1	Using w...
VAT	4	DERIVED	eq_ref	PRIMARY	PRIMARY	4	dealer.PRICES.VAT_ID	1	

Obrázek 25 – Exekuční plán dotazu 2.2 A

Návrh optimalizovaného kódu SQL

Díky skutečnosti, že je seznam nabídek vypisován vždy pro konkrétního uživatele (s výjimkou administrátorského přístupu), je vhodné provést omezení tabulky OFFERS pouze na řádky s konkrétním uživatelským id ještě před vstupem do operace spojení a zajistit, aby do operace spojení vstupovala jako první množina vyhovujících řádků z této tabulky. Toho je možné dosáhnout pomocí vnořeného dotazu, který bude obsahovat všechny podmínky vztahující se na výše uvedenou tabulku. Na následujícím obrázku je uveden exekuční plán upraveného dotazu, ze kterého je patrné, že došlo k potřebnému omezení řádků, které vstupují do operace spojení tabulek.

table	id	sele...	type	possible_keys	key	key_len	ref	rows	Extra
<derived3>	1	PRI...	ALL					110	Using where; U...
<derived4>	3	DER...	ALL					1321	Using temporar...
PRICE_LEVEL	4	DER...	ALL	PRIMARY				1	
<derived5>	4	DER...	ALL					110	Using join buffer
OFFERS	5	DER...	ref	OFFERS_USERS_FK	OFFERS_USERS_FK	4		110	
OFFER_STATES	4	DER...	eq_ref	PRIMARY	PRIMARY	4	T5.OFFER_STATES_ID	1	
USERS	4	DER...	eq_ref	PRIMARY	PRIMARY	4	T5.USERS_ID	1	
COMPANY	4	DER...	eq_ref	PRIMARY,COMPANY_PR...	PRIMARY	4	T5.COMPANY_ID	1	Using where
OFFERS_PRODUCTS	4	DER...	ref	OFFERS_PRODUCTS_O...	OFFERS_PRODU...	4	T5.ID	8	Using where
PRODUCTS	4	DER...	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS_PROD...	1	
PRODUCTS_TYPES	4	DER...	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS_PROD...	1	Using index
SUPPLIERS	4	DER...	eq_ref	PRIMARY	PRIMARY	4	dealer.OFFERS_PROD...	1	Using index
UNITS	4	DER...	eq_ref	PRIMARY	PRIMARY	4	dealer.PRODUCTS.UNI...	1	Using index
PRICES	4	DER...	ref	PRIMARY,PRICES_PRO...	PRICES_PRODUC...	4	dealer.OFFERS_PROD...	1	Using where
VAT	4	DER...	eq_ref	PRIMARY	PRIMARY	4	dealer.PRICES.VAT_ID	1	
OFFERS	2	SUB...	ref	OFFERS_USERS_FK	OFFERS_USERS_FK	4	const	110	Using index

Obrázek 26 – Exekuční plán dotazu 2.2 B

Vyhodnocení optimalizace dotazu č. 2.2

I v tomto případě se podařilo dosáhnout vylepšení prováděcího času SQL příkazu především díky správnému pořadí tabulek při spojování. Další vliv mělo maximální omezení tabulky, která do operace spojení vstupuje jako první, pomocí klauzule where. Další časové úspory se podařilo dosáhnout tím, že z vnořeného dotazu, který vrací celkový počet řádků výpisu, bylo vypuštěno spojení tabulek, které nebylo nutné zahrnout do agregační funkce count. Při optimalizaci bylo testováno i využití obdobného indexu jako v případě databáze Oracle, nicméně doplnění indexu nepřineslo žádné zvýšení výkonu dotazu. Níže je uvedena tabulka se statistikou doby provádění původního a upraveného dotazu.

Tabulka 13 – Časová statistika dotazu č. 2.2

		Čas provádění [s]				
Dotaz		2.2 A	2.2A - Table cache	2.2 B	2.2 B - Table cache	2.2 B - Query cache
Číslo měření	1	64,101	2,561	6,858	0,175	0,162
	2	62,523	2,565	5,957	0,171	0,167
	3	63,986	2,608	6,452	0,174	0,181
	4	64,125	2,598	5,815	0,182	0,173
	5	64,589	2,661	6,455	0,181	0,167
Průměr [s]		**63,865**	**2,599**	**6,307**	**0,177**	**0,170**

6.4.3 Optimalizace dotazu pro zobrazení seznamu komunikace – dotaz č. 2.3

Výpis seznamu komunikace byl další problematický dotaz, jehož doba provádění byla nepřijatelná, přestože se jednalo o dotaz poměrně jednoduché konstrukce. Vzhledem k tomu, že provádění dotazu trvalo řádově desítky vteřin, opět se dalo předpokládat, že se jedná o chybně navržený SQL dotaz.

```
SELECT
(SELECT COUNT(*) FROM  COMMUNICATION
LEFT JOIN USERS ON COMMUNICATION.USERS_ID=USERS.ID
JOIN OFFERS ON OFFERS_ID=OFFERS.ID WHERE (USERS.ID=10 OR USERS.ID IS NULL))
AS TC,
COMMUNICATION.ID AS ID, OFFERS_ID, USERS.SURNAME AS SURNAME, OFFERS.NAME AS
ONAME, MESSAGE,
ANSWER, COMMUNICATION.CREATED AS COMCREATED FROM COMMUNICATION LEFT JOIN
USERS ON COMMUNICATION.USERS_ID=USERS.ID
JOIN OFFERS ON OFFERS_ID=OFFERS.ID WHERE (USERS.ID=10 OR USERS.ID IS NULL)
ORDER BY COMMUNICATION.ID ASC LIMIT 20;
```

Z uvedeného kódu je patrné, že je pro výpis nutné spojit tabulky COMMUNICATION, USERS a OFFERS. Vzhledem k tomu, že sloupec cizího klíče USERS_ID může obsahovat hodnoty NULL, je třeba toto spojení realizovat jako vnější spojení. Výsledkem spojení je pak získání jednotlivých řádků tabulky komunikace, doplněných o jméno uživatele a název

připojené obchodní nabídky. Pro podrobnější informace byl vygenerován následující exekuční plán příkazu uvedený na obrázku 27.

table	id	select_type	type	possible_keys	key	k...	ref	rows	Extra
OFFERS	1	PRIMARY	ALL	PRIMARY				12808	Using tempor...
COMMUNICATION	1	PRIMARY	ref	COMMUNICATION...	COMMUNICATION_O...	4	dealer.OFFER...	4	
USERS	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMM...	1	Using where
OFFERS	2	SUBQUERY	index	PRIMARY	OFFERS_COMPANY_FK	4		12808	Using index
COMMUNICATION	2	SUBQUERY	ref	COMMUNICATION...	COMMUNICATION_O...	4	dealer.OFFER...	4	
USERS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMM...	1	Using where; ...

Obrázek 27 – Exekuční plán dotazu 2.3 A

Z exekučního plánu je patrné, že dochází ke kompletnímu načtení a procházení tabulky OFFERS přestože tato tabulka není pro výpis potřeba celá. Dále je z plánu vidět, že k omezení výsledků podle ID uživatele dochází až po načtení tabulky OFFERS.

Návrh optimalizovaného kódu SQL

Vzhledem k tomu, že cílem dotazu je získat pouze komunikaci, která přísluší danému uživateli, bylo by výhodné omezit již načítání řádků tabulek OFFERS a COMMUNICATION pouze na řádky obsahující ve sloupci USERS_ID příslušnou hodnotu identifikátoru. Toho je možné dosáhnout prostou změnou názvu sloupců v omezující podmínce tak, aby se restrikce vztahovala na největší tabulku ve výběru. Níže je uveden upravený SQL kód příkazu č. 2.3.

```
SELECT
(select count(*) FROM  COMMUNICATION LEFT JOIN USERS ON
COMMUNICATION.USERS_ID=USERS.ID
JOIN OFFERS ON OFFERS_ID=OFFERS.ID where( COMMUNICATION.USERS_ID=10 or
COMMUNICATION.USERS_ID is NULL )  ) as TC,
COMMUNICATION.ID as ID, OFFERS_ID, USERS.SURNAME as SURNAME, OFFERS.NAME as
ONAME, MESSAGE,
ANSWER, COMMUNICATION.CREATED as COMCREATED FROM COMMUNICATION LEFT JOIN
USERS ON COMMUNICATION.USERS_ID=USERS.ID
JOIN OFFERS ON OFFERS_ID=OFFERS.ID where ( COMMUNICATION.USERS_ID=10 or
COMMUNICATION.USERS_ID is NULL )  order by COMMUNICATION.ID asc LIMIT 20;
```

Na následujícím obrázku 28 je uveden příslušný exekuční plán upraveného dotazu, kde je jasně patrná markantní změna v počtu zpracovávaných řádku, ke které došlo díky omezení na úrovni největší tabulky.

table	id	select_type	type	possible_keys	key	key_len	ref	rows	Extra
COMMUNICATION	1	PRIMARY	ref	COMMUNICATIO...	COMMUNICATION_USERS_FK	4	const	1014	Using wh...
USERS	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMMUNIC...	1	
OFFERS	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMMUNIC...	1	
COMMUNICATION	2	SUBQUERY	ref	COMMUNICATIO...	COMMUNICATION_USERS_FK	4	const	1014	
USERS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMMUNIC...	1	Using index
OFFERS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMMUNIC...	1	Using index

Obrázek 28 – Exekuční plán dotazu 2.3 B

Vyhodnocení optimalizace dotazu č. 2.3

V tomto případě byla u dotazu nesprávně zvolena omezující podmínka v klauzuli where. Přestože se jednalo o totožné hodnoty sloupců cizího klíče, optimalizátor nedokázal vhodně aplikovat tuto podmínku na nejvhodnější tabulku. Po specifikaci klauzule where tak, aby byla restriktivní podmínka aplikována na tabulku COMMUNICATION, byl počet zpracovávaných záznamů značně omezen. Díky tomu se snížil čas potřebný pro vykonávání na zlomek původní hodnoty. V tomto případě byla opět aplikována obecná poučka o uspořádání klauzule where v takovém pořadí, aby bylo restrikcí z výsledné množiny odfiltrováno co největší množství řádků. Níže je uvedena tabulka se statistikou prováděcích časů příkazu.

Tabulka 14 – Časová statistika dotazu č. 2.3

		Čas provádění [s]				
Dotaz		**2.3 A**	**2.3 A - Table cache**	**2.3 B**	**2.3 B - Table cache**	**2.3 B - Query cache**
Číslo měření	1	107,014	86,619	4,606	0,140	0,011
	2	107,158	86,167	4,387	0,120	0,009
	3	107,297	87,305	4,398	0,110	0,010
	4	106,957	87,133	4,701	0,120	0,011
	5	107,688	86,617	4,699	0,140	0,013
Průměr [s]		**107,223**	**86,768**	**4,558**	**0,126**	**0,011**

U tohoto dotazu bylo opět otestováno využití obdobného indexu, jako v případě databáze Oracle, nicméně využití indexu nebylo z hlediska prováděcího času přínosné.

6.4.4 Optimalizace dotazu pro zobrazení seznamu firem – dotaz č. 2.4

Dotaz, který zajišťuje výpis seznamu firem, nepatří svojí konstrukcí mezi nejsložitější dotazy, nicméně pro jeho provedení je nutné spojit značné množství poměrně rozsáhlých tabulek. Z tohoto důvodu se doba vykonání dotazu pohybovala v řádu jednotek vteřin a byl zde tedy prostor pro optimalizaci. Na následující stránce je uveden SQL kód příslušného dotazu.

```
SELECT (SELECT COUNT(*) FROM  COMPANY
LEFT JOIN COMP_TYPE ON COMP_TYPE.ID=COMP_TYPE_ID
JOIN CUST_STATES ON CUST_STATES.ID=CUST_STATES_ID
JOIN PRICE_LEVEL ON PRICE_LEVEL.ID=PRICE_LEVEL_ID
LEFT JOIN (SELECT STREET, CITY,COMPANY_ID, ADDRESS_ID, ADDR_TYPES_ID  FROM
COMPANY_ADDR
LEFT JOIN ADDRESS ON ADDRESS_ID=ADDRESS.ID WHERE MAIN=1 AND
ADDR_TYPES_ID=1) T1 ON COMPANY_ID=COMPANY.ID
LEFT JOIN USERS ON USERS.ID=USERS_ID
WHERE
COMPANY.ACTIVE=1  AND  ( USERS.ID=10 OR USERS.ID IS NULL )  ) AS TC,
COMPANY.ID AS ID, COMPANY.NAME AS NAME, WEB, STREET, CITY, NO,
CUST_STATES.NAME AS CSTATE, PRICE_LEVEL.NAME AS PLEVEL, USERS.SURNAME AS
SURNAME, COMPANY.ACTIVE AS ACTIVE, DATE_FORMAT(COMPANY.CREATED, '%D.%M.%Y
%H:%I:%S' ) AS CREATED
FROM  COMPANY
LEFT JOIN COMP_TYPE ON COMP_TYPE.ID=COMP_TYPE_ID
JOIN CUST_STATES ON CUST_STATES.ID=CUST_STATES_ID
JOIN PRICE_LEVEL ON PRICE_LEVEL.ID=PRICE_LEVEL_ID
LEFT JOIN (SELECT STREET, CITY,COMPANY_ID, ADDRESS_ID, ADDR_TYPES_ID  FROM
COMPANY_ADDR
LEFT JOIN ADDRESS ON ADDRESS_ID=ADDRESS.ID WHERE MAIN=1 AND
ADDR_TYPES_ID=1) T1 ON COMPANY_ID=COMPANY.ID
LEFT JOIN USERS ON USERS.ID=USERS_ID
WHERE
COMPANY.ACTIVE=1  AND  ( USERS.ID=10 OR USERS.ID IS NULL )  ORDER BY
COMPANY.ID ASC       LIMIT 20;
```

Z uvedeného kódu je patrné, že je pro výpis nutné spojit celkem osm tabulek, přičemž především tabulky COMPANY, COMPANY_ADDR, ADDRESS mají řádově tisíce řádků. Výsledkem databázového dotazu je seznam firem spolu s jejich hlavní adresou a informacemi o cenové hladině příslušné firmy a obchodníkovi, který konkrétní firmu do systému přidal. Pro podrobnější informace byl vygenerován následující exekuční plán příkazu.

table	id	select_type	type	possible_keys	key	key_len	ref	rows	Extra
PRICE_LEVEL	1	PRIMARY	ALL	PRIMARY				1	Using temporary; Using filesort
CUST_STATES	1	PRIMARY	ALL	PRIMARY				3	Using join buffer
COMPANY	1	PRIMARY	ref	COMPANY_CUSTOMME...	COMPANY...	4	dealer.CUST_STA...	462	Using where
COMP_TYPE	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMPANY....	1	Using index
<derived4>	1	PRIMARY	ALL					2441	
ADDRESS	4	DERIVED	ref	PRIMARY,COMPANY_A...	COMPANY...	4		2442	Using where
COMPANY_ADDR	4	DERIVED	ref	PRIMARY	PRIMARY	4	dealer.ADDRESS.ID	1	Using where
USERS	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMPANY....	1	Using where
PRICE_LEVEL	2	SUBQUERY	index	PRIMARY	PRIMARY	4		1	Using index
CUST_STATES	2	SUBQUERY	index	PRIMARY	PRIMARY	4		3	Using index; Using join buffer
COMPANY	2	SUBQUERY	ref	COMPANY_CUSTOMME...	COMPANY...	4	dealer.CUST_STA...	462	Using where
COMP_TYPE	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMPANY....	1	Using index
<derived3>	2	SUBQUERY	ALL					2441	
ADDRESS	3	DERIVED	ref	PRIMARY,COMPANY_A...	COMPANY...	4		2442	Using where
COMPANY_ADDR	3	DERIVED	ref	PRIMARY	PRIMARY	4	dealer.ADDRESS.ID	1	Using where
USERS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMPANY....	1	Using where; Using index

Obrázek 29 – Exekuční plán dotazu 2.4 A

Z exekučního plánu je patrné, že dochází k načtení 462 záznamů tabulky COMPANY, ke kterým jsou následně připojovány adresy omezené podmínkou where. Vzhledem k tomu, že zmiňovaná podmínka where není příliš silná, je počet spojovaných řádků značný. Ostatní operace tvořící SQL příkaz již nemají příliš velký vliv na dobu jeho provádění, tudíž optimalizace by měla být zaměřena především na výběr záznamů z tabulky COMPANY a jejich spojení s tabulkou ADDRESS.

Návrh optimalizovaného kódu SQL

Jak již bylo uvedeno v předchozí části, vstupuje do operace spojení 462 záznamů firem, což je zapříčiněno uvedením silné podmínky where až po provedení spojení všech tabulek. Vhodnějším řešením tedy bude doplnit do dotazu další vnořený dotaz (označen aliasem T5), který zajistí restrikci řádků z tabulky COMPANY ještě před vstupem do operace spojení. Níže je uveden upravený SQL kód příkazu č. 2.4.

```
SELECT (SELECT COUNT(*) FROM  (SELECT * FROM COMPANY  WHERE ( USERS_ID=10
OR USERS_ID IS NULL )) T5
LEFT JOIN (SELECT STREET, CITY,COMPANY_ID, ADDRESS_ID, ADDR_TYPES_ID  FROM
COMPANY_ADDR
LEFT JOIN ADDRESS ON ADDRESS_ID=ADDRESS.ID WHERE ADDR_TYPES_ID=1 AND
MAIN=1) T1 ON COMPANY_ID=T5.ID
LEFT JOIN COMP_TYPE ON COMP_TYPE.ID=COMP_TYPE_ID
JOIN CUST_STATES ON CUST_STATES.ID=CUST_STATES_ID
JOIN PRICE_LEVEL ON PRICE_LEVEL.ID=PRICE_LEVEL_ID
LEFT JOIN USERS ON USERS.ID=T5.USERS_ID
WHERE T5.ACTIVE=1  AND  ( T5.USERS_ID=10 OR T5.USERS_ID IS NULL )  ) AS TC,
T5.ID AS ID, T5.ACTIVE, DATE_FORMAT(T5.CREATED, '%D.%M.%Y %H:%I:%S' ) AS
CREATED
FROM (SELECT * FROM COMPANY WHERE (USERS_ID=10 OR USERS_ID IS NULL )) T5
LEFT JOIN (SELECT STREET, CITY,COMPANY_ID, ADDRESS_ID, ADDR_TYPES_ID  FROM
COMPANY_ADDR
LEFT JOIN ADDRESS ON ADDRESS_ID=ADDRESS.ID WHERE ADDR_TYPES_ID=1 AND
MAIN=1) T1 ON COMPANY_ID=T5.ID
LEFT JOIN COMP_TYPE ON COMP_TYPE.ID=COMP_TYPE_ID
JOIN CUST_STATES ON CUST_STATES.ID=CUST_STATES_ID
JOIN PRICE_LEVEL ON PRICE_LEVEL.ID=PRICE_LEVEL_ID
LEFT JOIN USERS ON USERS.ID=T5.USERS_ID
WHERE  T5.ACTIVE=1  AND  ( T5.USERS_ID=10 OR T5.USERS_ID IS NULL )
ORDER BY T5.ID ASC       LIMIT 20;
```

Na následujícím obrázku 30 je uveden příslušný exekuční plán upraveného dotazu. Z exekučního plánu je patrné, že došlo k významnému omezení počtu řádků tabulky COMPANY, pro které je prováděno úplné prohledávání tabulky ADDRESS, respektive množiny řádků vzniklých spojením tabulek ADDRESS a COMPANY_ADDR. Jelikož je operace úplného prohledávání tabulky poměrně nákladná, snížení počtu kombinací, pro které je tato operace prováděna, vede ke značné úspoře času při vykonávání.

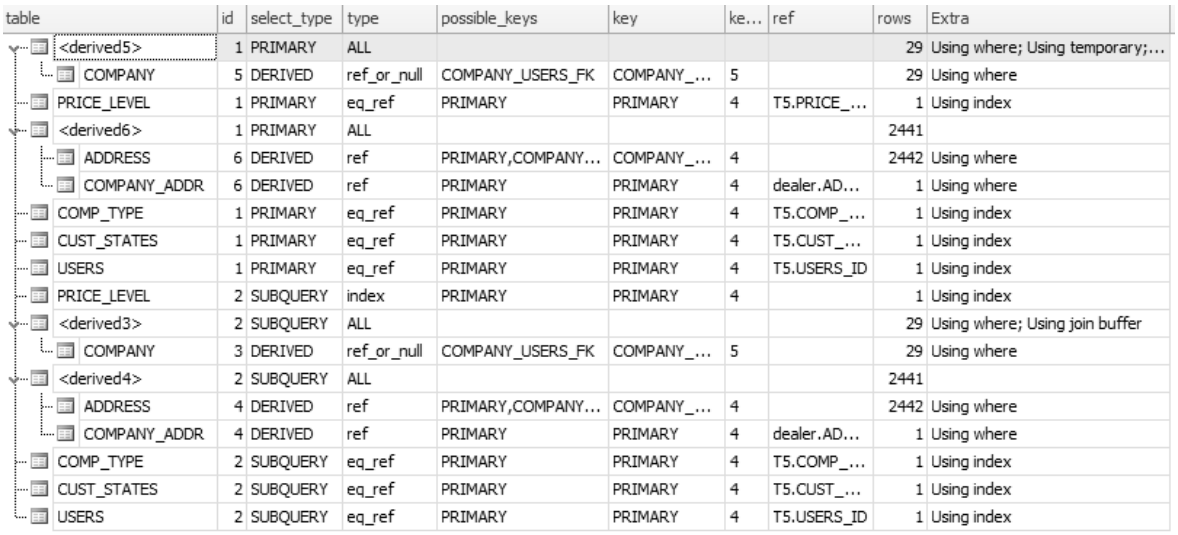


table	id	select_type	type	possible_keys	key	ke...	ref	rows	Extra
<derived5>	1	PRIMARY	ALL					29	Using where; Using temporary;...
COMPANY	5	DERIVED	ref_or_null	COMPANY_USERS_FK	COMPANY_...	5		29	Using where
PRICE_LEVEL	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	T5.PRICE_...	1	Using index
<derived6>	1	PRIMARY	ALL					2441	
ADDRESS	6	DERIVED	ref	PRIMARY,COMPANY...	COMPANY_...	4		2442	Using where
COMPANY_ADDR	6	DERIVED	ref	PRIMARY	PRIMARY	4	dealer.AD...	1	Using where
COMP_TYPE	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	T5.COMP_...	1	Using index
CUST_STATES	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	T5.CUST_...	1	Using index
USERS	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	T5.USERS_ID	1	Using index
PRICE_LEVEL	2	SUBQUERY	index	PRIMARY	PRIMARY	4		1	Using index
<derived3>	2	SUBQUERY	ALL					29	Using where; Using join buffer
COMPANY	3	DERIVED	ref_or_null	COMPANY_USERS_FK	COMPANY_...	5		29	Using where
<derived4>	2	SUBQUERY	ALL					2441	
ADDRESS	4	DERIVED	ref	PRIMARY,COMPANY...	COMPANY_...	4		2442	Using where
COMPANY_ADDR	4	DERIVED	ref	PRIMARY	PRIMARY	4	dealer.AD...	1	Using where
COMP_TYPE	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	T5.COMP_...	1	Using index
CUST_STATES	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	T5.CUST_...	1	Using index
USERS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	T5.USERS_ID	1	Using index

Obrázek 30 – Exekuční plán dotazu 2.4 B

Vyhodnocení optimalizace dotazu č. 2.4

U tohoto dotazu se ukázalo, že počet řádků, pro které bylo realizováno úplné prohledávání další tabulky, byl zbytečně velký. Tato situace byla zapříčiněna uvedením podmínky where až po provedení spojení jednotlivých tabulek. Díky využití vnořeného dotazu se silně omezující podmínkou v klauzuli where, došlo k citelnému snížení počtu řádků, pro které bylo nutné realizovat úplné prohledávání další tabulky. Tato skutečnost měla velmi příznivý vliv na zrychlení doby provádění celého dotazu. Statistika časové náročnosti provádění příkazu je uvedena v následující tabulce.

Tabulka 15 – Časová statistika dotazu č. 2.4

		Čas provádění [s]				
Dotaz		**2.4 A**	**2.4 A - Table cache**	**2.4 B**	**2.4 B - Table cache**	**2.4 B - Query cache**
Číslo měření	1	3,163	2,080	0,924	0,074	0,076
	2	3,253	2,072	0,917	0,077	0,072
	3	3,189	2,073	0,919	0,076	0,078
	4	3,201	2,090	0,921	0,081	0,078
	5	3,217	2,118	0,926	0,077	0,073
Průměr [s]		**3,205**	**2,087**	**0,921**	**0,077**	**0,075**

6.4.5 Optimalizace dotazu pro zobrazení seznamu osob – dotaz č. 2.5

Dotaz, který zajišťuje výpis seznamu osob, pracuje s tabulkami, které mají velké počty řádků. Z tohoto důvodu se neoptimální konstrukce dotazu může projevit značným zvýšením časové náročnosti. Dále je uveden původní SQL kód dotazu pro výpis seznamu osob.

```
SELECT (SELECT COUNT(*) FROM  PERSONS
LEFT JOIN COMPANY ON COMPANY_ID=COMPANY.ID
LEFT JOIN (SELECT * FROM PERSONS_CONT WHERE MAIN=1) T2 ON
T2.PERSONS_ID=PERSONS.ID
LEFT JOIN CONTACTS ON CONTACTS_ID=CONTACTS.ID
LEFT JOIN (SELECT * FROM PERSONS_ADDR WHERE MAIN=1) T1 ON
T1.PERSONS_ID=PERSONS.ID
LEFT JOIN ADDRESS ON ADDRESS_ID=ADDRESS.ID
LEFT JOIN USERS ON COMPANY.USERS_ID=USERS.ID
WHERE   ( USERS.ID=10 OR USERS.ID IS NULL )  ) AS TC, PERSONS.ID AS ID,
PERSONS.NAME AS NAME, PERSONS.SURNAME AS SURNAME,
COMPANY.NAME AS COMPNAME, STREET, CITY, CONTACTS.VALUE AS CEMAIL,
PERSONS.MAIN AS MA, DATE_FORMAT(PERSONS.CREATED, '%D.%M.%Y %H:%I:%S' ) AS
CREATED  FROM  PERSONS
LEFT JOIN COMPANY ON COMPANY_ID=COMPANY.ID
LEFT JOIN (SELECT * FROM PERSONS_CONT WHERE MAIN=1) T2 ON
T2.PERSONS_ID=PERSONS.ID
LEFT JOIN CONTACTS ON CONTACTS_ID=CONTACTS.ID
LEFT JOIN (SELECT * FROM PERSONS_ADDR WHERE MAIN=1) T1 ON
T1.PERSONS_ID=PERSONS.ID
LEFT JOIN ADDRESS ON ADDRESS_ID=ADDRESS.ID
LEFT JOIN USERS ON COMPANY.USERS_ID=USERS.ID
WHERE   ( USERS.ID=10 OR USERS.ID IS NULL )  ORDER BY PERSONS.ID ASC
LIMIT 20
```

Z uvedeného kódu je patrné, že obsahuje nákladné vnější spojení s tabulkami COMPANY, CONTACTS a ADDRESS. Dá se tedy očekávat, že tato spojení budou z hlediska časové náročnosti nejvíce prodlužovat vykonání SQL příkazu. Pro podrobnější informace byl vygenerován následující exekuční plán příkazu.

table	id	select_type	type	possible_keys	key	key_len	ref	rows	Extra
PERSONS	1	PRIMARY	ALL					4421	Using temporary; Using filesort
COMPANY	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	dealer.PERSONS.COMPANY_ID	1	
<derived5>	1	PRIMARY	ALL					6487	
PERSONS_CONT	5	DERIVED	ALL					5230	Using where
CONTACTS	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	T2.CONTACTS_ID	1	
<derived6>	1	PRIMARY	ALL					6487	
PERSONS_ADDR	6	DERIVED	ALL					6820	Using where
ADDRESS	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	T1.ADDRESS_ID	1	
USERS	1	PRIMARY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMPANY.USERS_ID	1	Using where; Using index
PERSONS	2	SUBQUERY	index		PERSONS_CO...	5		4421	Using index
COMPANY	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.PERSONS.COMPANY_ID	1	
<derived3>	2	SUBQUERY	ALL					6487	
PERSONS_CONT	3	DERIVED	ALL					5230	Using where
CONTACTS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	T2.CONTACTS_ID	1	Using index
<derived4>	2	SUBQUERY	ALL					6487	
PERSONS_ADDR	4	DERIVED	ALL					6820	Using where
ADDRESS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	T1.ADDRESS_ID	1	Using index
USERS	2	SUBQUERY	eq_ref	PRIMARY	PRIMARY	4	dealer.COMPANY.USERS_ID	1	Using where; Using index

Obrázek 31 – Exekuční plán dotazu 2.5 A

Exekuční plán potvrzuje předchozí domněnku o značné náročnosti vnějších spojení tabulek s velkým počtem řádků. Dále je zde patrné, že z tabulky PERSONS je zpracováváno zbytečné množství řádků, které pak vstupuje do operace spojení tabulek. Díky tomu je značně zvýšen počet spojení a čas provádění.

Návrh optimalizovaného kódu SQL

V tomto případě bohužel není možné odstranit úplné prohledávání tabulek ADDRESS a CONTACTS, tudíž je nutné zaměřit se na minimalizaci počtu uvažovaných řádků v tabulce PERSONS. Toho lze docílit vytvořením vnořeného dotazu s filtrem na cizí klíč sloupce COMPANY_ID. Do výběru stačí zahrnout pouze ty řádky, jejichž hodnota ve sloupci COMPANY_ID odpovídá sadě identifikátorů firem příslušného uživatele, pro kterého je zobrazován výpis. Níže je uveden kód upraveného SQL dotazu s totožnou funkcí.

```
SELECT
  (SELECT COUNT(*) FROM
  (SELECT * FROM PERSONS WHERE COMPANY_ID IS NULL OR COMPANY_ID IN ((SELECT
COMPANY.ID FROM COMPANY WHERE ( USERS_ID=10 OR USERS_ID IS NULL )) )) T4
  LEFT  JOIN COMPANY ON T4.COMPANY_ID = COMPANY.ID
  LEFT JOIN (SELECT * FROM PERSONS_CONT WHERE MAIN=1) T2 ON
T2.PERSONS_ID=T4.ID
  LEFT JOIN CONTACTS ON CONTACTS_ID=CONTACTS.ID
  LEFT JOIN (SELECT * FROM PERSONS_ADDR WHERE MAIN=1) T1 ON
T1.PERSONS_ID=T4.ID
  LEFT JOIN ADDRESS ON ADDRESS_ID=ADDRESS.ID
  LEFT JOIN USERS ON COMPANY.USERS_ID=USERS.ID) AS TC,
T4.ID AS ID, T4.NAME AS NAME, T4.SURNAME AS SURNAME, COMPANY.NAME AS
COMPNAME, STREET, CITY, CONTACTS.VALUE AS CEMAIL, T4.MAIN AS MA,
DATE_FORMAT(T4.CREATED, '%D.%M.%Y %H:%I:%S' ) AS CREATED
FROM
  (SELECT * FROM PERSONS WHERE COMPANY_ID IS NULL OR COMPANY_ID IN ((SELECT
COMPANY.ID FROM COMPANY WHERE ( USERS_ID=10 OR USERS_ID IS NULL )) )) T4
  LEFT JOIN COMPANY ON T4.COMPANY_ID = COMPANY.ID
  LEFT JOIN (SELECT * FROM PERSONS_CONT WHERE MAIN=1) T2 ON
T2.PERSONS_ID=T4.ID
  LEFT JOIN CONTACTS ON CONTACTS_ID=CONTACTS.ID
  LEFT JOIN (SELECT * FROM PERSONS_ADDR WHERE MAIN=1) T1 ON
T1.PERSONS_ID=T4.ID
  LEFT JOIN ADDRESS ON ADDRESS_ID=ADDRESS.ID
  LEFT JOIN USERS ON COMPANY.USERS_ID=USERS.ID
  ORDER BY T4.ID ASC       LIMIT 20;WHERE  T5.ACTIVE=1  AND  (
T5.USERS_ID=10 OR T5.USERS_ID IS NULL )  ORDER BY T5.ID ASC       LIMIT 20;
```

Na následujícím obrázku 32 je uveden příslušný exekuční plán upraveného dotazu. Z exekučního plánu je patrné, že došlo k výraznému omezení počtu řádků tabulky PERSONS, pro které je prováděno úplné prohledávání tabulek ADDRES a CONTACTS. Tato změna měla ve výsledku značný vliv na zrychlení doby provádění celého dotazu.

table	id	select_type	type	possible_keys	key	key_len	ref	rows	Extra
<derived7>	1	PRIMARY	ALL					82	Using temporary; U...
persons	7	DERIVED	ALL	PERSONS_COMPA...				5560	Using where
COMPANY	1	PRIMARY	eq_ref	PRIMARY	PR...	4	T4.COMPANY_ID	1	
<derived9>	1	PRIMARY	ALL					6487	
PERSONS_CONT	9	DERIVED	ALL					5230	Using where
CONTACTS	1	PRIMARY	eq_ref	PRIMARY	PR...	4	T2.CONTACTS_ID	1	
<derived10>	1	PRIMARY	ALL					6487	
PERSONS_ADDR	10	DERIVED	ALL					6820	Using where
ADDRESS	1	PRIMARY	eq_ref	PRIMARY	PR...	4	T1.ADDRESS_ID	1	
USERS	1	PRIMARY	eq_ref	PRIMARY	PR...	4	dealer.COMPANY.US...	1	Using index
company	8	DEPENDENT S...	unique_subquery	PRIMARY,COMPAN...	PR...	4	func	1	Using where
<derived3>	2	SUBQUERY	ALL					82	
persons	3	DERIVED	ALL	PERSONS_COMPA...				5560	Using where
COMPANY	2	SUBQUERY	eq_ref	PRIMARY	PR...	4	T4.COMPANY_ID	1	
<derived5>	2	SUBQUERY	ALL					6487	
PERSONS_CONT	5	DERIVED	ALL					5230	Using where
CONTACTS	2	SUBQUERY	eq_ref	PRIMARY	PR...	4	T2.CONTACTS_ID	1	Using index
<derived6>	2	SUBQUERY	ALL					6487	
PERSONS_ADDR	6	DERIVED	ALL					6820	Using where
ADDRESS	2	SUBQUERY	eq_ref	PRIMARY	PR...	4	T1.ADDRESS_ID	1	Using index
USERS	2	SUBQUERY	eq_ref	PRIMARY	PR...	4	dealer.COMPANY.US...	1	Using index
company	4	DEPENDENT S...	unique_subquery	PRIMARY,COMPAN...	PR...	4	func	1	Using where

Obrázek 32 – Exekuční plán dotazu 2.5 B

Vyhodnocení optimalizace dotazu č. 2.5

V tomto případě byla dlouhá doba provádění způsobena několika úplnými prohledáváními tabulek s velkým počtem záznamů pro každý řádek výchozí tabulky PERSONS. Tato úplná prohledávání nebylo možno odstranit, nicméně bylo možné optimalizovat počty řádků, pro které byla tato prohledávání realizována. Díky využití vnořených dotazů a správné volbě restriktivních podmínek došlo k citelnému zvýšení rychlosti dotazu. Statistika naměřených časů provádění je uvedena níže.

Tabulka 16 – Časová statistika dotazu č. 2.5

		Čas provádění [s]				
Dotaz		**2.5 A**	**2.5 A - Table cache**	**2.5 B**	**2.5B - Table cache**	**2.5 B - Query cache**
Číslo měření	1	16,704	13,574	0,924	0,289	0,282
	2	16,725	13,547	0,917	0,284	0,277
	3	16,749	13,575	0,919	0,287	0,279
	4	16,357	13,735	0,921	0,287	0,281
	5	16,699	13,710	0,926	0,283	0,273
Průměr [s]		**16,647**	**13,628**	**0,921**	**0,286**	**0,278**

6.4.6 Shrnutí optimalizace databázových dotazů na platformě MySQL

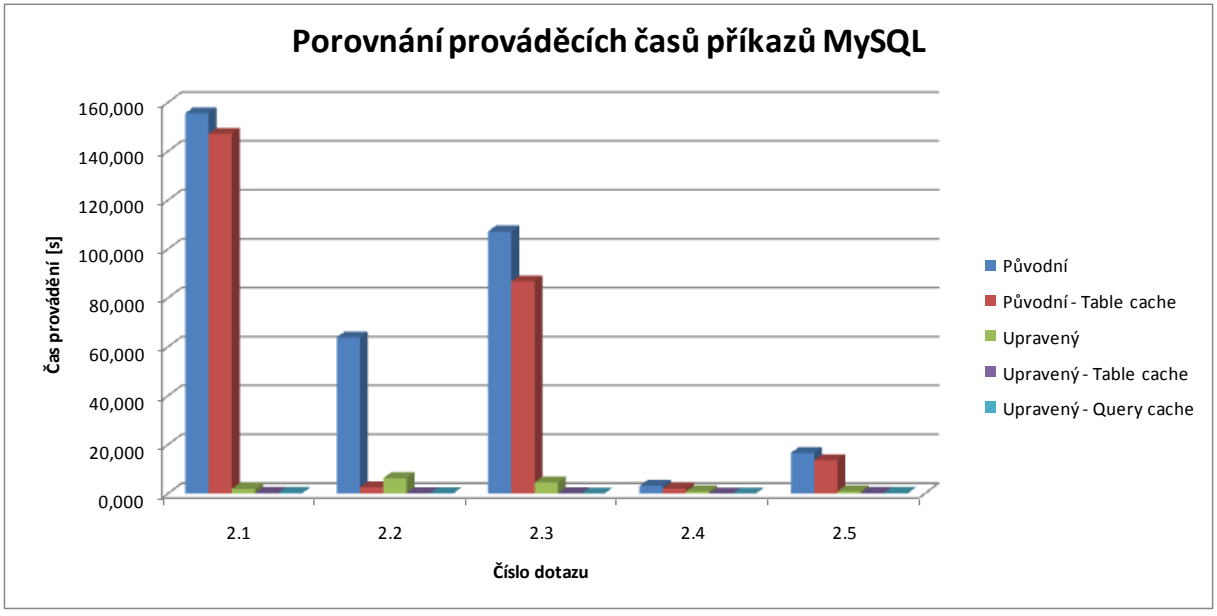
Po naplnění aplikace větším množstvím dat se v případě databázového serveru MySQL objevilo několik dotazů, které byly díky svému nízkému výkonu prakticky nepoužitelné. Nízký výkon dotazů byl většinou způsoben univerzálními mechanizmy, které v aplikaci zajišťují různé tabulkové výpisy. Tyto univerzální mechanizmy dobře fungovaly

pro standardní tabulkové výpisy, nicméně pro složitější výpisy, které byly získávány ze spojení většího množství tabulek, se ukázaly jako nevhodné.

Po provedení rozboru exekučních plánů zmiňovaných dotazů byly odhaleny příčiny způsobující nízký výkon příslušných dotazů. Následně byly SQL dotazy přepracovány a jejich výkon byl tak značně vylepšen. V následující tabulce jsou uvedeny jednotlivé dotazy s časovými údaji, jež popisují původní dobu provádění a dobu provádění po optimalizaci. SQL kódy příslušných dotazů jsou uvedeny buď v předchozí části, nebo na přiloženém CD. Níže je uvedena tabulka shrnující průměrné časy provádění u jednotlivých SQL dotazů pro databázový server MySQL.

Tabulka 17 – Průměrné časy provádění SQL dotazů pro server MySQL

Číslo dotazu	Původní [s]	Původní - Table cache [s]	Upravený [s]	Upravený - Table cache [s]	Upravený - Query cache [s]
2.1	155,585	147,110	1,931	0,314	0,297
2.2	63,865	2,599	6,307	0,177	0,170
2.3	107,223	86,768	4,558	0,126	0,011
2.4	3,205	2,087	0,921	0,077	0,075
2.5	16,647	13,628	0,921	0,286	0,278



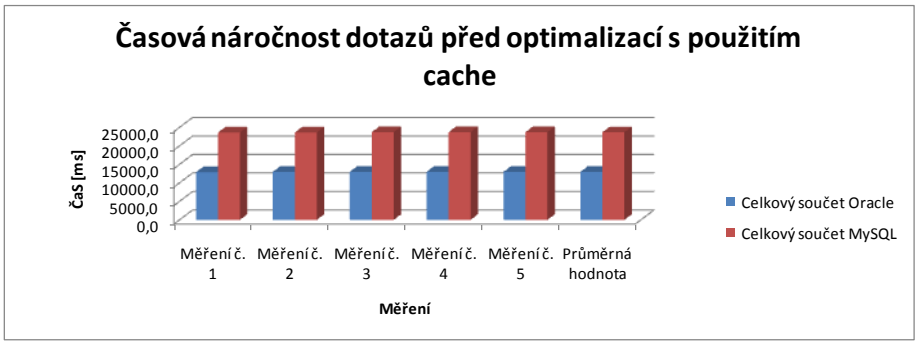
Obrázek 33 – Graf prováděcích časů původních a upravených SQL dotazů

7 POROVNÁNÍ RYCHLOSTI DATABÁZOVÝCH PLATFOREM

Provést relevantní porovnání rychlostí obou databázových platforem je poměrně komplikované, jelikož databáze nebyly instalovány na samostatném serveru, ale na lokálním stroji, kde kromě samotné databáze probíhalo množství paralelních procesů běžících v rámci operačního systému. Dále je třeba uvažovat, že v případě databáze Oracle se jedná o její základní variantu, jejíž možnosti konfigurace a využití systémových zdrojů jsou značně omezeny (omezení byla konkrétně popsána v kapitole 4.1.1). Dalším důležitým faktem je, že SQL dotazy, které databáze zpracovává, nejsou totožné. Rozdíly jsou například v omezení počtu výsledných řádků, práci s datem a časem, způsobu generování unikátních identifikátorů apod. V neposlední řadě je třeba zmínit, že rychlost databázových dotazů značně souvisí s kvalitou a správností jejich kódu. Pro jednoznačné porovnání rychlostí by tedy bylo nutné databáze nainstalovat na totožný hardware, při použití identických konfigurací. Rovněž by měla být provedena optimalizace všech databázových dotazů, které jsou v rámci aplikace používány, nicméně dosažení takového stavu je v praxi téměř nereálné. Výsledky prezentované v této kapitole si tedy nekladou za cíl obecně rozhodnout, která z databází je rychlejší. Tyto výsledky jsou úzce spjaty s konkrétní aplikací, provozované na konkrétním hardwaru s danou konfigurací.

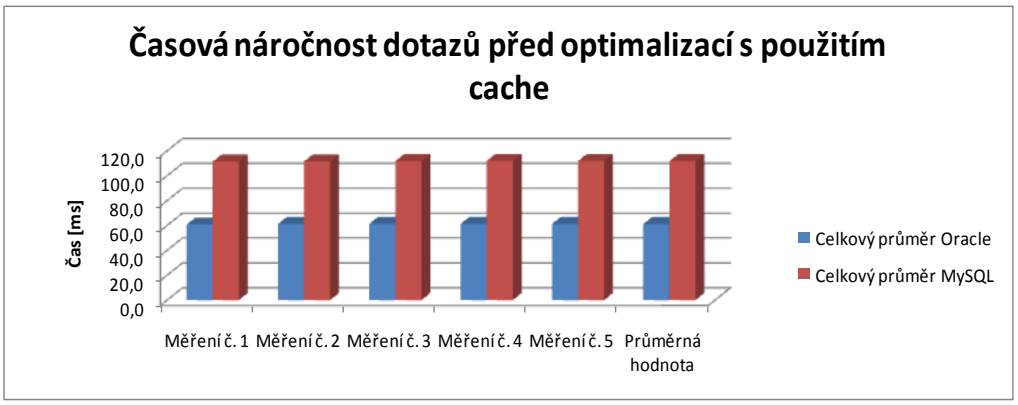
Rychlosti jednotlivých dotazů, porovnávané v této části, byly získány prováděním testovacího scénáře zaměřeného na operace typu select. To vychází ze zadání práce, kdy bylo třeba pracovat s náročnými databázovými dotazy, kterými jsou právě složité výpisy dat získaných z náročných spojení tabulek. Rychlost dalších operací (insert, update, delete) nebyla v této práci testována.

Obecně je z výsledků měření možné konstatovat, že optimalizátor databázového serveru Oracle dokázal volit vhodnější (z hlediska časové náročnosti) přístupové cesty k datům v případě, kdy databázový dotaz nebyl sestaven optimálně. Výše uvedené tvrzení je patrné z následujících grafů. Na obrázku 34 je graf, který zobrazuje celkové sumy časů potřebných pro vykonání celého testovacího scénáře. Jedná se tedy o součet prováděcích časů jednotlivých SQL dotazů. Celkem bylo realizováno pět měření, jejichž průměrná hodnota je v grafu rovněž uvedena.



Obrázek 34 – Graf celkových časů SQL dotazů před optimalizací s použitím cache

Na obrázku 35 je uveden graf průměrných hodnot prováděcích časů. Jedná se tedy o průměrný čas, který odpovídá jednomu databázovému dotazu. Všechny časy byly zaznamenávány při běžném provozu aplikace, tudíž databázový server mohl využívat data uložená v paměti cache a pomocných bufferech.

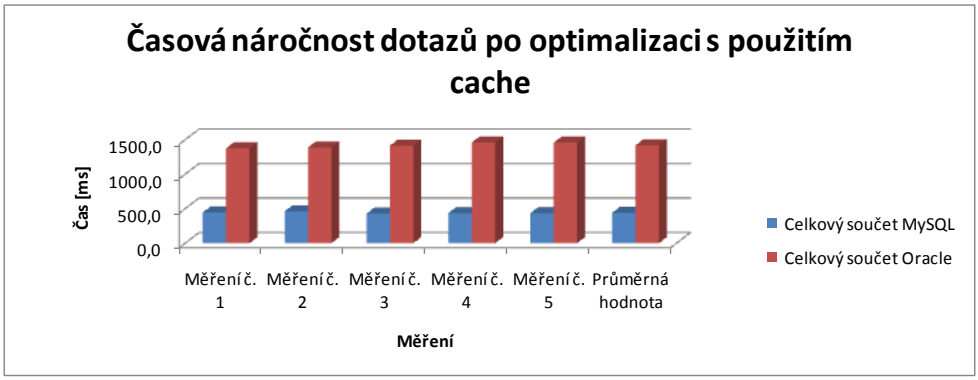


Obrázek 35 – Graf průměrů časů SQL dotazů před optimalizací s použitím cache

Při vyhodnocení předchozích grafů je třeba brát v úvahu to, že na vysokém prováděcím čase databáze MySQL má značný podíl malé množství SQL dotazů, jejichž kód nebyl konstruován ideálně a způsoboval velký nárůst prováděcí doby.

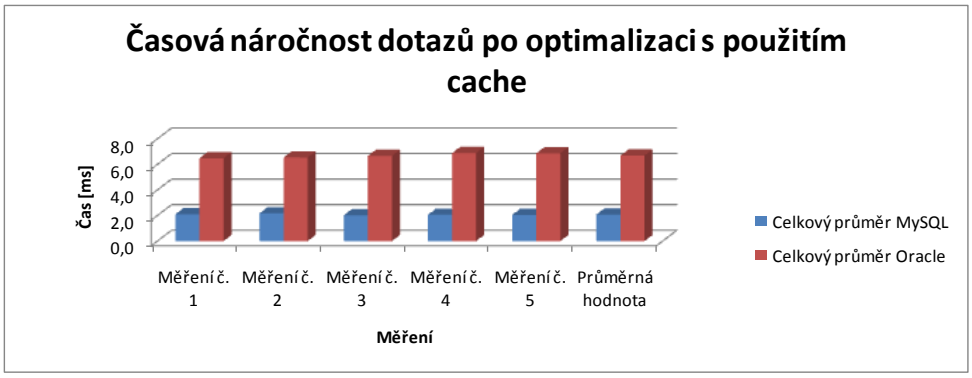
Po provedení optimalizace databázových dotazů došlo k výraznému snížení náročnosti vybraných příkazů, které způsobovaly zpomalování práce s aplikací. V případě databáze MySQL byly výsledky velice dobré. K rapidnímu snížení náročnosti došlo v důsledku odstranění chybně konstruovaných dotazů, se kterými si databázový server nedokázal vhodně poradit. Optimalizace prováděná na databázovém serveru Oracle nedosahovala z hlediska absolutních hodnot prováděcích časů tak výrazných výsledků. Tato situace je dána tím, že vstupní stav před optimalizací byl mnohem lepší, než v případě MySQL. Dále jsou uvedeny grafy, ze kterých je patrné snížení času potřebného na provedení testovacího scénáře.

Na obrázku 36 je uveden graf, který zobrazuje celkové sumy časů potřebných pro vykonání testovacího scénáře (odpovídá grafu na obr. 34). Z grafu je patrné, že po provedení optimalizace již MySQL dosahuje lepších výsledků než databáze Oracle.



Obrázek 36 – Graf celkových časů SQL dotazů po optimalizaci s použitím cache

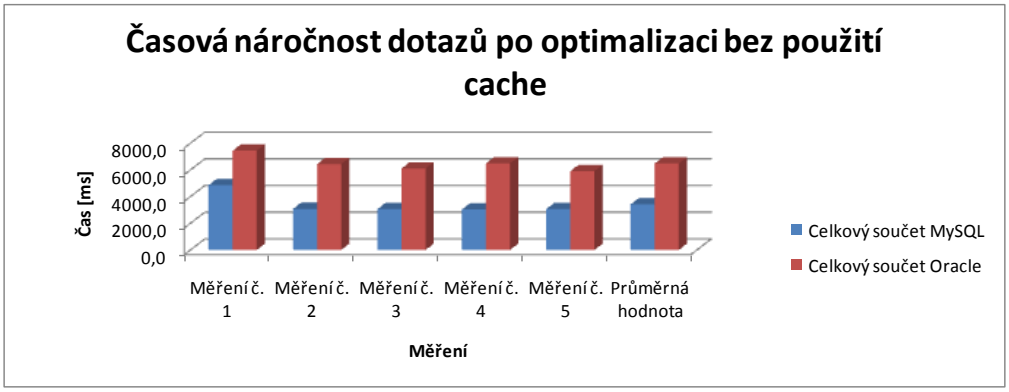
Na následujícím obrázku 37 je uvedena statistika průměrných časů, potřebných pro vykonání jednoho databázového dotazu. Tento graf je možno porovnávat s grafem na obrázku 35. Z obou grafů je patrné, že došlo k velmi výraznému snížení časů potřebných pro provedení SQL dotazů. Díky tomu, že po provedení optimalizace pracuje databázový server při jednotlivých dotazech s řádově menšími počty řádků, nebo data získává pouze z indexů, dochází k lepšímu využívání práce s pamětí cache a tím i k podstatnému zlepšení výkonu.



Obrázek 37 – Graf průměrů časů SQL dotazů po optimalizaci s použitím cache

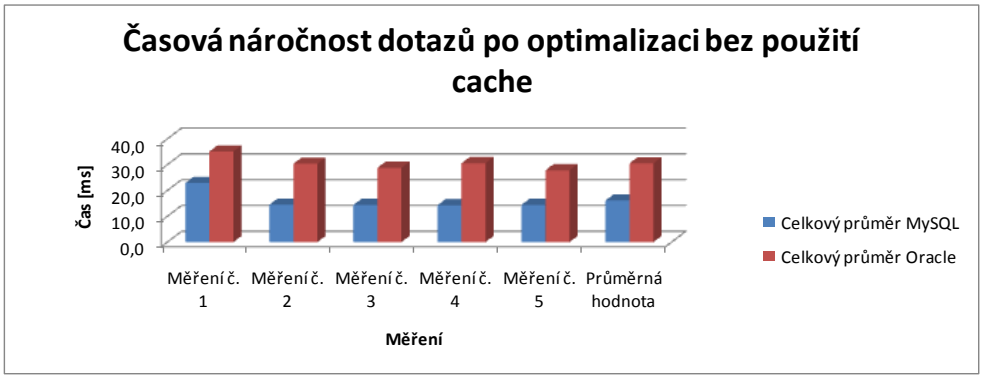
Další měření bylo prováděno s totožným testovacím scénářem, nicméně mezi jednotlivými opakováními scénáře byl prováděn restart databázového serveru. Díky této operaci došlo ke smazání dočasných dat ukládaných do paměti cache a pomocných bufferů. Toto měření tedy neodráží běh aplikace v reálném prostředí, jelikož paměť cache bývá standardně využívána. Zmiňovaný postup byl zvolen především pro zajištění srovnatelných podmínek při testování. Na následujícím obrázku 38 je uveden graf celkové časové náročnosti SQL

dotazu, která byla měřena po provedení optimalizace. Hodnoty je možné porovnat s grafem na obr. 36 a obr. 34. Z grafu vyplývá, že lepších celkových časů dosahoval systém při použití databáze MySQL.



Obrázek 38 – Graf celkové náročnosti optimalizovaných dotazů bez použití cache

Na obrázku 39 je uveden graf průměrných hodnot databázových dotazů, naměřených po provedení optimalizace. Tento graf je možné porovnat s obr. 37 a obr. 35.



Obrázek 39 – Graf průměrné náročnosti optimalizovaných dotazů bez použití cache

V příloze na CD a v příloze D jsou dále uvedeny grafy, na kterých je zobrazena průměrná doba vykonávání jednotlivých příkazů SQL (průměry jsou počítány vždy pro konkrétní dotaz za všechna prováděná měření). Z uvedených grafů je možné konstatovat, že u této konkrétní aplikace dosahovala databáze Oracle oproti MySQL vyššího výkonu v případě, kdy konstrukce jednotlivých SQL dotazů nebyly ideální (tedy před provedením optimalizace). Po provedení optimalizace došlo k výraznému nárůstu rychlosti databáze MySQL a právě tato databáze dosahovala v testu lepších výsledků než Oracle. Závěrem je třeba ještě upozornit, že proces optimalizace nebyl proveden u všech databázových dotazů, tudíž rychlost obou platforem by se v důsledku další optimalizace mohla měnit.

8 ZÁVĚR

Cíle této práce je možné rozdělit na dvě části. První částí bylo vytvoření funkční databázové aplikace pro snadnou tvorbu obchodních nabídek. Druhou částí pak bylo provedení optimalizace vybraných databázových dotazů, které vytvořená aplikace využívá.

Díky analýze stávajících řešení se podařilo navrhnout a vytvořit poměrně rozsáhlou aplikaci, která obsahuje všechny požadavky specifikované v zadání práce. Jedná se především o propojení se dvěma různými databázovými platformami a dále o využití existujících webových služeb, jako je komunikace s registrem ARES nebo napojení na aplikační programové rozhraní společnosti Google. Díky využití technologie AJAX je ovládání aplikace jednoduché a sugestivní, což odpovídá současným trendům ve vývoji webových aplikací.

Z hlediska implementační části je aplikace dokončena. Nasazení v produkčním prostředí by mělo ještě předcházet detailní otestování všech částí aplikace. Detailní otestování se nepodařilo realizovat zejména kvůli značné časové náročnosti, která je dána velkým množství modulů a funkcionalit dostupných v systému. V aplikaci je využíváno přibližně 50 vstupních formulářů a obdobný počet tabulkových výpisů s kombinovanými filtry. Do aplikace je možné přistupovat pod třemi úrovněmi uživatelských práv, což ve výsledku vytváří značné množství testovacích scénářů, které by bylo třeba provést.

Ve druhé části práce byla prováděna optimalizace vybraných SQL dotazů pro obě databázové platformy. Zde se ukázaly značné rozdíly mezi funkčností obou databází. Databázový server Oracle dokázal díky svým automatickým technikám optimalizace udržet rychlost vykonávání složitých databázových dotazů na přijatelné úrovni i v případě, kdy dotazy nebyly navrženy optimálně. Přesto se podařilo vybrat několik dotazů, jejichž doba provádění byla neuspokojivá. Díky analýze exekučních plánů byly odhaleny příčiny nízkého výkonu a navrženy úpravy, které tyto příčiny eliminovaly. Velice přínosné bylo také seznámení s profesionálním nástrojem pro automatické generování alternativních exekučních plánů. Díky tomuto programu se v jednom případě podařilo dosáhnout ještě lepšího výkonu dotazu i po provedení standardní ruční optimalizace založené na znalostech příslušných optimalizačních technik.

V případě serveru MySQL docházelo v některých situacích k nárůstu doby vykonávání dotazů nad přijatelnou mez. Tyto situace se týkaly především databázových dotazů, jejichž kód nebyl

z hlediska optimalizace příliš povedený. K tomuto stavu docházelo z důvodu využití univerzálních mechanismů, které v aplikaci zajišťují tabulkové výpisy. V praxi se ukázalo, že využívání těchto univerzálních mechanismů není u složitějších aplikací příliš vhodné především pro to, že výsledný kód je ve velkém množství případů nutné nakonec ručně upravit. Tím dojde ke ztrátě časové úspory získané využitím univerzálního výpisu.

I v případě databáze MySQL se po analýze exekučních plánu podařilo navrhnout řešení kritických situací, které způsobovaly prodlužování času potřebného pro vykonání databázového dotazu. Výsledné časy, kterých se po optimalizaci podařilo dosáhnout, byly velice dobré.

Při tvorbě aplikace se rovněž ukázalo, že je velice problematické vytvořit systém tak, aby transparentně komunikoval se dvěma různými databázemi. V ideálním případě pomocí totožných SQL příkazů. Rozdíly mezi syntaxí jednotlivých databázových platforem je možné překonat využitím vhodné oddělovací vrstvy, která provádí převod syntaxe SQL dotazu do správného tvaru odpovídajícího používané platformě. Nicméně v případě optimalizace, kdy je třeba dotaz pro každou databázovou platformu konstruovat odlišně, nebo v případě, kdy jsou pro optimalizaci využívány hinty, dochází k situaci, při které je nutné strukturu dotazu konstruovat pro každou z platforem odlišně.

Tato práce byla velice přínosná především díky praktickému využití teoretických znalostí souvisejících s optimalizací databázových dotazů. Teoretické poznatky jsou samozřejmě nepostradatelné, nicméně v praxi je možné setkat se s velkým množstvím softwarových nástrojů, které umožňují posunout optimalizaci opět směrem kupředu. Především v oblasti generování alternativních exekučních plánů, kdy těchto alternativ mohou být řádově stovky, je použití specializovaného softwaru velmi užitečné.

Přes veškeré komplikace související s tvorbou univerzálního systému v jazyce PHP se podařilo splnit zadání práce v celém rozsahu. Byla vytvořena poměrně rozsáhlá aplikace, kterou je možné využít i v komerčním prostředí. Následná optimalizace odhalila slabá místa v používaných SQL dotazech, jejichž eliminací se podařilo dobu provádění jednotlivých databázových dotazů značně snížit. Díky optimalizaci je aplikaci možné využívat i pro uchovávání většího objemu dat bez výrazné ztráty uživatelské použitelnosti a rychlosti.

9 SEZNAM LITERATURY

[1] Co je a není CRM: neboli Řízení vztahů se zákazníky. *Systémy On Line* [online]. 2012 [cit. 2013-05-05]. Dostupné z: http://www.systemonline.cz/crm/co-je-a-neni-crm.htm.

[2] Jak vyzrát nad dlouhými obchodními případy: Role CRM v komplexní komunikaci se zákazníkem. *Systémy On Line* [online]. 2012 [cit. 2013-05-05]. Dostupné z: http://www.systemonline.cz/crm/jak-vyzrat-nad-dlouhymi-obchodnimi-pripady.htm.

[3] Získajte Offeris. *Offeris* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://www.offeris.com/sk/ziskajte-offeris.

[4] Nabídky Plus. *LAN Consult* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://www.lc.cz/nabidky-plus.html.

[5] Tvorba Cenové nabídky a Faktury. *RodutData - Rozpočtové programy* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://www.rodutdata.cz/www-rodutdata-cz/eshop/0/0/5/3-Tvorba-Cenove-nabidky-a-Faktury.

[6] Oracle and MySQL Compared. *Oracle Documentation* [online]. 2008 [cit. 2013-05-05]. Dostupné z: http://docs.oracle.com/cd/E12151_01/doc.150/e12155/oracle_mysql_compared.htm.

[7] Úvod do architektury MVC. *Zdroják* [online]. 2009 [cit. 2013-05-05]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc.

[8] Oracle Database 11g Express Edition Quick Tour. *Oracle* [online]. 2011 [cit. 2013-05-05]. Dostupné z: http://www.oracle.com/technetwork/articles/sql/11g-xe-quicktour-498681.html.

[9] About MySQL. *MySQL* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://www.mysql.com/about/.

[10] SCHLOSSNAGLE, George. *Pokročilé programování v PHP 5*. Brno: Zoner Press, 2004, 640 s. ISBN 80-868-1514-5.

[11] JQuery Learning Center. *JQuery Learning Center* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://learn.jquery.com/

[12] Quick Start. *Dibi is Database Abstraction Library for PHP 5* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://generator.citace.com.

[13] Dokumentace. *Nette Framework* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://doc.nette.org/cs.

[14] TCPDF is a FLOSS PHP class for generating PDF documents. *TCPDF* [online]. 2011 [cit. 2013-05-05]. Dostupné z: http://www.tcpdf.org/index.php.

[15] Google APIs Client Library for PHP. *Google Code* [online]. 2013 [cit. 2013-05-05]. Dostupné z: https://code.google.com/p/google-api-php-client.

[16] BURLESON, Donald K. *Oracle Tuning: The Definitive Reference*. Second edition. Kittrell : Rampant Techpress, 2011. 1200 s. ISBN 0979795192.

[17] LONEY, Kevin a Marlene THERIAULT. *Mistrovství v Oracle: Kompletní průvodce tvorbou, správou a údržbou databází*. Vyd. 1. Praha: Computer Press, 2002. ISBN 80-7226-635-7.

[18] DUBOIS, Paul. *MySQL profesionálně: komplexní průvodce použitím, programováním a správou MySQL*. Vyd. 1. Překlad Jan Pokorný. Brno: Mobil Media, 2003, 1071 s. ISBN 80-865-9341-X.

[19] SQL Tuning Overview. *Oracle Documentation* [online]. 2008 [cit. 2013-05-05]. Dostupné z: http://docs.oracle.com/cd/B19306_01/server.102/b14211/sql_1016.htm.

[20] Introduction to the Optimizer. *Oracle Documentation* [online]. 2002 [cit. 2013-05-05]. Dostupné z: http://docs.oracle.com/cd/B10500_01/server.920/a96533/optimops.htm.

[21] ROWNUM Pseudocolumn. *Oracle Documentation* [online]. 2005 [cit. 2013-05-05]. Dostupné z: http://docs.oracle.com/cd/B19306_01/server.102/b14200/pseudocolumns009.htm.

[22] The Query Optimizer. *Oracle Documentation* [online]. 2008 [cit. 2013-05-05]. Dostupné z: http://docs.oracle.com/cd/B19306_01/server.102/b14211/optimops.htm.

[23] Range Optimization. *MySQL 5.0 Reference Manual* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://dev.mysql.com/doc/refman/5.0/en/range-optimization.html.

[24] Index Merge Optimization. *MySQL 5.0 Reference Manual* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://dev.mysql.com/doc/refman/5.0/es/index-merge-optimization.html.

[25] Nested-Loop Join Algorithms. *MySQL 5.0 Reference Manual* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://dev.mysql.com/doc/refman/5.0/en/nested-loop-joins.html.

[26] LACKO, Luboslav. *Oracle: správa, programování a použití databázového systému*. 2. dopl. vyd. Překlad Marek Kocan. Brno: Computer Press, 2007, 576 s. ISBN 978-80-251-1490-2.

[27] Schema Objects. *Oracle Documentation* [online]. 2005 [cit. 2013-05-05]. Dostupné z: http://docs.oracle.com/cd/B19306_01/server.102/b14220/schema.htm.

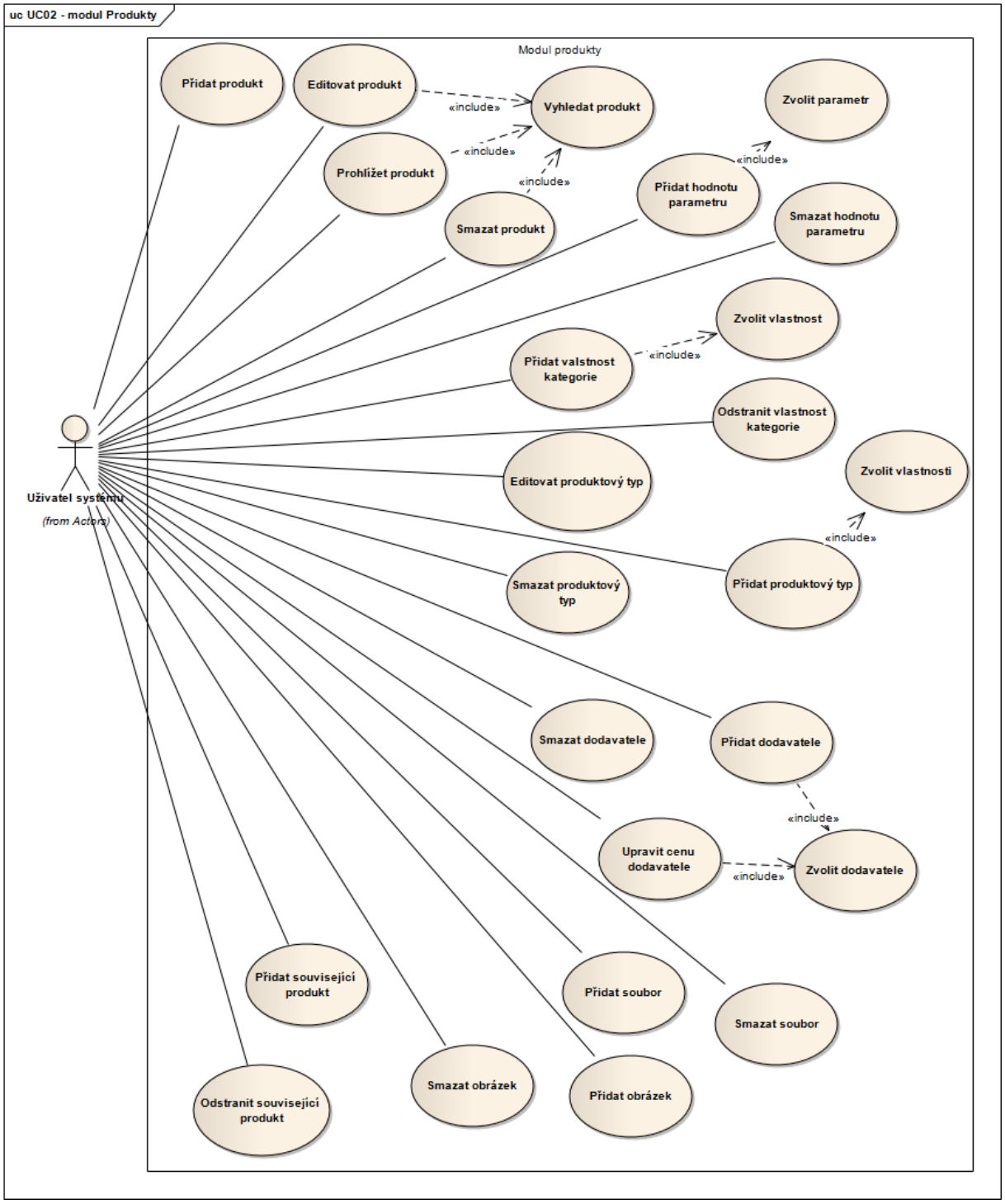
[28] Oracle Bitmap Index Techniques. *Oracle Consulting, Oracle Support and Oracle Training by BC Oracle Consulting* [online]. 2012 [cit. 2013-05-05]. Dostupné z: http://www.dba-oracle.com/oracle_tips_bitmapped_indexes.htm.

[29] How MySQL Uses Indexes. *MySQL 5.0 Reference Manual* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://dev.mysql.com/doc/refman/5.0/en/mysql-indexes.html.

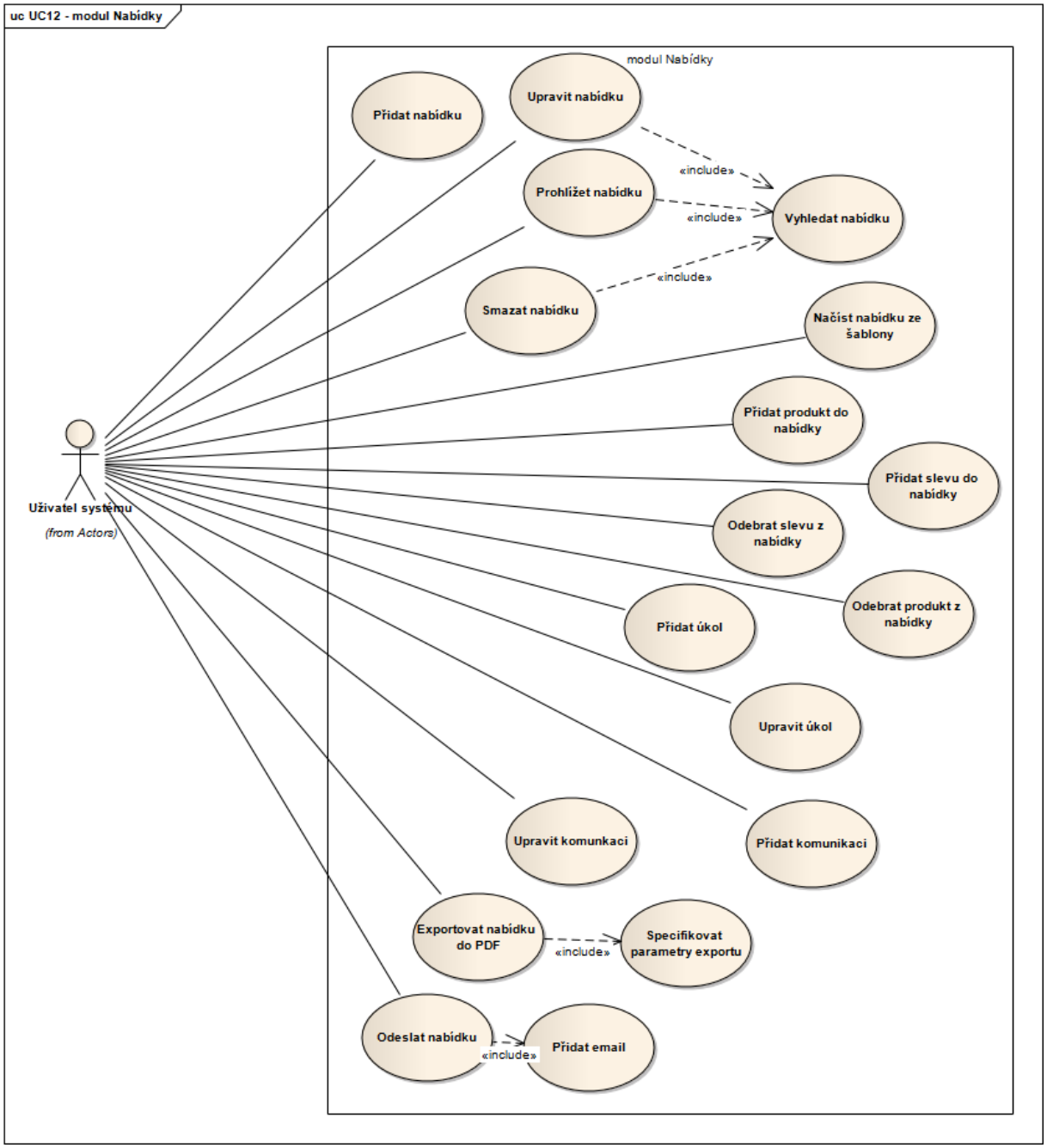
10 PŘÍLOHY

Příloha A – Případ užití modulu Produkty 96
Příloha B – Případ užití modulu Nabídky 97
Příloha C – Postup instalace aplikace 98
Příloha D – Grafy rychlosti všech SQL dotazů testovacího scénáře 102
Příloha E – Fragment datového modelu 103
Příloha F – Ukázka vzhledu aplikace 104
Příloha G – Seznam příloh na CD 105

Příloha A – Případ užití modulu Produkty



Příloha B – Případ užití modulu Nabídky



Příloha C – Postup instalace aplikace

Veškeré zdrojové kódy aplikace i DDL skripty pro vytvoření příslušné databáze jsou uvedeny na přiloženém CD. Vzhledem k tomu, že se jedná o internetovou aplikaci, je pro její spuštění nutné přizpůsobit příslušné cesty k souborům dle serveru, na kterém bude aplikace provozována. Pro provoz je samozřejmě potřeba webový server s podporou jazyka PHP a příslušná databáze (dle výběru MySQL, nebo Oracle).

Nastavení aplikace

Nastavení cest k souborům a skriptům

Hlavní nastavení přístupových cest probíhá prostřednictvím souboru *lang.php*, který je umístěn v adresáři *editor/lang*. Zde je třeba upravit následující konstanty:

- **WEB_ADDRESS** – konstanta musí obsahovat URL adresu úvodní stránky aplikace. Pro provoz na lokálním serveru může být konstanta nastavena následovně:
  define ("WEB_ADDRESS", "http://localhost/Dealer");
- **EDITOR_ADDRESS** – konstanta obsahuje úvodní stránku editoru aplikace
  Pro provoz na lokálním serveru může být konstanta nastavena následovně:
  define("EDITOR_ADDRESS", "http://localhost/Dealer/editor/index.php");

Toto nastavení postačuje v případě, že je aplikace provozována na doméně druhého řádu. V případě, že by byla aplikace provozována na doméně třetího řádu, je nutné provést korektní nastavení všech přístupových cest v souboru *lang.php*. Význam jednotlivých konstant je uveden v komentářích příslušného souboru.

Dále je třeba nastavit správné cesty pro JavaScriptové funkce. Toto nastavení se provádí v souboru *global_vars.js*, který je umístěn v adresáři *editor/js/scripts*. Zde je třeba upravit hodnoty následujících proměnných:

- **web_path** – konstanta obsahuje URL adresu úvodní stránky aplikace. Pro provoz na lokálním serveru může být konstanta nastavena následovně:
  var web_path = 'http://localhost/Dealer/';
- **editor_path** – obsahuje cestu k úvodní stránce editoru. Pro provoz na lokálním serveru může být konstanta nastavena následovně.
  var editor_path = 'http://localhost/Dealer/editor/';

Nastavení komunikace s databází

Nastavení komunikace s databází probíhá prostřednictvím souboru *conf.php*, který je umístěn v adresáři *editor/lang*. Připojení je nastavováno pomocí pole, které je předáváno metodě *connect* jako argument. Je důležité vyplnit položky:

- **host** – adresa serveru databáze,
- **username** – uživatelské jméno pro přístup aplikace do databáze,
- **password** – heslo pro přístup do databáze,
- **charset** – kódování dat přenášených mezi databází a aplikací.

Pro připojení k databázi Oracle na lokálním serveru mohou být parametry pole následující.

```
  dibi::connect(array(
        'driver' => 'oracle',
        'host' => '127.0.0.1:8080/XE',
        'username' => 'jmeno',
        'password' => '******',
        'charset' => 'UTF8',
        'formatDateTime' => "'Y-m-d'",
        'profiler' => array(
            'run' => TRUE,
            'file' => MAIN_SERVER_FOLDER . '/editor/logOracle.sql',
        ),
    ));
```

Obdobné nastavení je použito pro připojení k databázi MySQL. Zde je navíc potřeba uvést název příslušné databáze (klíč pole database).

```
dibi::connect(array(
        'driver' => 'mysql',
        'host' => 'localhost',
        'username' => 'jmeno',
        'password' => '*****',
        'database' => 'dealer',
        'charset' => 'UTF8',
        'profiler' => array(
            'run' => TRUE,
            'file' => MAIN_SERVER_FOLDER . '/editor/logMySql.sql',
        ),
    ));
```

Aplikace může pracovat jak s databázovým serverem Oracle 11, tak i s databázovým serverem MySQL. Výběr druhu databáze se provádí v souboru *lang.php*. Konkrétně se jedná o konstantu DB, která může mít následující hodnoty:

- *define("DB", 'oracle');* - pro použití s databází Oracle,
- *define("DB", 'mysql');* - pro použití s databází MySQL.

Nastavení komunikace s Google API

Komunikace s Gmail účtem probíhá prostřednictvím Google API. Pro využití tohoto rozhraní je nejprve nutná registrace příslušného projektu a zvolení služeb, které budou prostřednictvím API dostupné (jednotlivé služby mají specifické limity na počet operací provedených v rámci bezplatného provozu). Registrace projektu se provádí pomocí aplikace na webové adrese https://code.google.com/apis/console#access. Pro přístup do aplikace je nutné přihlášení pomocí stávajícího Gmail účtu. Po zaregistrování projektu je třeba v části API Access vygenerovat jedinečné identifikátory, které slouží pro autorizaci aplikace vůči aplikačnímu rozhraní. Identifikátory se vždy váží k příslušné doméně, na které je projekt provozován. Konkrétně je třeba vygenerovat následující údaje:

- **Client ID** – jedinečný identifikátor klienta.
- **Email address** – emailová adresa pro přidělení přístupu k účtu.
- **Client secret** – bezpečnostní klíč k danému Client ID.
- **Redirect URIs** – tento identifikátor odpovídá adrese, na kterou bude API přesměrovávat. V případě tohoto systému musí být adresa následující (identifikátor nazev_domeny bude nahrazen konkrétní doménou, na které bude aplikace provozována):
  http://nazev_domeny/editor/index.php?section=import&module=g-con-import&action=g-con-import.
- **JavaScript origins** – identifikátor ve tvaru doménového jména.

Podrobný návod pro nastavení přístupu do API je dostupný na adrese https://developers.google.com/console/help/#UsingOAuth2.

V závislosti na vygenerovaných hodnotách je třeba upravit následující konstanty v souboru *lang.php*. Místo hodnot xxxxxx budou doplněny vygenerované údaje.

```
define("G_CLIENT_ID",'xxxxxx');
define("G_CLIENT_SECRET",'xxxxxx');
define("G_REDIRECT_URI",'xxxxxx');
define("G_DEVELOPER_KEY",'xxxxxx');
```

Nastavení dávkového odesílání emailů

Kvůli ochraně před zařazením emailových adres do seznamu spamů jsou hromadné emaily odesílány po menších dávkách v určitých časových intervalech. Odesílání emailů je třeba nastavit pomocí služby CRON. Prakticky je možné zvolit jakýkoli časový interval, nicméně je doporučováno nastavení automatického odesílání v intervalu 30 min. Skript *sender.php*, který zajišťuje dávkové odesílání, je umístěn v adresáři *editor/core/scripts/*.

Dalším důležitým parametrem je počet emailů odeslaných v jedné dávce. Tento počet je možné nastavit následující konstantou v souboru *lang.php*.

```
define("EMAIL_BATCH", 15);
```

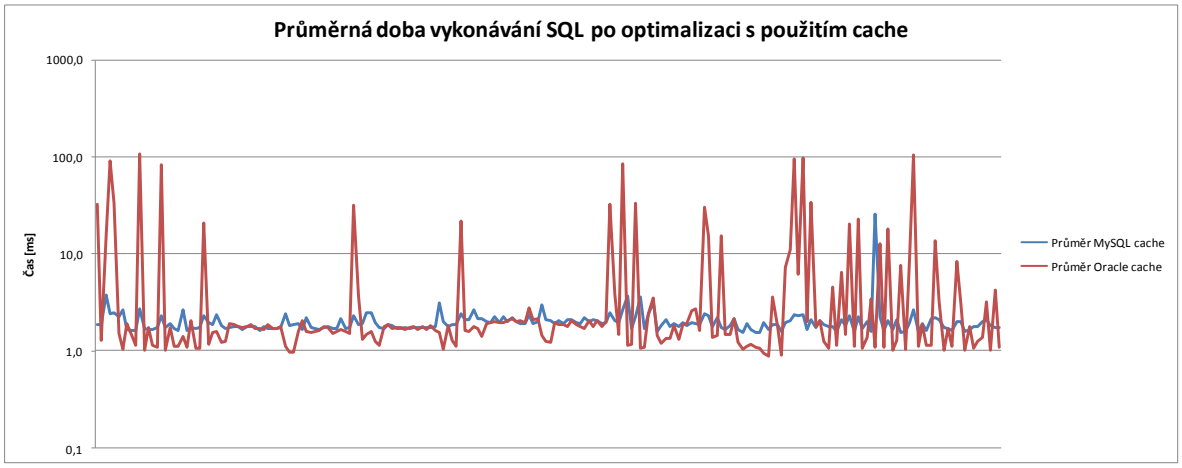
Konkrétní parametry je třeba přizpůsobit konkrétnímu webhostingovému řešení, na kterém je systém provozován.

Vytvoření databáze

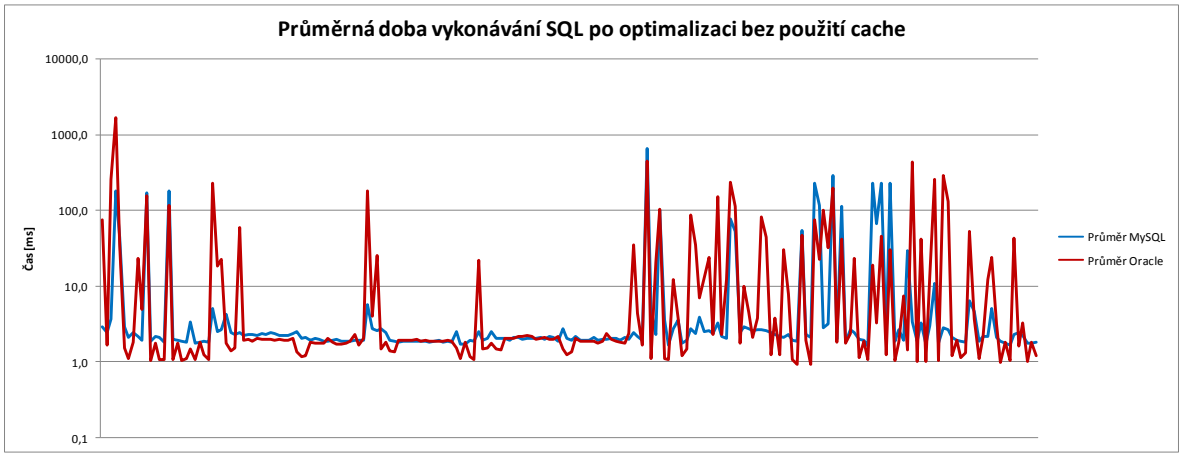
Databázi je možné vytvořit nahráním DDL skriptů, které jsou umístěny na přiloženém CD. Pro MySQL je třeba nejprve vytvořit příslušnou databázi s defaultním kódováním UTF8.

Dále je třeba vytvořit uživatelský účet, který bude sloužit pro přístup aplikace do databáze. Tento účet musí mít pro všechny tabulky a pohledy dostatečné oprávnění, které umožní operace select, insert, update a delete.

Příloha D – Grafy rychlosti všech SQL dotazů testovacího scénáře

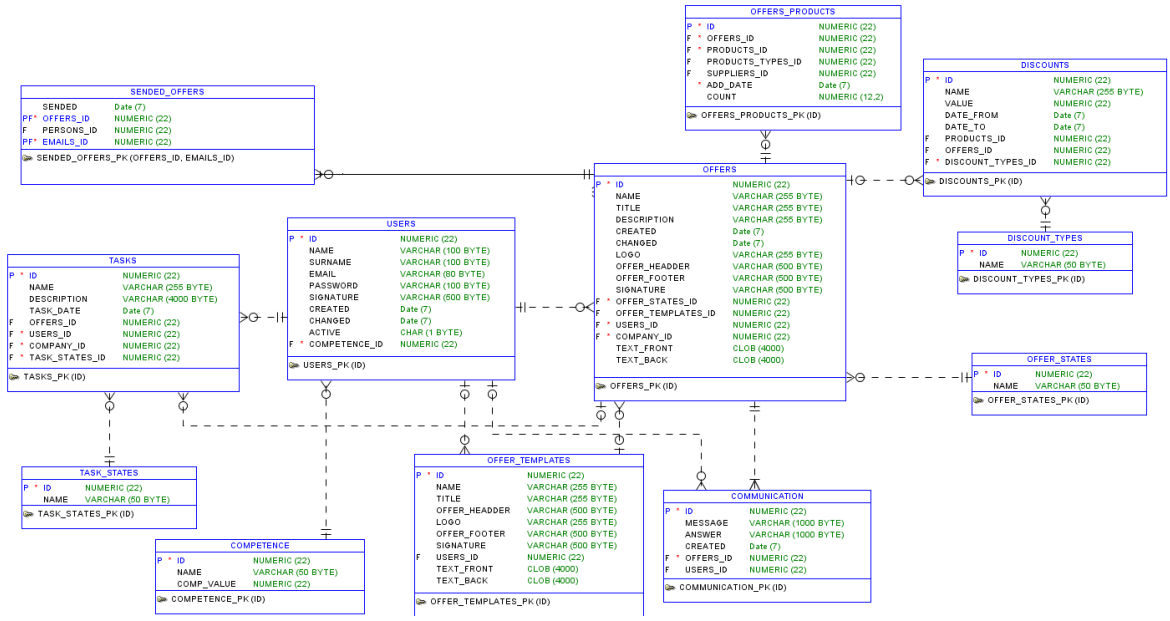


Obrázek D.1 – Graf průměrné doby provádění SQL dotazů s využitím paměti cache

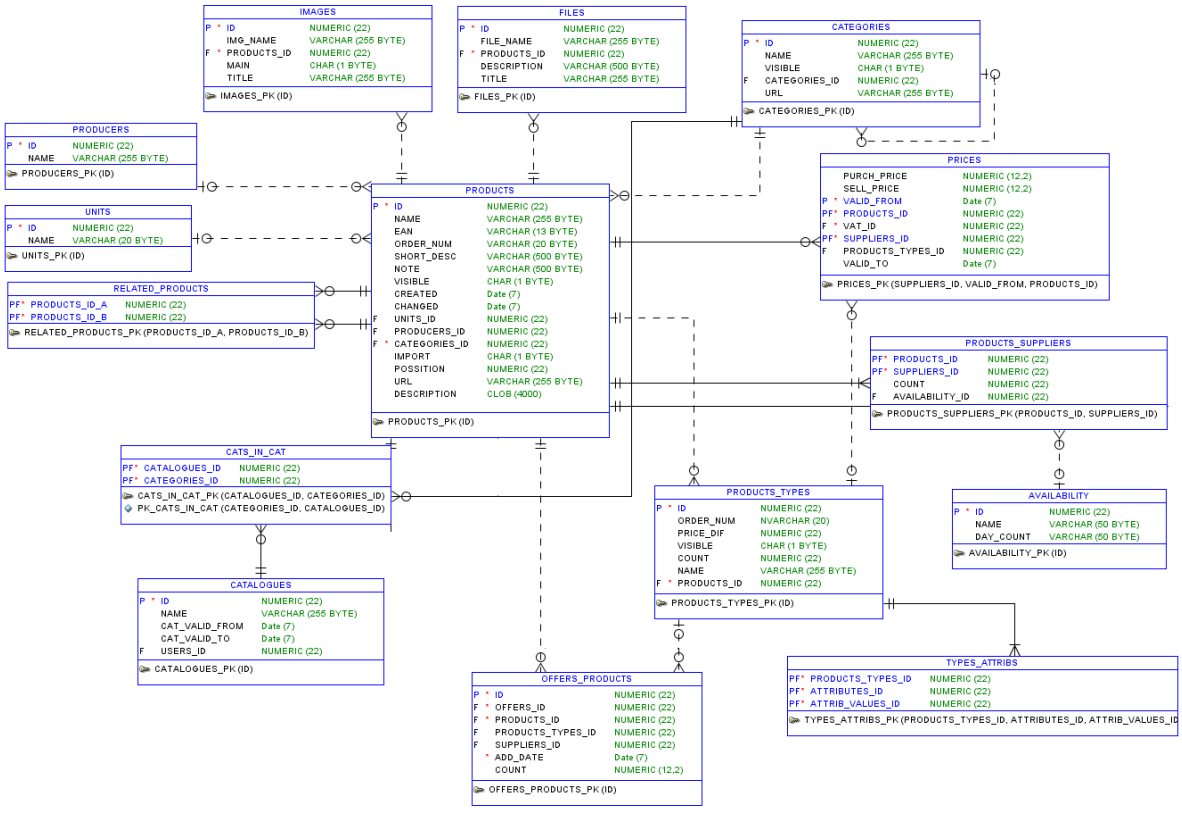


Obrázek D.2 – Graf průměrné doby provádění SQL dotazů bez využití paměti cache

Příloha E – Fragment datového modelu

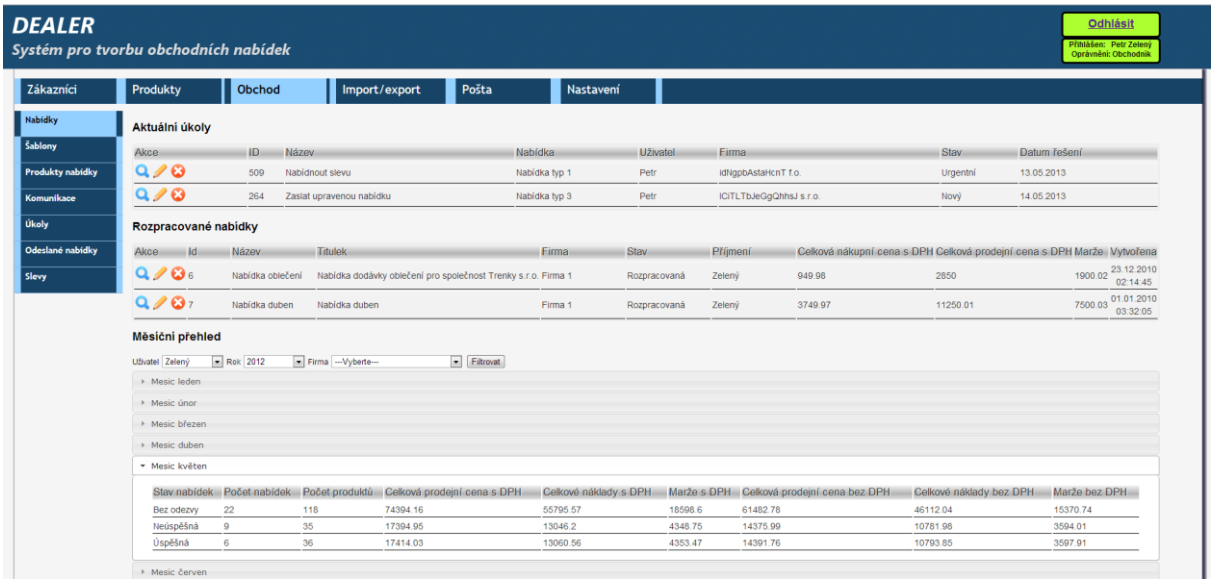


Obrázek E.1 - fragment ER diagramu modulu nabídky

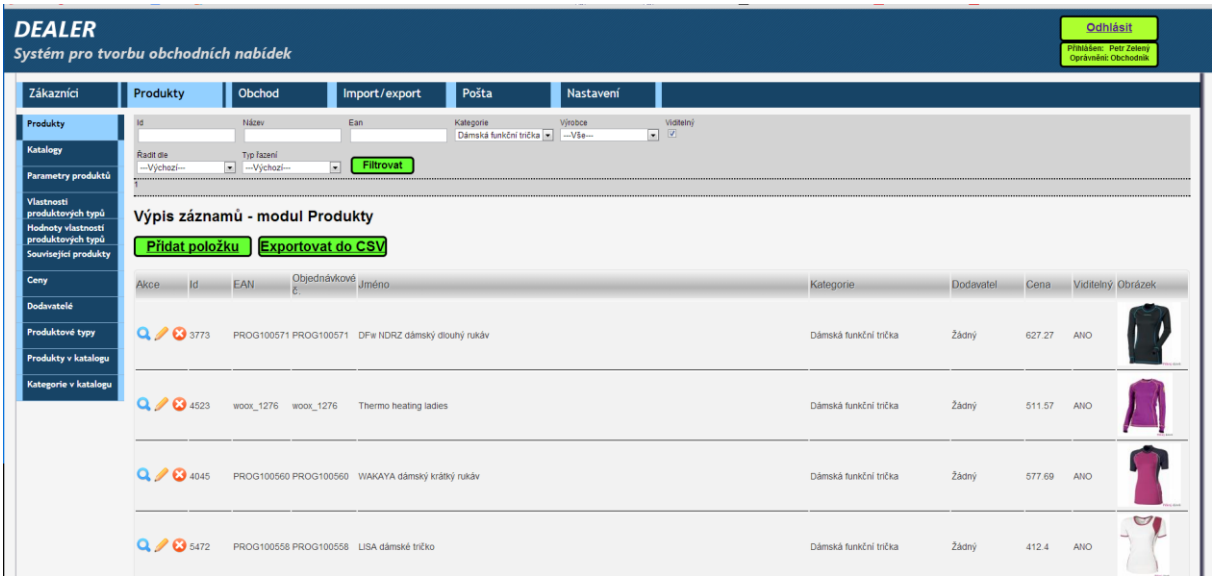


Obrázek E.240 - fragment ER diagramu modulu produkty

Příloha F – Ukázka vzhledu aplikace



Obrázek F.1 - Úvodní obrazovka aplikace



Obrázek F.2 - Výpis produktů

Příloha G – Seznam příloh na CD

Přílohy na CD jsou umístěny ve složce *prilohy*. Vzhledem k velkému počtu souborů jsou dále děleny do zanořených složek.

Datový model

Datový model je umístěn ve složce *datovy_model*. Model byl vytvořen v programu *Oracle SQL Developer Data Modeler verze 3.1.1.703* a obsahuje dva relační modely. Prvním modelem je *Model DEALER – Oracle*, který reprezentuje relační model pro databázi Oracle. Druhým modelem je *Model DEALER – MySQL*, který reprezentuje relační model určený pro databázi MySQL.

DDL script pro vytvoření databáze Oracle

Skript pro vytvoření databáze Oracle je přiložen ve složce *datovy_model*, pod názvem *ddl_oracle.sql*.

DDL script pro vytvoření databáze MySQL

Skript pro vytvoření databáze MySQL je přiložen ve složce *datovy_model*, pod názvem *ddl_mysql.sql*.

Dokumentace UP Enterprise Architect

Dokumentace návrhu aplikace pomocí metodiky UP je přiložena ve složce *dokumentace_up*. Projekt byl vytvořen pomocí programu Enterprise Architect verze 9.0 a má název *dealer.eap*

Podklady pro optimalizaci SQL dotazů.

Veškeré podklady související s optimalizací vybraných SQL dotazů jsou přiloženy ve složce *optimalizace_dotazu*. Tato složka je dále rozdělena na dvě vnořené složky *mysql* a *oracle*. V těchto složkách jsou umístěny následující soubory:

- *databaze_mereni_testovaciho_scenare.xlsx* – soubor obsahuje zaznamenané hodnoty měření testovacího scénáře pro danou databázi.
- *databaze_statistika_rychlosti_SQL_dotazu.xlsx* – soubor obsahuje časové údaje naměřené při praktické optimalizaci dotazu.

V každé ze složek je umístěna sada vnořených složek, které odpovídají jednotlivým optimalizovaným dotazům. Obsahem složky *dotaz_x* je soubor ve tvaru *SQL_dotaz_x.sql*, který obsahuje znění původního a optimalizovaného dotazu. Dále je zde umístěna sada pdf

dokumentů, ve kterých jsou zaznamenány kompletní exekuční plány a statistiky získané z používaných programů.

Statistiky rychlosti

Tato složka obsahuje podklady, které byly využity pro porovnání rychlosti obou databázových platforem. Ve složce jsou uvedeny následující soubory.

- *testovaci_scenar.xlsx* – soubor obsahuje postup kroků prováděných při vykonávání testovacích scénářů.
- *statistika_rychlosti_neoptimalizovano.xlsx* – soubor obsahuje vyhodnocení statistiky rychlosti databázových dotazů před provedení optimalizace.
- *statistika_rychlosti_optimalizovano.xlsx* – soubor obsahuje vyhodnocení statistiky rychlosti databázových dotazů po provedení optimalizace.

Konfigurační soubory databázových serverů

Konfigurační soubory databázových serverů jsou přiloženy ve složce *konfigurace_databaze*, respektive ve vnořených složkách *oracle* a *mysql*.

Ukázkový tiskový výstup nabídky

Ve složce *tiskovy_vystup* je přiložen pdf dokument s ukázkou tiskového výstupu nabídky.

Zdrojový kód aplikace

Kompletní projekt pro IDE Netbeans 7.2.1 je umístěn ve složce *zdrojovy_kod_aplikace*. Aplikace byla vyvíjena pod pracovním označením dealer, tudíž hlavní složka projektu nese tento název.

Uživatelská příručka

Uživatelská příručka je umístěna ve složce *uzivatelska_prirucka*.

Text diplomové práce

Text této diplomové práce je umístěn ve složce *dokumentace*.