UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Knihovní systém implementovaný v jazyce C++ a MySQL

Pavel Michalíček

Bakalářská práce
2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Pavel MICHALÍČEK**
Osobní číslo:	**I07717**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Knihovní systém implemetovaný v jazyce C++ a MySQL**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Teoretická část:
- návrh databáze
- vyřešení ochrany před SQL Injection
- rozvržení obslužného programu do dvou částí - hlavní program (určený pro zaměstnance knihovny), klientský program (určený pro zákazníky knihovny)

Praktická část:
Cílem této práce je návrh a realizace systému určeného pro knihovny. Práce bude zaměřena na dvě části. První část bude realizace vhodné relační databáze MySQL pro knihovní systém a druhá část se bude týkat vytvoření vhodného obslužného programu v jazyce C++. Aplikace bude obsahovat evidenci knih, zákazníků a pracovníků. Dále bude zobrazovat aktuality knihovny.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

Základní:

(1.) ŠIMŮNEK Milan. SQL kompletní kapesní průvodce. , 1999.246 s. ISBN 80-7169-692-7.

(2.) KENT, Jeff. C++ bez předchozích znalostí: Průvodce pro samouky. (s.l.): Computer Press, 2009. 256 s. ISBN 978-80-251-2411-6.

Doporučená:

(3.) KERNIGHAN, Brian, RITCHIE, Denis M.. The C Programming Language., 1978. 542 s. ISBN 80-251-0897-X.

(4.) PRATA, Stephen. Mistrovství v C++. 3. aktualizované vydání (s.l.): Computer Press, 2007. 1120 s. ISBN 978-80-251-1749-1.

(5.) http://www.programujte.com

(6.) http://www.linuxsoft.cz

(7.) http://www.builder.cz

Vedoucí bakalářské práce: **Ing. Jaroslav Štroch**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. ledna 2010**

Termín odevzdání bakalářské práce: **14. května 2010**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2010

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 10. 5. 2010 Pavel Michalíček

Poděkování

Tímto bych chtěl poděkovat vedoucímu mojí bakalářské práce panu ing, Jaroslavu Štrochovi za jeho rady a připomínky při tvorbě bakalářské práce. Dále bych chtěl poděkovat všem profesorům, kteří mě vyučovali za jejich kvalitní hodiny.

Anotace

Knihovní systém je aplikace určená pro knihovny a slouží ke spravování všech pro knihovnu důležitých informací. Systém by měl nabízet kompletní přehled o knihách, zákaznících a pracovnících knihovny a měl by umožňovat kompletní správu uvedených objektů. Tento systém by však měl být pokud možno co nejjednodušší a jeho ovládání by mělo být intuitivní, aby jeho obsluha nebyla náročná na pochopení a naučení.

Klíčová slova

Qt, jazyk C++, knihovna, kniha, knihovní systém

Title

Library systém implemented by C++ langure and MySQL.

Annotation

Library system is an application designate for libraries and used for management all important information for library. System should offer full view about books, clients and library workers and system should make possible full management of these objects. This system should by still easy and intuitive for control, and its attendance should be modest and easy to learn.

Keywords

Qt, C++ language, library, book, library system

Obsah

Seznam zkratek

API	Application programming interface
BCNF	Boyce-Coddova normalní forma
CPU	Central processing unit
DCL	Data control language
DDL	Data definition language
DML	Data manipulation language
ERD	Entity-relationship diagram
FK	Foreign key
GPL	General public license
IP	Internet Protocol
ISBN	International standard book number
KDE	K desktop environment
MD5	Message-digest algorithm
MS	Microsoft
NF	Normální forma
PK	Primary key
SHA	Secure hash algorithm
SQL	Structured query language
TCP	Transmission control protocol
UML	Unified modeling language
XML	Extensible markup language

Seznam obrázků

Seznam tabulek

1 Úvodní informace

Vytvořený knihovní systém byl naprogramován v jazyce C++. Veškerá data jsou ukládána do databázového systému MySQL. Při vývoji aplikace byl použit MySQL server 5.1, který byl spravován pomocí programu EMS SQL Manager 2010 Lite for MySQL. Programová část byla vytvořena ve vývojovém prostředí Microsoft Visual Studio 2008. Pro tvorbu grafického uživatelského rozhraní byla použita nadstavbová knihovna jazyka C++ s názvem Qt, která byla integrována do Visual Studia. Výsledná aplikace podporuje ISBN-10 a ISBN-13 a je určena spíše pro menší knihovny. Při vývoji byl kladen důraz na správné přidělení práv k jednotlivým uživatelským rolím tak, aby každá role měla dostupné všechny možnosti pro ni určené. Tento knihovní systém je koncipován jako celek, byl by tedy nabízen se zřízením MySQL serveru, popřípadě pouze databáze a s možností menších úprav dle přání zákazníka.

Druhá kapitola je věnována relační databázi. Jsou zde popsány základní principy a důležité pojmy spjaté s relačními databázemi. Dále jsou vysvětleny základy zabezpečení databází a způsoby obrany před útokem na databázi. Poslední částí kapitoly je představení databázového systému MySQL.

Ve třetí kapitole je popsán Qt framework. Uvedeny jsou informace o historii, o způsobu instalace Qt knihovny do Visual Studia a je zde uveden i základní popis frameworku a Qt.

Kapitola číslo čtyři obsahuje informace o tom, jakým způsobem byli navrženy obslužné programy, společně s informacemi o jednotlivých uživatelských rolích a funkcích, které mají zpřístupněny.

Další kapitola je věnována realizaci relační databáze. Popsány jsou jednotlivé tabulky a další databázové objekty, které jsou důležité pro fungování databáze. Uveden je i ERD.

Šestá kapitola se zabývá propojením jazyka C++ a databázového systému MySQL. V této kapitole je vysvětlena vytvořená třída MYSQL_CL, společně s popisem nejdůležitějších metod, které třída obsahuje.

V sedmé kapitole jsou uvedeny UML diagramy, které vyjadřují architektury obou aplikací. Mezi uvedené diagramy patří Class diagram, Use case diagram a Activity diagram.

V předposlední kapitole jsou popsány ukázky zdrojových kódů, které byli použity při tvorbě aplikací. Je zde například popsáno, jakým způsobem se pracuje s databází, jak se pracuje s českými znaky apod.

Poslední kapitola je závěrečné shrnutí celé práce.

2 Relační databáze

2.1 Co to je databáze

Databáze je určitá uspořádaná množina dat, uložena na paměťovém mediu. Vznik databází můžeme „trochu s nadhledem“ tradovat až k samotnému vzniku uchovávání informací jako byly třeba nástěnné malby. Již ty byly jakousi „databází“, která uchovává informace o tehdejší kultuře. První databáze, v pravém slova smyslu, začaly vznikat až v období, kdy se rozšířilo psaní a čtení. Napsané spisy se uchovávaly v různých knihovnách a archívech, kde byly uloženy podle nějakého klíče, aby byly snadno dohledatelné. Vytvořily tedy uspořádanou množinu.

Přímými předchůdci databází byly kartotéky. Fungovaly na stejném principu jako dnešní elektronické databáze s tím rozdílem, že jejich obsluhu zajišťoval člověk. Dalším stupněm vývoje byly mechanické databáze, kde obsluhu zajišťoval stroj. Na dalším vývoji databází měl velký vliv překotný rozvoj počítačů ve 20. století, jenž umožnil jejich rozšíření a dobrou dostupnost. Logicky se tedy databáze přesunuly do elektronické podoby, jak je známe dnes.

Podle způsobu jakým jsou data v databázi uložena a dle vazeb mezi nimi můžeme databáze rozdělit do následujících skupin:

- objektově relační databáze.
- objektová databáze.
- síťová databáze.
- hierarchická databáze.
- relační databáze.

Dále se budeme bavit o relační databázi, která je založena na relačním modelu. Základem relační databáze jsou tabulky, řádky tabulek, které chápeme jako záznamy a eventuelně některé sloupce, které v sobě uchovávají informace o relacích mezi jednotlivými záznamy.

Každý atribut v tabulce má přesně definovaný datový typ – doménu. Jednotlivé záznamy jsou pak uloženy v řádcích tabulky. Každý řádek v tabulce tedy odpovídá právě jednomu záznamu.

2.2 Klíče

Dalším důležitým prvkem v relační databázi jsou klíče. Slouží nám pro jednoznačnou identifikaci záznamu v databázi. Rozdělujeme je na dva druhy.

2.2.1 Primární klíč (Primary key)

Primárním klíčem v tabulce může být jeden sloupec, nebo skupina více sloupců, které však dohromady musí tvořit jednoznačný identifikátor. V každém sloupci, ve kterém jsou uloženy primární klíče, je několik podmínek, které musí být splněny. Hodnoty ve

sloupci primárního klíče musí být unikátní, aby bylo možno každý záznam v tabulce jednoznačně identifikovat. Druhou podmínkou je, že hodnota primárního klíče musí být nenulová, tzn. nesmí v něm být uložena hodnota NULL.

Dnes se většinou jako primární klíče používají automaticky generované identifikátory. Můžeme použít i jiný jednoznačný údaj jako je například rodné číslo, číslo občanského průkazu nebo ISBN knihy. V tuto chvíli ovšem může vzniknout problém. Tyto primární klíče nebyly generovány naší databází a i když by se to nemělo stát, mohla by nastat situace, kdy například dva lidé mají stejné rodné číslo a v databázi by vznikly dva záznamy se stejným jednoznačným identifikátorem, což je pro správné fungování databáze nepřípustné.

2.2.2 Cizí klíč (Foreign key)

Jedná se o sloupec nebo skupinu sloupců, která slouží k vyjádření vztahů, relací, mezi jednotlivými tabulkami. Pomocí cizího klíče můžeme tedy navzájem k sobě přiřazovat záznamy z jednotlivých tabulek. Pokud se v tabulce A shoduje hodnota cizího klíče s hodnotou primárního klíče v tabulce B je daný řádek z tabulek A dále rozvinut o údaje z daného řádku tabulky B.

Pomocí cizího klíče můžeme dále definovat akce, které nastanou, pokud dojde ke změně nebo smazání dat v cizí tabulce. Pokud tedy dojde například ke smazání záznamu z tabulky, který má určitou hodnotu primárního klíče, může dojít ke smazání záznamu v cizí tabulce, který má stejnou hodnotu ve sloupci cizího klíče. Cizí klíč je tedy mechanismus pro udržení referenční integrity databáze.

2.3 Integrita databáze

Integrita databáze znamená, že data v ní uložená jsou validní, tzn. splňují definovaná pravidla. Do databáze jdou vložit pouze ta data, která splňují podmínky (například musí splňovat pro daný sloupec daný datový typ, data ve sloupci musí být unikátní nebo musí splňovat další pravidla). Integritu databáze nám zajišťují integritní omezení. Jsou to mechanismy, které zabraňují vložení nesprávných dat nebo zabraňují poškození dat při manipulaci s nimi. Existují následující čtyři druhy integritních omezení:

- Referenční integritní omezení – zabývá se vztahem dvou tabulek, kde jejich relace je určena vazbou primárního a cizího klíče.
- Aktivní referenční integrita – je to soubor činností, které server provádí, pokud dojde k porušení integrity dat.
- Doménové integritní omezení – zajišťuje dodržování datových typů nadefinovaných u jednotlivých sloupců tabulky.
- Entitní integritní omezení – je to povinné integritní omezení, které hlídá úplnost primárního klíče, zda jsou zadány data pro všechny sloupce, které tvoří primární klíč a hlídá, aby nedošlo ke vzniku duplicitního primárního klíče.

2.4 Normální formy

Normalizace je proces, při němž dochází ke zjednodušování a optimalizování navržených struktur databázových tabulek. Účelem normalizace je návrh tabulek takovým způsobem, aby obsahovaly co nejmenší počet nadbytečných dat a zlepšila se efektivita ukládání dat. Pro ohodnocení návrhu struktury se používají tzv. normální formy a platí, že v čím vyšší normální formě tabulka je, tím je její návrh kvalitnější. Pro jednotlivé normální formy platí následující pravidla:

- Nultá normální forma (0NF) – alespoň jeden sloupec tabulky může obsahovat více jak jednu hodnotu.

Student	Adresa školy
Karel Plíšek	Nermuťova 123, Pardubice
Tomáš Ječný	Holcova 565, Chrudim
Jan Bezděk	Gulášová 909, Pardubice
Daniel Plíživý	U skály 2, Holice

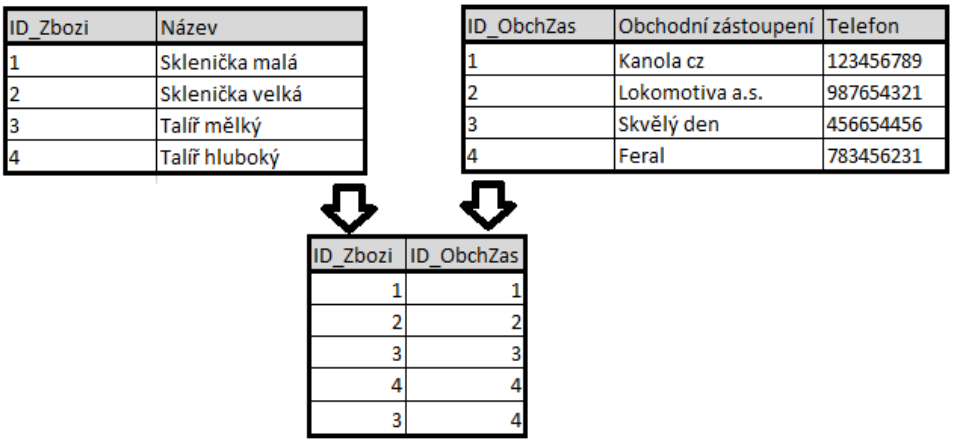
Obrázek 1 – Příklad tabulky v 0NF

- První normální forma (1NF) – v každém sloupci tabulky může být maximálně jedna hodnota, která nejde již dále rozdělit, tzn. je atomická.

Student jméno	Student příjmení	Škola město	Škola ulice
Karel	Plíšek	Pardubice	Nermuťova 123
Tomáš	Ječný	Chrudim	Holcova 565
Jan	Bezděk	Pardubice	Gulášová 909
Daniel	Plíživý	Holice	U skály 2

Obrázek 2 – Příklad tabulky v 1NF

- Druhá normální forma (2NF) – tabulka je ve druhé normální formě, pokud je v první normální formě, obsahuje jednoznačný identifikátor pro každý řádek a ostatní atributy v řádku jsou funkcí identifikátoru (jsou na něm závislé).



ID_Zbozi	Název
1	Sklenička malá
2	Sklenička velká
3	Talíř mělký
4	Talíř hluboký

ID_ObchZas	Obchodní zástoupení	Telefon
1	Kanola cz	123456789
2	Lokomotiva a.s.	987654321
3	Skvělý den	456654456
4	Feral	783456231

ID_Zbozi	ID_ObchZas
1	1
2	2
3	3
4	4
3	4

Obrázek 3 – Příklad tabulky ve 2NF

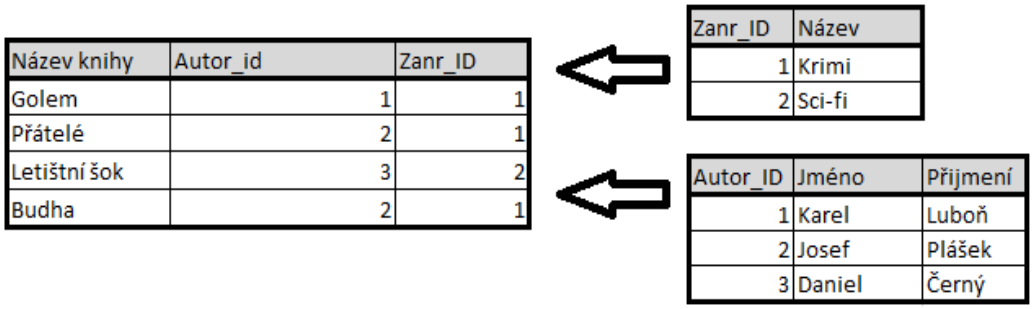
- Třetí normální forma (3NF) – tabulka je ve třetí normální formě, pokud je ve druhé normální formě a zároveň neexistuje jediná závislost mezi neklíčovými prvky.

ID_ObchZas	Obchodní zástoupení	Telefon
1	Kanola cz	123456789
2	Lokomotiva a.s.	987654321

ID_ObchZas	Název
1	Sklenička malá
2	Sklenička velká
2	Talíř mělký
2	Talíř hluboký

Obrázek 4 – Příklad tabulky ve 3NF

- Čtvrtá normální forma (4NF) – tabulka je ve čtvrté normální formě, je-li ve třetí normální formě a zároveň sloupce v ní obsažené popisují pouze jeden fakt (jednu závislost).



Název knihy	Autor_id	Zanr_ID
Golem	1	1
Přátelé	2	1
Letištní šok	3	2
Budha	2	1

Zanr_ID	Název
1	Krimi
2	Sci-fi

Autor_ID	Jméno	Přijmení
1	Karel	Luboň
2	Josef	Plášek
3	Daniel	Černý

Obrázek 5 – Příklad tabulky ve 4NF

- Pátá normální forma (5NF) – tabulka je v páté normální formě, pokud by se přidáním jediného sloupce rozpadla na více tabulek.
- Boyce-Coddova normální forma (BCNF) – poslední prakticky užívaná forma. Tabulka je v BCNF, právě když pro dvě množiny atributů X, Y platí: X->Y a současně Y není podmnožinou X, pak množina X obsahuje primární klíč tabulky. Ve většině případů dosáhneme BCNF pokud jsme dobře postupovali při tvorbě tabulek, aby splňovali 1NF, 2NF a 3NF.

2.5 Vztahy mezi tabulkami

2.5.1 Kardinalita vztahu

Kardinalita vztahu mezi dvěma tabulkami vyjadřuje počet řádků z jedné tabulky, které mohou vstoupit do vztahu s druhou tabulkou. Máme tři typy vztahů:

- Vztah 1:1 – jednomu řádku v první tabulce odpovídá právě jeden záznam ve druhé tabulce. Tento vztah se běžně nevyskytuje, protože většinou neexistuje důvod, proč by data nemohla být uložena v jedné tabulce. Ale existuje několik výjimek, jako například situace, kdy je právě jeden hlavní sudí přiřazen k právě jednomu probíhajícímu fotbalovému utkání.

Obrázek 6 – Značka vztahu tabulek 1:1

- Vztah 1:M – vyjadřuje situaci, kdy jednomu řádku v první tabulce odpovídá více řádků druhé tabulky a naopak. Tento vztah vzniká například mezi tabulkou fotbalisti a tabulkou fotbalové kluby. Každý fotbalista patří maximálně jednomu klubu a každý klub vlastní více fotbalistů.

Obrázek 7 – Značka vztahu tabulek 1:M

- Vztah M:N – při tomto vztahu náleží k několika záznamům z první tabulky několik záznamů z tabulky druhé. V reálném světě je tento vztah nejčastější. Je to situace, kdy například lidé nakupují ve více obchodech a každý obchod má více zákazníků. Tento vztah však není možné v relačních databázích modelovat přímo. Vyžívá se proto vazební tabulky, která se vloží mezi tabulky, které mají vztah M:N. Vztah mezi vazební tabulkou a tabulkami, které spojuje je potom 1:M.

Obrázek 8 – Značka vztahu tabulek M:N

2.5.2 Parcialita vztahu

Parcialita vztahu mezi dvěma tabulkami vyjadřuje skutečnost, zda vztah musí existovat či nemusí.

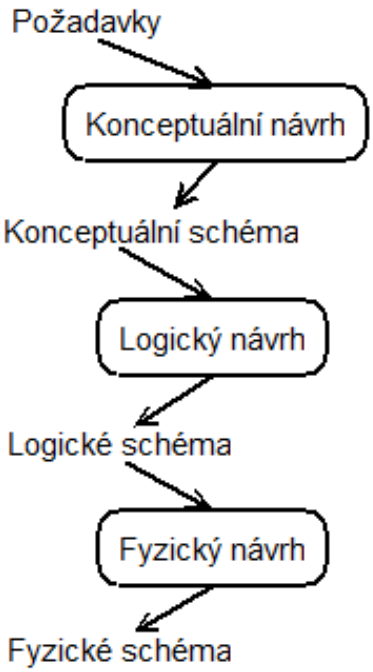
- Jednostranně parciální vztah – záznam z první musí být přiřazen k záznamu z druhé tabulky, avšak záznam z druhé tabulky nemusí náležet k žádnému záznamu z první tabulky. Například každá vyrobená bota musí náležet k nějakému výrobci. Výrobce ale nemusí žádnou botu vyrábět.
- Oboustranně parciální vztah – záznam z první tabulky nemusí být přiřazen k žádnému záznamu ze druhé tabulky a záznam ze druhé tabulky nemusí být přiřazen k žádnému záznamu z první tabulky. Například člověk může a nemusí mít zaměstnavatele a zaměstnavatel může a nemusí mít žádné zaměstnance.

Obrázek 9 – Značení parciality a kardinality (jedna strana)

2.5.3 Unární vztah

Unární vztah je speciální případ vztahu dvou tabulek. Jedná se o situaci, kdy je tabulka spojena sama se sebou. Taková tabulka potom obsahuje cizí klíč, který se odkazuje na primární klíč ve stejné tabulce.

2.6 Fáze návrhu databáze



Obrázek 10 – Fáze návrhu databáze

V první fázi návrhu dochází ke komunikaci mezi zákazníkem a výrobcem databáze. Musí ujasnit všechny požadavky, které má zákazník vůči databázi. Ze získaných informací potom vznikne konceptuální model. Tento model zobrazuje data pouze na abstraktní úrovni a nezávisle na jejich fyzickém uložení. Model je zobrazen pomocí konceptuálního schéma. Nejčastěji používaným konceptuálním datovým modelem je ER diagram.

Při další části návrhu databáze se zabýváme logikou databáze. K jednotlivým sloupcům tabulek přiřazujeme datové typy, vytváříme integritní omezení, vytváříme primární a cizí klíče a další. Z těchto údajů následně vytvoříme logické schéma databáze.

Poslední část při návrhu databáze je již fyzické vytvoření databáze na serveru.

2.7 Dotazovací jazyk SQL

SQL je zkratka anglických slov Structured Query Language (strukturovaný dotazovací jazyk). Jazyk SQL je standardizovaný dotazovací jazyk určený pro prácí s daty uloženými v relačních databázích. Pomocí SQL můžeme definovat strukturu databáze, manipulovat s daty, získávat data, sdílet data, řídit přístup a zajistit integritu dat. U většiny databázových systémů najdeme implementaci jazyka SQL dle standardů SQL2. Příkazy jazyka SQL dělíme do následujících:

- Příkazy pro definici dat (DDL) – pomocí těchto příkazů vytváříme strukturu databáze (tabulky, pohledy, …). Pomocí příkazů DDL můžeme již vytvořenou strukturu měnit. Mezi příkazy DDL patří CREATE, ALTER, DROP.

- Příkazy pro manipulaci s daty (DML) – skupina příkazů DML nám slouží pro práci s daty v databázi. S těmito dotazy můžeme data vybírat, měnit a vkládat. Mezi příkazy DML patří SELECT, INSERT, DELETE, UPDATE.
- Příkazy pro řízení dat (DCL) – poslední skupina příkazů je určena pro řízení provozu, nastavování přístupových práv a údržba transakcí. Mezi příkazy DCL patří GRANT, REVOKE, ALTER USER, DROP USER, COMMIT, ROLLBACK, SET TRANSACTION.
- Ostatní příkazy – do této skupiny patří příkazy, pomocí nichž se nastavují systémové parametry jako například kódování, způsob řazení dat a další. Tyto příkazy však nejsou standardizovány a jsou různé pro každý databázový systém.

2.8 Druhy databázových systémů

V dnešní době existuje několik databázových systémů. Některé jsou volně dostupné, za jiné se naopak musí platit. Mezi nejznámější a nejpoužívanější databázové systémy patří Oracle, MySQL, Firebird a PostgreSQL.

2.8.1 Oracle

Oracle je moderní, multiplatformní, komerční databázový systém s vysokým výkonem, velice pokročilým zpracováním dat a snadnou škálovatelností. Aktuální verzí je 11g, která umožňuje používat standardní dotazovací jazyk SQL dle normy SQL92, umožňuje využívat proprietární firemní rozšíření Oracle (např. pro hierarchické dotazy). Dále podporuje jazyk PL/SQL. Tento jazyk jakýmsi rozšířením jazyka SQL. Je v něm možno definovat uložené procedury, uživatelské funkce, triggery a programové balíky.

2.8.2 MySQL

MySQL je asi nejpoužívanějším databázovým systémem. Je to multiplatformní systém, který je šířen jako open source a je možné ho tedy bezplatně používat. Tento systém blíže popíšeme dále. Vývojářem je Oracle Corporation.

2.8.3 Firebird

Firebird je opět multiplatformní databázový systém. Je šířen pod licencemi odvozenými od Mozilla Public License. Mezi výhody tohoto systému patří především velká podpora ze strany vývojových nástrojů a to především Delphi či Java. Tento systém se hodí pro střední firmy. Jeho výhodou je jednoduché zálohování, slušná podpora PL/SQL a možnost tvorby vlastních uživatelsky definovaných funkcí.

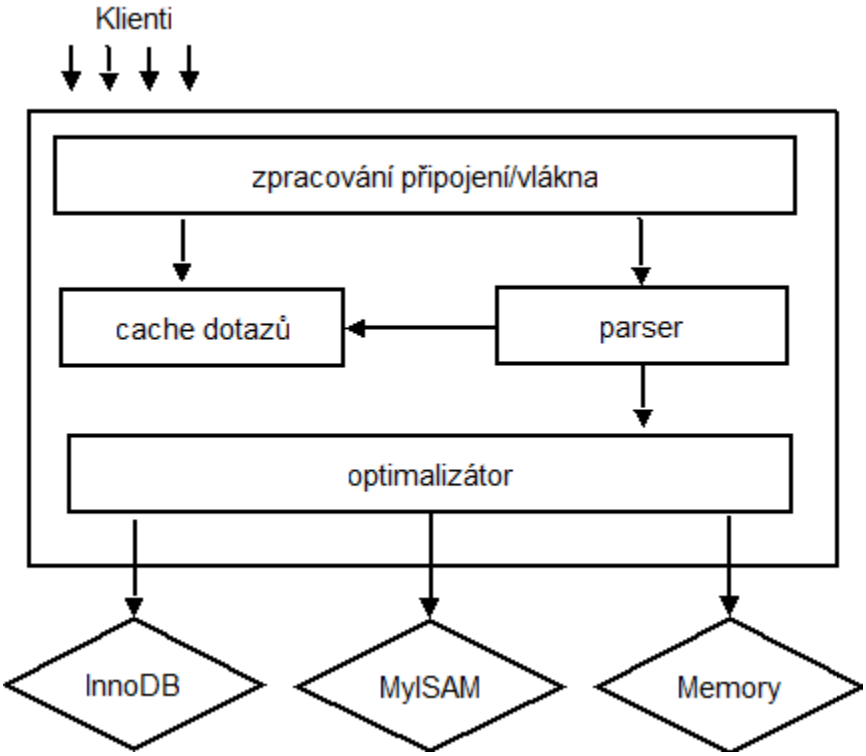
2.8.4 PostgreSQL

PostgreSQL je objektově-relační databázový systém. Je šířen jako open source a tudíž je volně dostupný. Sytém není vlastněn jednou firmou, ale na jeho vývoji se podílí široká komunita vývojářů a firem. Tento systém je primárně vyvíjen pro linux, ale existují i balíčky pro platformu win32.

2.9 Databázový systém MySQL

MySQL pracuje na principu klient/server. Klientská aplikace tedy může běžet na jiném počítači než serverová aplikace a každá aplikace může běžet i na jiném operačním sytému. Veškerá vzájemná komunikace probíhá pomoci protokolu TCP/IP. MySQL je od začátku vyvíjeno pro rychlost a pro některé vlastnosti má jednodušší než jiné databázové systémy.

2.9.1 Architektura MySQL serveru



Obrázek 11 – Architektura MySQL serveru

První vrstva, která je nahoře se týká komunikace po sítí. Tato vrstva není jedinečná pro MySQL a slouží pro udržování a nastavování připojení k serveru. Ve druhé vrstvě se nachází většina životně důležitých funkcí serveru. Obsahuje parser pro rozbor dotazů, analýzu a optimalizaci. V poslední vrstvě se nacházejí úložné enginy. Jejich úlohou je ukládání a získávání všech dat, které jsou na serveru uloženy. Komunikace s úložnými enginy probíhá pomocí API. Toto rozhraní spojuje rozdíly v komunikaci s jednotlivými enginy a tato komunikace se stává transparentní. Jednotlivé úložné enginy mezi sebou nekomunikují, pouze odpovídají na požadavky.

2.9.2 Správa připojení uživatelů

Autentizace uživatele spočívá na kontrole uživatelského jména, hesla a na hostiteli, odkud se uživatel připojuje. Jakmile se klient připojí, dostane uvnitř serverového procesu vlákno. Každé vlákno sídlí na jednom CPU nebo na jednom jádru a všechny dotazy se vykonávají uvnitř tohoto vlákna. Server si jednotlivá vlákna ukládá do cache, ty se pak nemusejí při každém připojení a odpojení znova vytvářet a likvidovat.

2.9.3 Optimalizace a vykonávání

Při parsování parser vytvoří stromovou strukturu (parse tree), na kterou poté aplikuje optimalizace. Při optimalizování může dojít k přepsání dotazu, určí se pořadí čtení tabulek a rozhodne se o tom, které indexy budou použity atd. Optimalizování může programátor ovlivnit pomocí klíčových slov (hints). Optimalizátor se nestará o to, který úložný engine používá tabulka. Úložný engine ovšem ovlivňuje to, jakým způsobem bude optimalizace probíhat. Ještě předtím, než se optimalizace začne provádět, se server podívá do query cache, kde jsou uloženy selecty a jejich výsledky. Pokud je nalezena shoda, ihned se vrací výsledky z query cache.

2.9.4 Úložné enginy

Každá databáze je uložena do podadresáře datového adresáře MySQL serveru na odkladovém souborovém systému. Po vytvoření tabulky se její definice uloží do souboru *.frm* se stejným názvem jako je název tabulky. MySQL používá na ukládání definic souborový systém, proto je rozlišování velikosti písem různé dle platformy. Na platformě windows se velikost písem nerozlišuje, kdežto na unixových systémech se velikost písem rozlišuje. Jednotlivé úložné enginy ukládají tabulky jinak, definici tabulky však zpracovává server. Úložné enginy jsou moduly, je tedy možné je do databázového systému doinstalovat. Každý engine se líší svými možnostmi a způsobem ukládání dat. Zde je výpis několika úložných enginů, které MySQL podporuje:

- MyISQM.
- InnoDB.
- CSV.
- MERGE.
- BerkeleyDB (BDB).
- MEMORY.

2.10 SQL injection

SQL injection je technika, pomocí níž je hacker schopen napadnout databázovou část aplikace. Jak už název napovídá (injection = vsunutí), jedná se v podstatě o vsunutí jiného, nebezpečného kódu přes neošetřený vstup, kterým se pozmění prováděný dotaz. SQL injection je spojeno hlavně s webovými stránkami. Na internetu se pohybuje spousta lidí a stránky na internetu jsou tedy přístupné široké veřejnosti, a tudíž jsou útoky pomocí SQL injection velice časté. Tato technika však lze použít i v případě aplikace, která komunikuje s databází umístěnou někde na serveru a je určena pro potřeby veřejnosti.

2.10.1 Důsledky SQL injection

Důsledky útoku na databázi pomocí SQL injection se mohou podstatně lišit. V nejlepším případě se útočníkovi podaří dostat do míst, kam by neměl a neprovede tam žádnou škodu. V dalším případě bude chopen vyčíst z databáze údaje, jako jsou například hesla, přihlašovací údaje a další citlivé informace. V případě přečtení hesel, se většinou jedná pouze hashe, takže útočník nemá přímo heslo, ale mohlo by se mu podařit ho

dekódovat. V nejhorších případech útočník změní data v databázi. Některé položky přepíše nebo smaže a může úplně změnit obsah aplikace, která nad danou databází běží. Aplikaci může útočník vyřadit z provozu úplně a to tím, že smaže některé tabulky nebo i celou databázi.

2.10.2 Ukázka útoku

Pokud naše aplikace vypisuje chyby, může si útočník snadno pomocí určitých pokusů o vyvolání chyb, zjistit názvy tabulek v naší databázi.

Základní dotaz, se kterým budeme dále pracovat:

```
SELECT * FROM workers WHERE name LIKE ‘ ”+name_in+” ’;
```

Útočník může jako login zadat například:

```
x‘ or ‘y‘ = ‘y
```

Program dotaz doplní (viz. níže) a následně jej odešle ke zpracování do databáze. Nebezpečí spočívá v tom, že když nebude žádný atribut ve sloupci name odpovídat hodnotě x, dotaz stejně bude funkční a vypíše nám celou tabulku workers, protože podmínka y = y bude platit pro každý záznam.

```
SELECT * FROM workers WHERE name LIKE ‘ x‘ or ‘y‘ = ‘y ‘
```

Pro SQL injection lze samozřejmě použít kterýkoliv příkaz. Pokud například bude do pole pro jméno zadáno:

```
x‘ ; DROP TABLE workers; --
```

To způsobí, že výsledný dotaz bude vypadat takto:

```
SELECT * FROM workers WHERE name LIKE ‘x‘ ; DROP TABLE workers; --’;
```

Tento dotaz se zpracuje tak, že se nejprve zpracuje select a po něm se provede příkaz drop table, který bude mít za následek smazání celé tabulky workers. Poslední apostrof a středník nebude brán v potaz, protože je pomocí sekvence – opoznámkován. Takových průniků může nastat celá řada a díky klauzulím union a join se může útočník dostat i k dalším tabulkám.

2.10.3 Obrana před SQL injection

Na straně databáze

Útok můžeme znemožnit nebo alespoň velice ztížit správným nastavením přístupových práv. Ne každý uživatel potřebuje pro svoji funkčnost přístup ke všem tabulkám a práva na provádění všech příkazů. Pro uživatele tedy nastavíme práva pouze na nutné tabulky a pouze na nutné příkazy.

Na straně aplikace

Obrana na straně aplikace spočívá v kontrole vstupních dat. Nejběžnějším způsobem je kontrola nebezpečných znaků a její následné převedení na znaky bezpečné. Pomocí tzv. escapování vložíme před nebezpečné znaky znak /, čímž způsobíme to, že databáze následný znak nebere jako příkaz, ale jako běžný znak. Dále můžeme nastavit maximální délku vkládaného textu, tím zamezíme vložení dlouhého nebezpečného kódu.

2.11 Zabezpečení dat v databázi

Všechna citlivá data, která se ukládají do databáze, by měla být nějakým způsobem chráněna, kódována. MySQL má pro tuto situaci několik funkcí, které nám ze vstupních dat vytvářejí hash. Jsou jimi funkce Password, MD5 a SHA.

2.11.1 Co to je hash, hashovaní funkce

Hash je výstup hashovací funkce, jindy nazýván také jako otisk, výtah nebo miniatura. Hashovací funkce je algoritmus, který převádí vstupní data do relativně malého čísla. Hashovací funkce má několik základních vlastností:

- I při malém rozdílu vstupních dat dostaneme velice rozdílný hash.
- Z jakkoliv velkého vstupního řetězce dostaneme pokaždé stejně dlouhý hash.
- Prakticky nemožné z hashe dostat původní vstup.
- Je vysoce nepravděpodobné, že by dva různé vstupy měli stejný hash.

Z popisu je jasné, že může dojít ke kolizi. To znamená, že pro dvojici vstupních dat (x, y), kde x je různé od y, platí, že h(x) = h(y). Kolize jsou nežádoucí, ale v praxi se jim nevyhneme, protože počet možných vstupních zpráv, je vyšší než počet výstupů. Kolize se dají pouze eliminovat volbou vhodné funkce.

2.11.2 Password(), funkce MySQL

Tato funkce je v MySQL použita pro ukládání hesel v tabulce users a byla použita i v mém případě pro ukládání hesel uživatelů knihovny. Do verze MySQL 4.1, měl hash velikost 16B. Vyšší verze mají velikost výstupního hashe 41B. Rozdíl mezi těmito dvěma hashi je dále takový, že nová verze funkce ukládá na začátek hashe znak *, aby bylo možno poznat, kterou verzí funkce Password bylo heslo hashováno.

3 Qt framework

3.1 Co je to framework

Framework je softwarová struktura, která slouží jako podpora při programování a vyvíjení aplikací. Dává aplikaci vzhled a funkcionalitu, tím že přidává aplikaci grafické rozhraní. Tím značnou mírou zjednodušuje používání počítače a v dnešní době i různých embedded zařízení jako jsou například smartphony.

3.2 Qt

Qt je jedna z nejoblíbenějších knihoven pro tvorbu graficky uživatelského rozhraní. Je to knihovna jazyka C++, ale existuje i pro jiné programovací jazyky jako například Python, Ruby, C, Perl, atd. Podporuje správu vláken, přístup k souborům, XML a také SQL.

3.3 Rozdělení Qt

Qt knihovna je multiplatformní a je vyvíjena pro platformy Windows CE, MS Windows, Mac OS X a X Window System. Dostupné jsou tři komerční edice, které je nutné zakoupit:

- Qt console – vývoj konzolových aplikací.
- Qt desktop light – základní vývoj grafických aplikací (bez podpory sítí, databází).
- Qt desktop – kompletní vývoj grafických aplikací.

Dále je možné získat Qt knihovnu pod licencí GPL. Tato edice je určena pro Open source vývoj:

- Qt open source edition – kompletní vývoj grafických aplikací (pod Open source licencí).

3.4 Historie

Qt knihovna se objevila v roce 1991 a za jejím zrodem stál vývojář Haavard Nord a Eirik Chambe-Eng, prezident společnosti Trolltech. První velkou událostí v historii Qt knihovny byl rok 1998, kdy byla zvolena jako hlavní pro implementaci grafického prostředí KDE. V nedávné době odkoupila Qt od Trolltechu společnost Nokia, která uvolnila edici pod licencí GPL. V současné době je aktuální verze 4 (4.5.2).

3.5 Instalace

Instalace Qt do počítače lze provézt mnoha způsoby. Asi nejjednodušší postup, je stáhnutí kompletního balíku pro vámi požadovaný systém z oficiálních stránek Qt (http://qt.nokia.com/downloads). Tento balík obsahuje všechny potřebné knihovny, program pro vytváření grafického rozhraní a různé vývojářské nástroje. Pokud nechceme

kompletní balík, máme možnost si stáhnout pouze námi požadované knihovny, které si pak musíme sami připojit do námi používaného vývojového prostředí.

Vzhledem k tomu, že jsem zvyklý používat při programování v jazyce C++ vývojové prostředí Microsoft Visual Studio 2008, zvolil sem si cestu doinstalování Qt do zmíněného vývojového prostředí. Nutné k tomu jsou instalační soubory. Prvním je qt-vsintegration-1.4.3, který nám zajistí integraci Qt do Visual Studia. Druhým je soubor qt-win-commercial-4.4.3-vs2008, kterým se nainstalují potřebné knihovny a nástroje do Visual Studia. Výsledkem je Qt, které je plně integrované do Visual Studia.

4 Rozvržení obslužného programu

V této kapitole budou rozebrány jednotlivé funkce, které bude obsahovat obslužný program pro zaměstnance a pro zákazníky. Důležité je, aby v obou obslužných programech byly dostupné všechny funkce pro plnohodnotné využívání programu a zároveň, aby nebyly přístupné ty funkce, které by mohly být nějakým zneužity.

4.1 Obslužný program pro zaměstnance

Obslužný program pro zaměstnance by měl mít funkce pro kompletní správu zákazníků, knih, výpůjček a ve správcovské části, též správu zaměstnanců.

V části, která je určena pro spravování zákazníků knihovny, jsou dostupné funkce pro přidávání zákazníků, odebírání zákazníků a funkce pro úpravu jejich osobních údajů. Dále je možnost vypsat si zákazníkovi půjčené knihy.

Ve správě knih jsou funkce pro přidání nové knihy a smazání knihy z databáze. Je umožněno vyhledávání v knihách, podle zadaných parametrů. U vyhledané knihy si můžeme zobrazit detailní informace knize. V zobrazeném okně je vypsaný seznam výtisků se jménem zákazníka, který má daný výtisk vypůjčený. Ke každé knize v databázi jsou evidovány výtisky. To znamená, že od každé knihy, můžeme mít několik kusů. Ke každé knize je možnost přidat libovolný počet výtisků a samozřejmě lze zvolený výtisk i smazat.

Při spravování výpůjček lze využít několik funkcí programu. V půjčkách lze vyhledávat podle zadaných kritériích. O každé vyhledané výpůjčce si můžeme zobrazit detailní informace, kde jsou uvedeny základní údaje společně s datem vypůjčení a datem vrácení. U každé výpůjčky máme možnost ji prodloužit o dalších 30 dní. Při zadávání výpůjčky do systému máme dvě možnosti, jak to učinit. První je, že výpůjčka bude do systému zadána na základě ID zákazníka a ID knihy. Druhou cestou je možnost potvrzení vypůjčení knihy, kterou si předem zákazník zamluvil pomocí zákaznické aplikace (viz. níže).

Pro lepší přehlednost při správě autorů a vydavatelství, je část programu, věnující se těmto prvkům, umístěna samostatně, nikoliv v části pro přidávání knihy. Je zde umožněno přidávat vydavatelství, přidávat autory a mazat vydavatelství a zároveň autory. Je zde umožněno spravovat i žánry. U vydavatelství je možnost si o vybraném vydavatelství vypsat bližší informace. Ve výpisu je možno upravit informace o vydavatelství a jsou zde vypsané knihy, které jsou evidovány v naší databázi a jsou od vybraného vydavatelství.

Poslední část určená pro běžné zaměstnance je modul pro přidávání aktualit. Zde je možno vkládat krátké aktuality, týkající se knihovny. Tyto aktuality jsou následně zobrazovány v zákaznické aplikaci.

Pokud je přihlášený uživatel správcem knihovny, má přístupný poslední modul, který určený pro správu zaměstnanců knihovny. Funkce v tomto modulu jsou v podstatě

stejné jako v modulu pro správu zákazníků. Opět je tu možnost přidání pracovníka, vypsání jeho osobních údajů, možnost změny osobních údajů a smazání pracovníka. Je zde navíc umožněno nastavit, zda je pracovník správcem nebo ne. Systém nedovoluje ze systému smazat posledního správce. Je tedy vyloučena možnost, že by v systému nezůstal žádný správce a nešlo by tedy spravovat pracovníky. V tomto modulu je nadále umožněno měnit přihlašovací hesla pro zaměstnance.

4.2 Obslužný program pro zákazníky

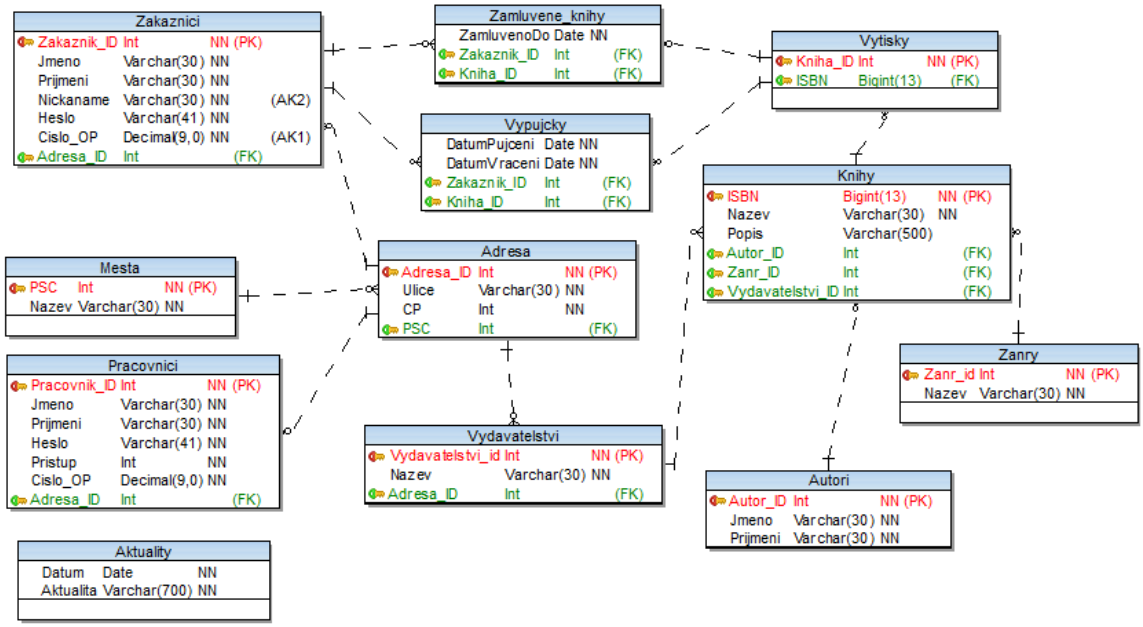
Obslužný program pro zákazníky nemusí zdaleka obsahovat takové množství funkcí jako obslužný program pro pracovníky knihovny. Základem je možnost, aby zákazník mohl vyhledávat v knihách, které knihovna nabízí.

V programu pro zákazníky lze pracovat jak v nepřihlášeném, tak v přihlášeném stavu. V nepřihlášeném stavu je umožněno uživateli vyhledávat knihy dle zadaných parametrů, a dále mu jsou zobrazeny aktuality z knihovny.

V přihlášeném stavu je pak přístupno několik funkcí navíc. Je zde možno zamluvit si vyhledanou knihu, pokud má kniha právě volný výtisk. Kniha se zamluví na 7 dní, během kterých je nutno si knihu vyzvednout, jinak se zamluvení zruší a kniha bude opět dostupná všem. Dále se přihlášenému zákazníkovi zobrazí jeho vypůjčené knihy, ke kterým je možnost si zobrazit bližší informace. Zákazník má taky možnost si v zákaznickém programu změnit svoje osobní údaje a přihlašovací údaje.

5 Realizace relační databáze

5.1 ERD



Obrázek 12 – ER diagram vytvořené databáze

5.2 Popis jednotlivých tabulek

Všechny tabulky byly uloženy pomocí úložného enginu InnoDB.

Tabulka Zakaznici uchovává informace o zákaznících knihovny. Ve sloupci Zakaznik_ID je zákazníkovo ID, které je primárním klíčem a slouží k identifikaci zákazníka v databázi. Obsahuje sloupec Adresa_id, jenž je cizím klíčem adresy. Sloupec Cislo_OP obsahuje číslo občanského průkazu daného zákazníka a slouží pro hlídání vzniku duplicitních zákazníků

Tabulka 1 – Tabulka Zakaznici

Klíč	Název sloupce	Datový typ	Not Null	Unique
PK	Zakaznik_ID	Integer	NN	U
	Jmeno	Varchar(30)	NN	
	Prijmeni	Varchar(30)	NN	
	Nickname	Varchar(30)	NN	U
	Heslo	Varchar(41)	NN	
	Cislo_OP	Decimal(9, 0)	NN	U
FK	Adresa_ID	Integer	NN	U

Tabulka Pracovnici obsahuje informace o zaměstnancích knihovny. Ve sloupci Pracovník_ID je uloženo ID pracovníka pro jeho identifikaci v databázi. Sloupec Cislo_OP opět slouží ke hlídání duplicity záznamů. Hodnota ve sloupci Pristup může nabývat dvou

hodnot. Hodnota „1“ znamená, že pracovník je zároveň správcem knihovny. Hodnota „2“ znamená, že pracovník má omezený přístup k funkcím programu. Tabulka obsahuje také cizí klíč adresy.

Tabulka 2 – Tabulka Pracovnici

Klíč	Název sloupce	Datový typ	Not Null	Unique
PK	Pracovnik_ID	Integer	NN	U
	Jmeno	Varchar(30)	NN	
	Prijmeni	Varchar(30)	NN	
	Heslo	Varchar(41)	NN	
	Pristup	Integer	NN	
	Cislo_OP	Decimal(9, 0)	NN	U
FK	Adresa_ID	Integer	NN	U

V tabulce Vydavatelstvi je uložen název vydavatelství, jedinečné číslo, ID, pro identifikaci v databázi a dále obsahuje cizí klíč adresy.

Tabulka 3 – Tabulka Vydavatelstvi

Klíč	Název sloupce	Datový typ	Not Null	Unique
PK	Vydavatelstvi_ID	Integer	NN	U
	Nazev	Varchar(30)	NN	
FK	Adresa_ID	Integer	NN	U

V následující tabulce, která se jmenuje Autori, jsou uloženy údaje o autorech. Je v ní dále uloženo ID autora pro jeho identifikaci v databázi.

Tabulka 4 – Tabulka Autori

Klíč	Název sloupce	Datový typ	Not Null	Unique
PK	Autor_ID	Integer	NN	U
	Jmeno	Varchar(30)	NN	
	Prijmeni	Varchar(30)	NN	

Tabulka Zanry slouží pro uložení žánrů. Obsahuje ID žánru pro jeho identifikaci v databázi a dále obsahuje název žánru.

Tabulka 5 – Tabulka Zanry

Klíč	Název sloupce	Datový typ	Not Null	Unique
PK	Zanr_ID	Integer	NN	U
	Nazev	Varchar(30)	NN	

V tabulce Mesta jsou uloženy názvy měst. Jako primární klíč je použito poštovní směrovací číslo, které má každá obec jedinečné.

Tabulka 6 – Tabulka Mesta

Klíč	Název sloupce	Datový typ	Not Null	Unique
PK	PSC	Integer	NN	U
	Nazev	Varchar(30)	NN	

Následující tabulka, s názvem Adresa, slouží pro uložení adres pro zákazníky, pracovníky a vydavatelství. Obsahuje ID pro jednoznačnou identifikaci adresy v databázi, název ulice, číslo popisné a cizí klíč, do tabulky Města, s názvem PSC.

Tabulka 7 – Tabulka Adresa

Klíč	Název sloupce	Datový typ	Not Null	Unique
PK	Adresa_ID	Integer	NN	U
	Ulice	Varchar(30)	NN	
	CP	Integer	NN	
FK	PSC	Integer	NN	U

Tabulka s názvem Knihy obsahuje údaje o knihách. Jako primární klíč bylo použito ISBN, které má každá kniha jedinečné. Dále je v tabulce uložen název, popis knihy a cizí klíče do tabulek Autor, Zanry a Vydavatelstvi.

Tabulka 8 – Tabulka Knihy

Klíč	Název sloupce	Datový typ	Not Null	Unique
PK	ISBN	Bigint(30)	NN	U
	Nazev	Varchar(30)		
	Popis	Varchar(500)		
FK	Autor_ID	Integer	NN	U
FK	Zanr_ID	Integer	NN	U
FK	Vydavatelstvi_ID	Integer	NN	U

Tabulka Vytisky slouží pro ukládání údajů o jednotlivých výtiscích od každé knihy. Je v ní uložen cizí klíč do tabulky Knihy (ISBN) a jako primární klíč je zde Kniha_ID, který slouží k identifikaci výtisku v databázi.

Tabulka 9 – Tabulka Vytisky

Klíč	Název sloupce	Datový typ	Not Null	Unique
PK	Kniha_ID	Integer	NN	U
FK	ISBN	Bigint (30)	NN	U

Vazební tabulka Vypujcky je určena pro uchovávání údajů o jednotlivých výpůjčkách. Je v ní uloženo datum půjčení knihy, datum vrácení knihy, dále pak cizí klíč

do tabulky Zakaznici, pro identifikaci zákazníka, a cizí klíč do tabulky Vytisky, pro identifikaci zapůjčené knihy.

Tabulka 10 – Tabulka Vypujcky

Klíč	Název sloupce	Datový typ	Not Null	Unique
	DatumPujceni	Date	NN	
	DatumVraceni	Date	NN	
FK	Zakaznik_ID	Integer	NN	U
FK	Kniha_ID	Integer	NN	U

Další vazební tabulkou je tabulka Zamluvene_knihy. Slouží k uchovávání dat o zamluvených knihách. Data uložená ve sloupcích Zkaznik_ID a Kniha_ID jsou cizí klíče do tabulek Zakaznici a Vytisky, které slouží pro identifikaci zákazníka a zamluvené knihy. Poslední sloupec v tabulce, s názvem ZamluvenoDo, je určen pro uložení data, do kdy je daný výtisk zamluven.

Tabulka 11 – Tabulka Zamluvene_knihy

Klíč	Název sloupce	Datový typ	Not Null	Unique
	ZamluvenoDo	Date	NN	
FK	Zakaznik_ID	Integer	NN	U
FK	Kniha_ID	Integer	NN	U

Poslední tabulkou v databázi je tabulka s názvem Aktuality. Tato tabulka není nijak propojena s ostatními tabulkami v databázi. Slouží pro ukládání krátkých aktualit knihovny. Je v ní uloženo datum, kdy byla aktualita vydána a obsah samotné aktuality, který je uložen ve sloupci Aktualita.

Tabulka 12 – Tabulka Aktuality

Klíč	Název sloupce	Datový typ	Not Null	Unique
	Datum	Date	NN	
	Aktualita	Varchar (700)	NN	

5.3 Popis použitých databázových objektů

5.3.1 Pohledy

Pohled v_VolnychKnih

První pohled se jmenuje v_VolnychKnih. Vrací v jednom sloupci počet volných výtisků ke každé knize a ve druhém je ISBN příslušné knihy. Pokud tedy kniha s určitým ISBN má nějaké dostupné výtosky, které je možné si zapůjčit (tzn. nejsou vypůjčeny ani zamluveny), vrací tento pohled ve výsledku u daného ISBN počet dostupných výtisků. Ve chvíli kdy kniha nemá dostupný výtisk je vrácena na příslušném řádku hodnota null.

```
create view v_VolnychKnih as
select
    `vytisky`.`ISBN` AS `isbn`,
    count(`vytisky`.`Kniha_ID`) AS `pocet`
  from
    `vytisky`
  where
    ((not(`vytisky`.`Kniha_ID` in (
  select
    `vypujcky`.`Kniha_ID` AS `kniha_id`
  from
    `vypujcky`))) and (not(`vytisky`.`Kniha_ID` in (
  select
    `zamluvene_knihy`.`Kniha_ID` AS `kniha_id`
  from
    `zamluvene_knihy`))))
  group by
    `vytisky`.`ISBN`;
```

Pohled v_PocetVytisku

Druhým pohledem v databázi je pohled se jménem v_PocetVytisku. Tento pohled, stejně jako předchozí, vrací ve výsledku dva sloupce. V prvním sloupci je ISBN knihy a ve druhém je celkový počet výtisků dané knihy, které jsou v knihovně. Pokud kniha nemá výtisk, je vrácena hodnota null, jinak je vrácena číselná hodnota počtu výtisků.

```
create view v_PocetVytisku as
select
    `vytisky`.`ISBN` AS `isbn`,
    count(`vytisky`.`Kniha_ID`) AS `pocet`
  from
    `vytisky`
  group by
    `vytisky`.`ISBN`;
```

5.3.2 Číslování ID

Většina tabulek v databázi obsahuje hodnotu primárního klíče v podobě nějakého uměle vytvořeného identifikátoru ID. Při vytváření nového záznamu jsou dvě možnosti jak inicializovat hodnotu ID. První možností je, že hodnotu ID si do záznamu vložíme sami. Toto řešení však není příliš vhodné, pokud používáme výše zmíněný uměle vytvořený identifikátor. V této chvíli je lepší, aby hodnota ID, byla vkládána do každého nově vloženého záznamu automaticky. V MySQL databázi toho dosáhneme připsáním klauzule AUTO_INCREMENT do příkazu na vytvoření tabulky. Příklad:

```
Creat table test (
  `radek` int(11) NOT NULL AUTO_INCREMENT
);
```

Seznam tabulek a sloupců, kde je použita funkce AUTO_INCREMENT:

Tabulka 13 – Seznam sloupců s funkcí AUTO_INCREMENT

Tabulka	Sloupec
Adresa	Adresa_ID
Autori	Autor_ID
Pracovnici	Pracovník_ID
Vydavatelstvi	Vydavatelstv_ID
Vytisky	Kniha_ID
Zakaznici	Zakaznik_ID
Zanry	Zanr_ID

5.3.3 Indexy

Index je pomocná datová struktura, která nám usnadňuje určení polohy záznamu na základě jeho hodnoty. Indexy se umisťují nad některé sloupce, ve kterých se často vyhledává a je v nich umístěno velké množství záznamů. To umožňuje databázi rychlejší vyhledávání v dané tabulce. Indexy se automaticky vytvářejí nad sloupci primárních klíčů. Definovat index lze po vytvoření tabulky nebo hned při jejím vytváření. Příklad:

```
Creat table test (
  `radek` int(11) NOT NULL AUTO_INCREMENT,
  index(`radek`)
);
```

Seznam tabulek a sloupců, kde byly indexy vytvořeny (nejsou uvedeny sloupce PK):

Tabulka 14 – Seznam sloupců s indexem

Tabulka	Sloupec
Autori	Prijmeni
Knihy	Nazev
Pracovnici	Cislo_OP, Prijmeni
Vydavatelstvi	Nazev
Zakaznici	Nickname, Cislo_OP, Prijmeni
Zanry	Nazev

5.3.4 Triggery

Trigger je v podstatě uložená procedura, která se nespouští příkazem, ale po provedení určitého dotazu nad určitou tabulkou. Je to takový hlídač, který může být použit k validaci vstupních dat, k odstraňování nepotřebných dat apod. V naší databázi jsou vytvořeny tři triggery. Všechny plní stejnou funkci. Tyto triggery jsou umístěny na tabulkách Zakaznici, Vydavatelstvi a Pracovnici. Jejich úkolem je vymazat případná nadbytečná data, která nemají žádnou vazbu v databázi, v tabulkách Adresa a Mesta. Tyto triggery jsou umístěny nad tabulkami, které mají vazbu s tabulkou adresa. Duplicita triggeru se stejnou funkcí na více tabulek je zapříčiněna tím, že trigger v MySQL databázi neumí mazat data v tabulce, na níž je definován. Jako příklad je zde uveden pouze jeden z triggerů. Ostatní jsou zcela shodné až na název tabulky.

```
create trigger trSmazAdresy_zak
after delete
on zakaznici
for each row
begin
  delete from adresa where adresa_id not in (select adresa_id from
zakaznici) and adresa_id not in (select adresa_id from pracovnici) and
adresa_id not in (select adresa_id from vydavatelstvi);
  delete from mesta where psc not in (select psc from adresa);
end;
```

5.3.5 Uživatelé

Vzhledem k tomu, že k databázi se přistupuje ze dvou různých klientských aplikací, přičemž každá využívá jiné tabulky a provádí nad nimi různé operace, jsou v databázi vytvořené dva uživatelské účty. Jeden pro zákaznickou aplikaci a druhý pro zaměstnaneckou aplikaci. V tabulkách je znázorněno, které dotazy, nad kterou tabulkou jsou povoleny.

Tabulka 15 – Přidělení práv uživateli pro zákaznickou aplikaci

Tabulka/právo	Select	Insert	Update	Delete
Adresa	ANO	ANO		
Aktuality	ANO			
Autori	ANO			
Knihy	ANO			
Mesta	ANO	ANO		
Pracovnici				
Vydavatelstvi	ANO			
Vypujcky	ANO			
Vytisky	ANO			
Zakaznici	ANO	ANO	ANO	
Zamluvene_knihy	ANO	ANO		
Zanry	ANO			

Tabulka 16 – Přidělení práv uživateli pro zaměstnaneckou aplikaci

Tabulka/právo	Select	Insert	Update	Delete
Adresa	ANO	ANO		
Aktuality	ANO	ANO		ANO
Autori	ANO	ANO		ANO
Knihy	ANO	ANO		ANO
Mesta	ANO	ANO		
Pracovnici	ANO	ANO	ANO	ANO
Vydavatelstvi	ANO	ANO	ANO	ANO
Vypujcky	ANO	ANO	ANO	ANO
Vytisky	ANO	ANO		ANO
Zakaznici	ANO	ANO	ANO	ANO
Zamluvene_knihy	ANO	ANO		ANO
Zanry	ANO	ANO		ANO

5.3.6 Nastavení češtiny

Aby databáze byla schopna ukládat české znaky a správně s nimi pracovat, musí být správně nastavena znaková sada. Znakovou sadu můžeme nastavit pro každý sloupec zvlášť, nebo pro celou tabulku klauzulí CHAR SET. Dalším podmínkou správného fungování je, že klient musí posílat znaky ve znakové sadě, na které se dohodli s databází, tzn. buď ve výchozí znakové sadě, nebo v jiné znakové sadě nastavené pomocí klauzule SET NAMES. Dále je nutné nastavit správné řazení dat. Jelikož má čeština několik speciálních znaků je toto nastavení důležité. Řazení se nastavuje klauzulí COLLATE. V našem případě tedy byla znaková sada nastavena na *utf8* a řazení bylo nastaveno na *utf8_czech_ci*. Příklad:

```
Creat table test (
  `radek` int(11) NOT NULL AUTO_INCREMENT,
  index(`radek`)
) DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;
```

6 Propojení MySQL s jazykem C++

Pro komunikaci mezi programem vytvořeným v Qt a databází MySQL, byla vytvořena třída, která se jmenuje MYSQL_CL. Qt sice mají podporu pro MySQL již integrovanou, takže vytváření vlastní třídy se zdá jako zbytečné. Vlastní třída byla použita, protože jsem se touto problematikou zabýval již dříve a tuto třídu jsem si vytvořil. Avšak nikdy nebyla použita v rozsáhlejším projektu, a proto jsem se rozhodl, že ji použiji, abych zjistil jaké má nedostatky, ty následně opravil a mohl třídu používat v dalších projektech, které nebudou vyvíjeny pod Qt. Třída využívá hlavičkového souboru mysql.h a knihovny libmysql.dll. Oba tyto soubory je možné nainstalovat do PC při instalaci MySQL serveru nebo si je stáhnout samostatně. Zdrojový kód souboru MYSQL_CL.cpp je přiložen v příloze A.

6.1 Připojení mysql.h a libmysql.lib do Visual Studia

Aby bylo možné při programování využívat funkce pro komunikaci s MySQL, je nutné nejdříve připojit potřebné soubory do našeho programovacího prostředí. Připojení souborů do vývojového prostředí Microsoft Visual Studio se provádí následujícím způsobem. Adresář, v němž jsou uloženy „include“ soubory se nastaví v: *Project > Properties > Configuration Properties > C/C++ > General > Additional Include Directories*. Následně je možné do projektu přidat:

```
#include <mysql.h>
```

Pro vložení statické knihovny se postupuje podobným způsobem. Nejprve se nastaví cesta k adresáři s knihovnou: *Project > Properties > Configuration Properties > Linker > General > Additional Library Directories*. Poté je ještě nutné přidat knihovnu libmysql.lib. To se provede v: *Properties > Linker > Input > Additional Dependencies*. Nyní je již možné v našem projektu vytvořit instanci třídy MYSQL.

```
MYSQL mysql;
```

Dále je nutné vložit soubor libmysql.dll do adresáře: *C:/windows/system32/*. A v mém případě, kdy jsem použil MySQL server 5.1, bylo nutné přidat do hlavičkového souboru mysql.h následující include:

```
#include <winsock2.h>
```

6.2 Popis třídy MYSQL_CL

6.2.1 Popis atributů

Třída obsahuje tři privátní atributy. Hlavním je atribut s názvem *mysql*, který je datového typu *MYSQL*. Po vytvoření třídy a následném připojení k databázi, jsou v tomto atributu uloženy potřebné informace o připojení k databázi. Dále třída obsahuje dva atributy datového typu *bool*. První se jmenuje *init*. Při konstruování třídy je tento atribut

nastaven na hodnotu *false*. Pokud dojde k úspěšnému inicializování atributu *mysql* je nastavena hodnota na *true*. Druhým atributem je *conn*. Ten je také při konstruování třídy nastaven na hodnotu *false*. Hodnota *true* je pak nastavena ve chvíli, kdy dojde k úspěšnému připojení k databázi. Oba tyto atributy se využívají ke kontrole při provádění metod, aby nedocházelo k pádu aplikace, když by nebylo inicializováno spojení nebo nebylo vytvořeno připojení k databázi.

6.2.2 Popis metod

Zde bude popsáno několik nejdůležitějších funkcí třídy MYSQL_CL.

Konstruktor

V těle konstruktoru se nastaví atributy *init* a *conn* na hodnotu false. Následně dojde k inicializování atributu *mysql*. Pokud inicializace proběhne úspěšně je atribut *init* nastaven na hodnotu *true.* Když dojde při inicializaci k chybě je vržena výjimka datového typu VYJIMKA_CL.

Destruktor

Při provádění destruktoru dojde k uzavření spojení s databází.

Connect

Tato metoda slouží k otevření spojení s databází. Jejími vstupními parametry jsou IP adresa hostitele, jméno uživatele, heslo, jméno databáze a číslo portu. Jestliže byla před zavoláním této metody úspěšně provedena inicializace, dojde k připojení k databázi a do atributu *mysql* se nastaví potřebné informace a atribut *conn* je nastaven na hodnotu *true*. Ve chvíli kdy dojde k jakékoliv chybě, je vržena výjimka typu VYJIMKA_CL.

Query

Účelem metody Query je vykonání dotazu v databázi a vrácení jeho výsledku. Metoda vrací ukazatel na datový typ RESULT_MYSQL, ve kterém je uložen výsledek dotazu. Vstupním parametrem metody je dotaz, který má být vykonán. Metoda opět vrhá výjimku typu VYJIMKA_CL, ve chvíli kdy není inicializováno nebo vytvořeno spojení s databází, nebo dojde k chybě při vykonávání dotazu.

EscapeString

Poslední blíže popisovanou metodou je EscapeString. Metoda slouží k ochraně před SQL injection. Vstupní parametr metody je nějaký text. V metodě dojde k identifikaci každého znaku ve vstupním textu. Pokud je určitý znak vyhodnocen jako nebezpečný, dojde k jeho převedení na znak bezpečný (viz. Kapitola 7.3.2). Po převedení textu na bezpečné znaky je bezpečný text vrácen návratovou hodnotou. Opět při výskytu chyby dojde k vržení výjimky typu VYJIMKA_CL.

Ostatní metody

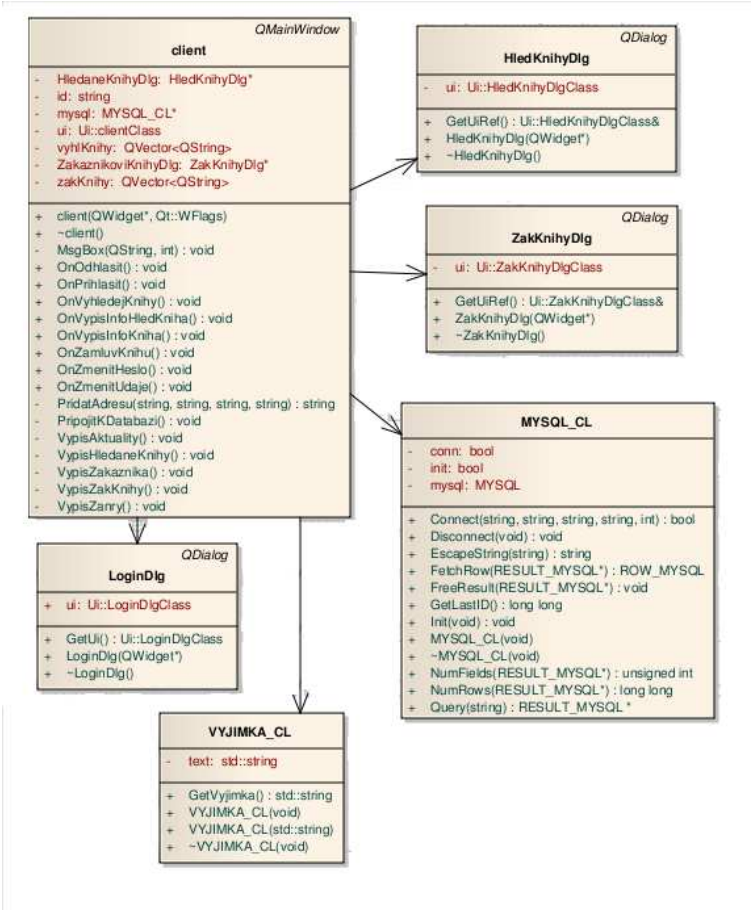
Třída obsahuje několik dalších metod, které zde nebudou blíže popisovány. Jedná se o metody sloužící pro práci s výsledkem dotazu (počet řádků a sloupců ve výsledku, vrácení dalšího řádku z výsledku, …). Dále jsou dostupné metody pro inicializaci spojení, odpojení od databáze nebo metoda vracející poslední vložené ID. Metoda pro Commit v této třídě není, z toho důvodu, že většina MySQL databází má nastavené AutoCommit po každém dotazu. Pokud by měla být tato třída použita pro práci s transakcemi, bylo lepší použít nějaké robustnější řešení jako například využít podpory práce s MySQL v Qt.

6.3 Popis třídy VYJIMKA_CL

Tato třída slouží pro vrhání výjimek ve třídě MYSQL_CL. Třída je velice jednoduchá, obsahuje jeden atribut, ve kterém je uložen text výjimky. V bezparametrickém konstruktoru se do textu výjimky vloží slovo chyba. Dále třída obsahuje parametrický konstruktor, jehož vstupním parametrem je test výjimky. Poslední metoda třídy je GetVyjimka, která vrací text z atributu.

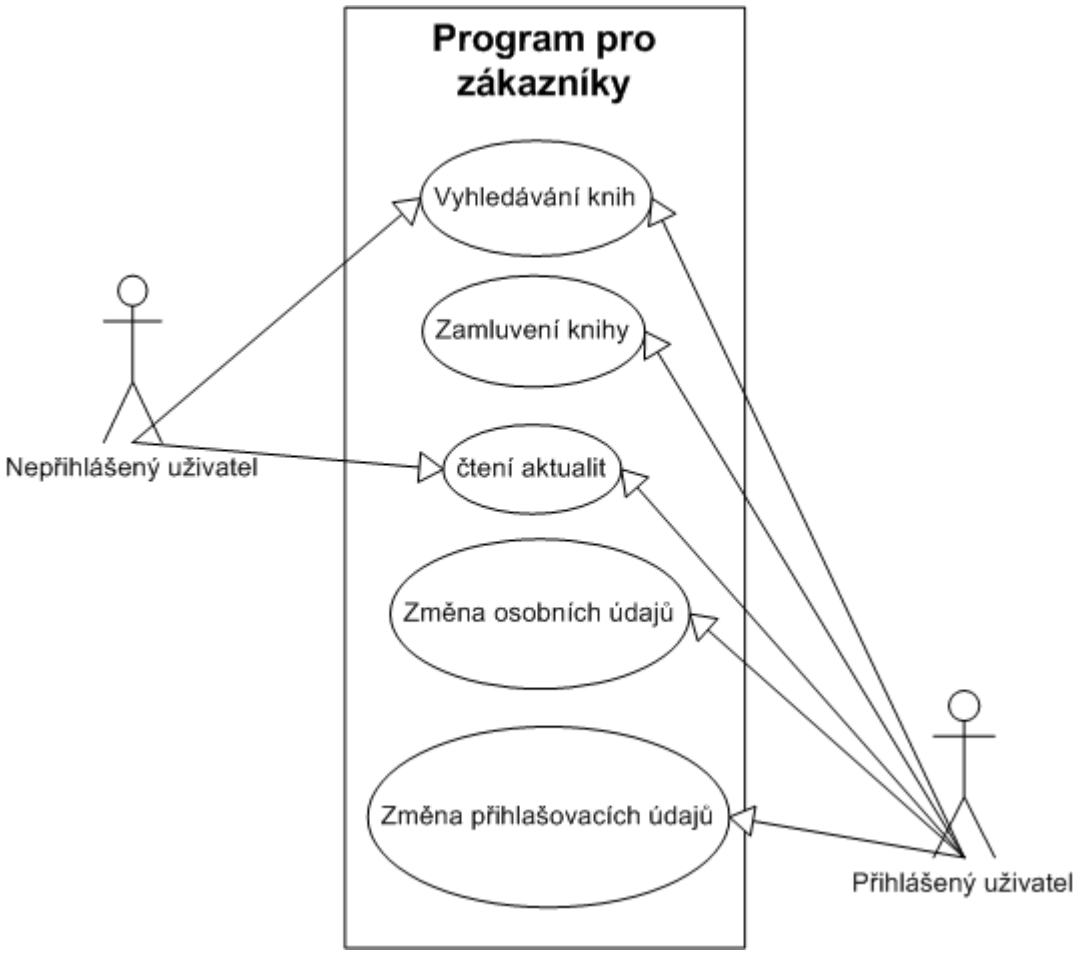
7 Architektura aplikací

7.1 UML class diagram zákaznická aplikace



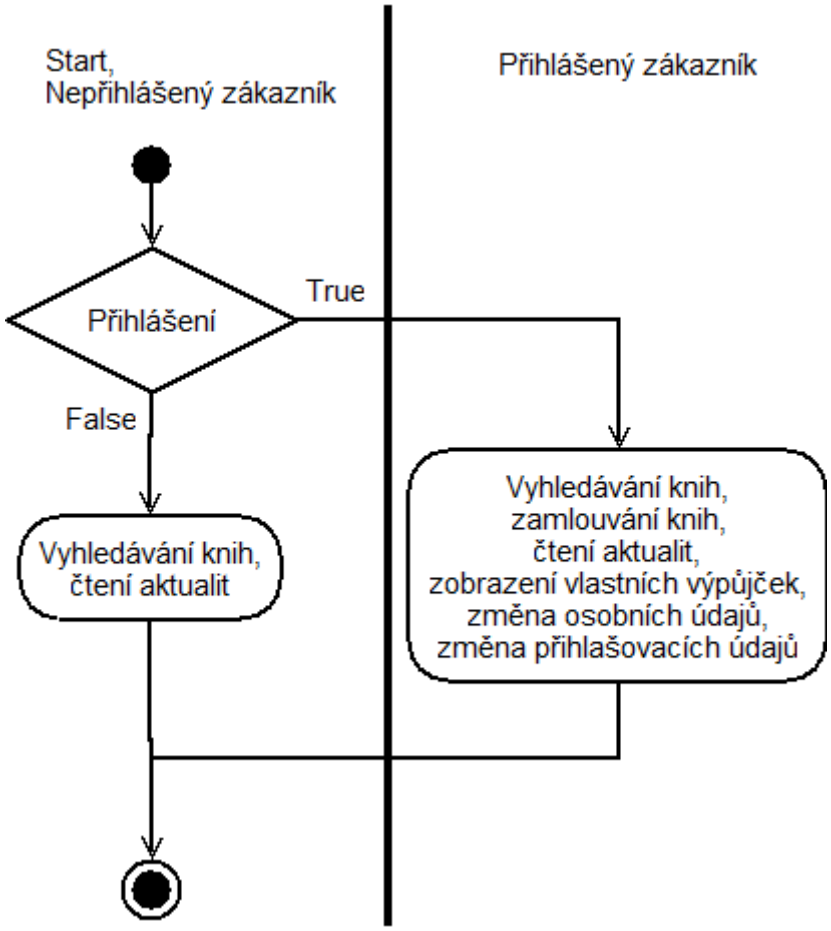
Obrázek 13 – Class diagram zákaznické aplikace

7.2 UML use case diagram zákaznická aplikace



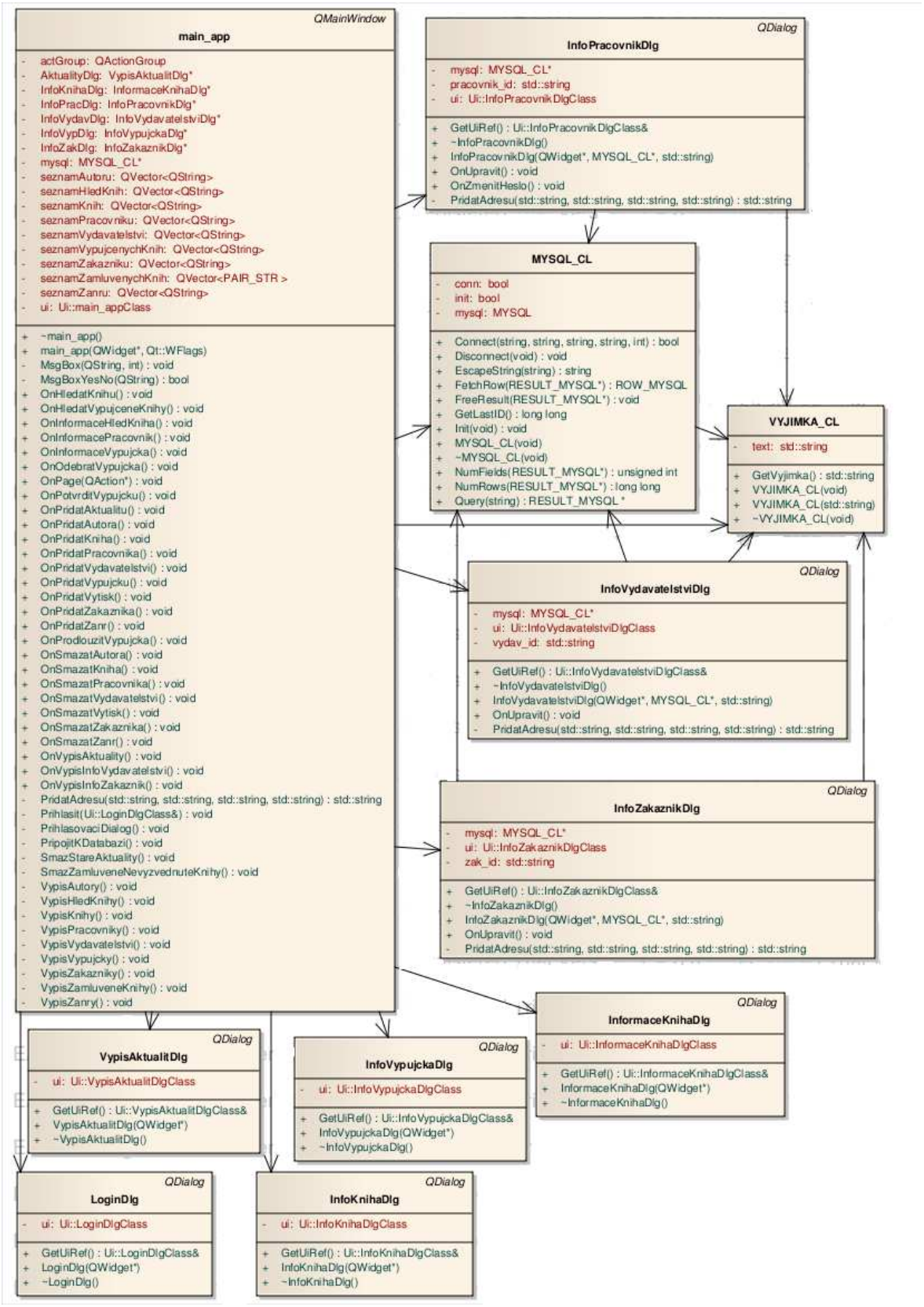
Obrázek 14 – Use case diagram zákaznické aplikace

7.3 UML activity diagram zákaznická aplikace



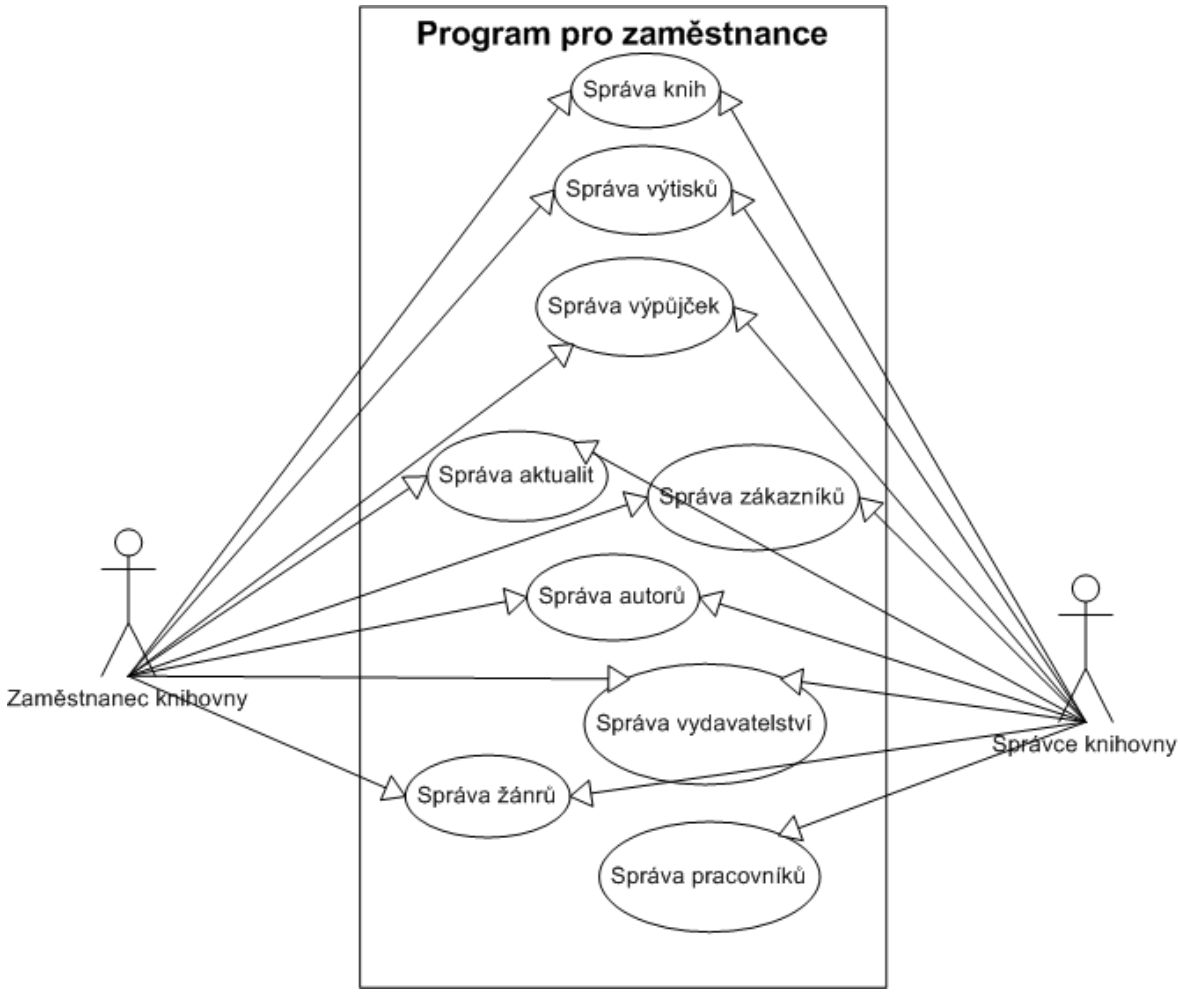
Obrázek 15 – Activity diagram zákaznické aplikace

7.4 UML class diagram zaměstnanecká aplikace



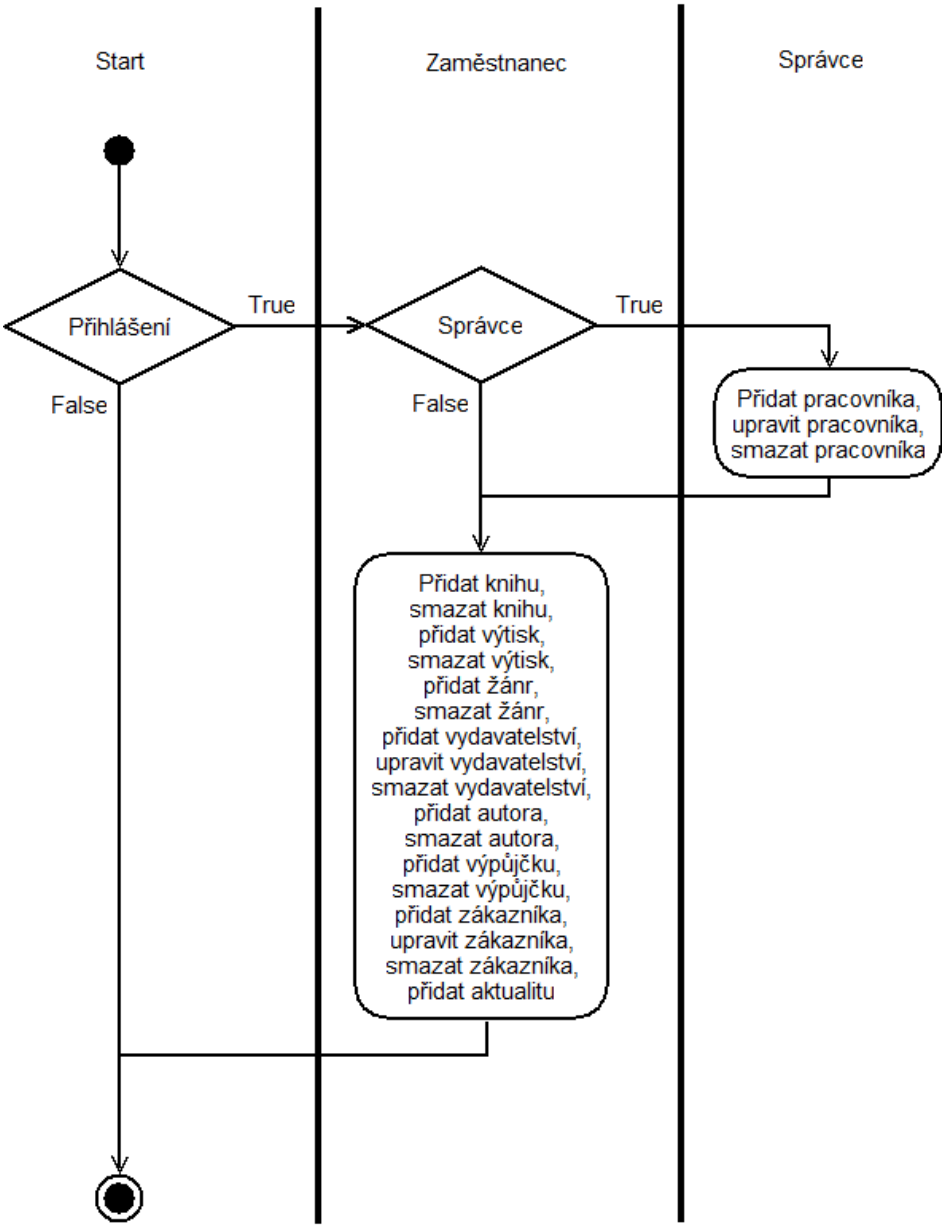
Obrázek 16 – Class diagram zaměstnanecké aplikace

7.5 UML use case diagram zaměstnanecká aplikace



Obrázek 17 – Use case diagram zaměstnanecké aplikace

7.6 UML activity diagram zaměstnanecká aplikace



Obrázek 18 – Activity diagram zaměstnanecké aplikace

8 Příklady zdrojových kódů

8.1 Čeština

Jak už to tak bývá, tak při programování aplikace, která by měla podporovat český jazyk a tím pádem i české znaky, nebylo jednoduché najít vhodné řešení, aby vše spolupracovalo správně i s databází MySQL. Dalším problémem bylo, že třída MYSQL_CL je naprogramována takovým způsobem, aby ji bylo možno využít při tvorbě aplikací v jazyce C++ bez knihovny Qt. Tudíž třída pracuje s datovými typy jazyka C++ a veškeré textové řetězce, se kterými pracuje, jsou datového typu string. Bylo tedy nutné, aby se načítaný text byl převeden z datového typu QString na datový typ string. Čeština v databázi byla nastavena dle výše uvedeného popisu. Pokud by se na převod z QString na string použila funkce QString::toStdString(), která je určena právě pro tento převod a její návratovou hodnotou je datový typ string, došlo by k znehodnocení českých znaků. Pro převod byla tedy použita funkce QString::toLocal8Bit(). Tato funkce vrací 8-bitovou reprezentaci stringu jako QByteArray, který lze již použít při konstruování stringu. Při zpětném načítání, aby byly české znaky správně zobrazeny, byla použita funkce QString::fromLocal8Bit(const char * str, int size = -1). Funkce má dva vstupní parametry. Parametr size slouží pro nastavení délky načítaného textu, pokud je tento parametr uveden, je použita základní hodnota -1 a text je načítán po ukončovací znak. Druhým parametrem funkce je ukazatel na načítaný text. Návratová hodnota funkce je QString, který obsahuje správné české znaky.

Načítání textu z Line Editu do string:

```
string text;
text = ui.lineEdit->text().toLocal8Bit();
```

Načítání textu převedeného pomocí funkce QString::toLocal8Bit() do QString:

```
QString qText;
qText = QString::fromLocal8Bit (text.c_str());
```

8.2 Práce s databází

Zde je uvedena ukázková funkce práce s databází pomocí třídy MYSQL_CL. Všechny funkce, prováděné objektem *mysql*, který je datového typu MYSQL_CL, jsou uzavřeny v bloku *try*. V tomto bloku jsou odchytávány všechny případně vržené výjimky objektem *mysql*. V prvním kroku je instance datového typu string s názvem *q*. Do ní uložen požadovaný dotaz do databáze., v našem případě dotaz na vypsání všech názvů žánrů. Následně je vytvořen ukazatel na výsledek dotazu datového typu RESULT_MYSQL, do kterého je následně přiřazena hodnota z metody *Query*, která požadovaný dotaz vykonala a vrátila ukazatel na výsledek. Dále je testováno zda nebyla vrácena hodnota NULL. Pomocí metody *NumRows* se zjistí počet řádků ve výsledku. Pokud je ve výsledku jeden a více řádků, dojde k vytvoření instance s názvem *row*, která je datového typu ROW_MYSQL. Do *row* se poté pomocí metody *FetchRow* ukládají

jednotlivé řádky z výsledku dotazu. Výsledek je v *row* uložen jako pole a přistupuje se k němu stejně jako k datům v poli. Ve chvíli, kdy by nějaká metoda vrhla výjimku, bude tato výjimka odchycena a zobrazena její chybová zpráva. Při vkládání, upravování nebo mazání dat je postup poněkud snažší. V tu chvíli vytvoříme pouze požadovaný dotaz a zavoláme metodu *Query*, která dotaz vykoná. Pokud dojde k nějaké chybě při vykonávání dotazu je vržena výjimka.

```
try{
	string q = "select nazev from zanry";
	RESULT_MYSQL *res;
	res = mysql->Query(q);
	if(res != NULL){
		if(mysql->NumRows(res) > 0){
			ROW_MYSQL row;
			while(row=mysql->FetchRow(res))
			QString nazevZanru (row[0]);
		}
	}
	}
	catch (VYJIMKA_CL *v){
		QString vyjText(v->GetVyjimka().c_str());
		MsgBox(vyjText,4);
	}
```

8.3 Vypisování údajů z databáze a práce s nimi

Pro vypisování seznamů určitých informací byl použit widget QListWidget. Po provedení dotazu v databázi a získání výsledku se pro každý řádek provede následující operace. Příjmení zákazníka je uloženo do instance datového typu QString s názvem *prameni*. Následně je vytvořen nový objekt typu QListWidgetItem, který je hned inicializován na hodnotu *prameni* a je vložen do QListWidgetu *lwZakaznici*. Nakonec je do atributu třídy *zakazniciPrijmeni*, jenž je datového typu QVector<QString> vloženo ID zákazníka. Jednotlivá umístění ID ve vectoru korespondují s hodnotami v řádcích *lwZakaznici*. Pokud je tedy vybrán některý řádek v list widgetu, je zjištěn jeho řádek, z vectoru je následně načtené ID vybraného zákazníka a s tímto ID se dále pracuje. Může být například použito jako podmínka dotazu do databáze, kdybychom chtěli vypsat podrobnější informace o daném zákazníkovi.

```
try{
	string q = "select id, prijmeni from zakaznici";
	RESULT_MYSQL *res;
	res = mysql->Query(q);
	if(res != NULL)
		if(mysql->NumRows(res) > 0){
			ROW_MYSQL row;
			while(row=mysql->FetchRow(res)){
				QString prijmeni(row[1]);
				new QListWidgetItem(prijmeni, ui.lwZakaznici);
				zakazniciPrjmeni.push_back(row[0]);
			}
		}
	}
	catch (VYJIMKA_CL *v){
```

```
            QString vyjText(v->GetVyjimka().c_str());
            MsgBox(vyjText,4);
      }
```

8.4 Zobrazování informačních zpráv

V Qt je dostupná třída s názvem QMessageBox, která zobrazuje modální okno s otázkou nebo s informací a je možno přijímat odpověď. Lze zobrazit několik základních tlačítek (Yes, No, Cancel, …) a několik ikon. Pro zjednodušení byly vytvořeny dvě metody, které zobrazují toto modální okno. První je metoda MsgBox, která zobrazuje informativní okno s ikonou, textem a tlačítkem Ok. Jedním ze dvou vstupních parametrů je text informace a druhým parametrem je číslo, které určuje, jaká ikona bude zobrazena.

```
void client::MsgBox(QString text, int typ){
      QMessageBox msgBox;
      switch(typ){
            case 1: msgBox.setIcon(QMessageBox::Question);
                  break;
            case 2: msgBox.setIcon(QMessageBox::Information);
                  break;
            case 3: msgBox.setIcon(QMessageBox::Warning);
                  break;
            case 4: msgBox.setIcon(QMessageBox::Critical);
                  break;
      }
      msgBox.setText(QString::fromLocal8Bit(text.toStdString().c_str()));
      msgBox.exec();
}
```

Druhou metodou na zobrazování message boxu je metoda se jménem MsgBoxYesNo. Má jeden vstupní parametr, kterým je zobrazovaný text. Zobrazené modální okno obsahuje tlačítka Yes a No. Pokud je zvoleno Yes, metoda vrací hodnotu *true*, pokud No, je vrácena hodnota *false*.

```
bool main_app::MsgBoxYesNo(QString text){
      QMessageBox msgBox;
      msgBox.setIcon(QMessageBox::Question);
      msgBox.setText(QString::fromLocal8Bit(text.toStdString().c_str()));
      msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);
      msgBox.setDefaultButton(QMessageBox::No);
      int ret = msgBox.exec();
      if(ret == QMessageBox::Yes)
            return true;
      else
            return false;
}
```

8.5 Kontrola dat aktualit a zamluvených knih

Ve chvíli, kdy si zákazník zamluví knihu, dojde k zapsaní určitých údajů do tabulky Zamluvene_knihy. Zamluvená kniha se poté pro ostatní zákazníky jeví jako nedostupná. Mohlo by dojít k situaci, že si zákazník knihu zamluví, ale již si ji nevyzvedne a tím pádem není výpůjčka potvrzena a kniha by zůstala „viset“ mezi zamluvenými knihami a nikdo by

si ji nemohl vypůjčit. Dále v tabulce aktuality jsou ukládány krátké novinky o knihovně. Je ale zbytečné, aby v tabulce byly uloženy staré a již dávno nepotřebné informace. Z těchto důvodů je důležité, aby data v tabulkách Zamluvene_knihy a Aktuality byly po vhodné době smazány. V původním plánu řešení bylo vytvoření triggerů nad těmito tabulkami, které by stará data mazaly. Databázový systém MySQL však neumožňuje, aby trigger mazal data v tabulce, nad níž je vytvořen. Zároveň práce s datumy při vytváření uložených procedur nebo triggerů se mi zdála velice zmatená, až skoro nepochopitelná.

Mazání nepotřebných údajů ve výše zmíněných tabulkách bylo tedy vyřešeno pomocí metod v obslužném programu pro zaměstnance. Tyto metody se provedou pokaždé při spuštění programu. Z použitého řešení je jasné, že může nastat situace, kdy bude kniha zamluvena déle než je nastaveno a totéž platí i pro aktuality. Avšak při použití triggerů bychom se dostali do stejné situace, protože nemáme jistotu, že bude proveden potřebný dotaz nad danými tabulkami.

8.6 Obsluhování událostí v Qt

Pomocí grafického rozhraní lze přehledně navrhnout aplikaci v Qt. Pokud chceme, aby nějaká událost způsobila zavolání nějaké metody, máme dvě možnosti. Můžeme pomocí wizardu na daný objekt připojit tzv. slot. Slot je v podstatě metoda, která může být spojena s nějakou událostí. Pokud připojíme slot k signálu (události) pomocí wizardu, vývojový program automaticky vygeneruje vše potřebné a my můžeme jen napsat libovolnou funkci. Toto řešení však má jisté nedostatky. V kódu není zřetelně jasné, jaký objekt je spojen s jakým slotem a nelze připojit ke slotu všechny signály, které objekty nabízí. Druhým způsobem je připojovat signály na sloty pomocí funkce connect.

```
QObject::connect(ui.scrollBar, SIGNAL(valueChanged(int)),
this,  SLOT(setNum(int)));
```

Do této funkce vstupují čtyři parametry. První je objekt, který vyšle daný signál, druhý typ signálu, který je vyslán, třetím parametrem je objekt, který daný signál odchytí a poslední parametr je slot, který bude zavolán. Sloty mohou být deklarovány jako public slots nebo private slots. Při definování slotu se pak postupuje stejným způsobem jako při definování kterékoliv jiné metody.

8.7 Ochrana před SQL injection

Ochranu před SQL injection zajišťuje metoda třídy MYSQL_CL se jménem EscapeString, která je popsána výše. Pokud chceme tuto metodu použít, postup je následující. Máme vytvořené instance datového typu string s názvem *text* a *textEscape*. Do *text* si přiřadíme vstupní text, který chceme ošetřit proti nebezpečným znakům. Tato proměnná je poté použita jako vstupní parametr metody EscapeString, která ošetří nebezpečné znaky a „bezpečný“ text je přiřazen do proměnné *textEscape*.

```
string text = "zkusebni text";
string textEscape = mysql->EscapeString(text);
```

Dalším prvkem, který zabraňuje SQL injection je fakt, že aplikace nevypisuje žádné chybové hlášky z databáze a tudíž si pomocí nich nemůže útočník zjišťovat názvy tabulek a na ně poté útočit. Ochraně aplikace také přispívá to, že aplikace přistupují do databáze přes speciální uživatelské účty, které mají nastaveny práva takovým způsobem, že mají povoleny pouze nejnutnější dotazy pro správné fungování.

9 Závěr

S výsledkem práce jsem spokojen. Obě aplikace, zákaznická i zaměstnanecká, podle mého mínění splňují zadání. Byla vyřešena ochrana proti SQL injection a obslužná část byla rozdělena do dvou programů. Každý z obou programů obsahuje všechny potřebné funkce a naopak nemá funkce, ke kterým by neměl mít přístup.

Při práci se vyskytlo několik problémů. Prvním z nich bylo, že hlavičkový soubor mysql.h, potřebný k propojení jazyka C++ s databází MySQL, neměl „includovaný“ hlavičkový soubor winsock2.h. Po přidání tohoto hlavičkového souboru již bylo vše v pořádku. Dalším problémem bylo vyřešení problémů s češtinou, konkrétně aby klientské aplikace a databázový systém při své komunikaci „neztrácely“ české znaky. Tento problém byl vyřešen nastavenou znakovou sadou v databázi a užitím výše zmíněných funkcí Qt.

Aby byly oba programy zcela bez „much“ a vyhovovaly potřebám uživatelů, bylo by zapotřebí, aby programy po nějakou běžely ve zkušebním provozu se zpětnou vazbou od testovacích uživatelů. Připomínky by poté byly vyhodnoceny a případně zaneseny do programů. Do budoucna by bylo vhodné program rozšířit o podporu čárových kódů. V takovém případě by pak odpadlo zadávání údajů ručně. Potřebné údaje by byly načteny pomocí čtečky z čárového kódu. Dále by program v budoucnu mohl obsahovat automatické rozesílání emailů uživatelům, kterým končí doba na vrácení vypůjčené knihy.

Literatura

cCuMiNn. 2004. Útoky - SQL injection. *SOOM.cz.* [Online] 3. 9 2004. [Citace: 5. 5 2010.] http://www.soom.cz/index.php?name=articles/show&aid=167.

Corporation, Oracle. 2010. *MySQL :: Developer Zone.* [Online] 2010. [Citace: 5. 5 2010.] http://dev.mysql.com/.

Horváth, Tomáš. 2007. Teoretický úvod do relačních databází. *Programujte.* [Online] 8. 11 2007. [Citace: 5. 5 2010.] http://programujte.com/?akce=clanek&cl=2007110801-teoreticky-uvod-do-relacnich-databazi.

Ludačka, Radek. 2009. QT framework - pomocník programátora. *SWMAG.cz.* [Online] 15. 10 2009. [Citace: 5. 5 2010.] http://www.swmag.cz/546/qt-framework-pomocnik-programatora/.

Prata, Stephen. 2007. *Mistrovství v C++*. místo neznámé : Computer Press, 2007. 978-80-251-1749-1.

Šimůnek, Milan. 1999. *SQL kompletní kapesní průvodce.* místo neznámé : GRADA Publishing, a.s., 1999. 80-7169-692-7.

Vojáček, Petr. 2007. SQL Injection a zabezpečení. *Programujte.* [Online] 23. 4 2007. [Citace: 5. 5 2010.] http://programujte.com/?akce=clanek&cl=2007041802-sql-injection-a-zabezpeceni.

Welling, Luke a Thompson, Laura. 2005. *MySQL : průvodce základy databázového systému.* Brno : CP Books, 2005. 80-251-0671-3.

Zajíc, Petr. 2005. mysql. *Linux Software.* [Online] 2005. [Citace: 5. 5 2010.] http://www.linuxsoft.cz/mysql/.

Žák, Karel. 2001. Historie relačních databází. *Root.cz.* [Online] 19. 10 2001. [Citace: 5. 5 2010.] http://www.root.cz/clanky/historie-relacnich-databazi/.

RNDr. **Žák, David, Ph.D.**. *Databázové systémy I.* (přednáška) Pardubice :UPCE, 2009.

Ing. **Štroch, Jaroslav.**. *Jazyk C++ III.* (přednáška) Pardubice :UPCE, 2010.

Příloha A – Zdrojový kód souboru mysql_cl.cpp

```
#include "mysql_cl.h"
#include "vyjimka_cl.h"
#include <mysql.h>
#include <iostream>
#include <string>
using namespace std;

//KONSTRUKTOR(bezparametricky) - pri zavolani konstruktoru dojde k
inicializaci atributu mysql
//promenne init a conn se nastavi na false. pokud se inicializace provede
nastavi se promenna init na hodnotu true,
//pokud se inicializace neprovede je vrzena vyjimka.
MYSQL_CL::MYSQL_CL(void) throw (VYJIMKA_CL)
{
      init=false;
      conn=false;
      if(mysql_init(&mysql)) init=true;
      else{
            throw new VYJIMKA_CL("Chyba při inicializaci připojení k
databázi");
      }

}

//DESTRUKTOR - pri zavolani destruktoru se ukonci spojeni s databazi
MYSQL_CL::~MYSQL_CL(void)
{
      mysql_close(&mysql);
}

//CONNECT - slouzi pro pripojeni k databazi se zadanymi parametry. Pokud
pripojeni probehne v poradku,
//vrati true a atribut conn nastavi na true, jinak false a je vrzena
vyjimka.
bool MYSQL_CL::Connect(std::string IPAddr, std::string user, std::string
passw, std::string datName, int port) throw (VYJIMKA_CL)
{
      if(init)
            if(mysql_real_connect(&mysql, IPAddr.c_str(), user.c_str(),
passw.c_str(), datName.c_str(), port, 0, 0)){
                  conn=true;
                  return true;
            }
            else{
                  throw new VYJIMKA_CL("Chyba při pokusu o připojení k
databázi");
            }
      else{
            throw new VYJIMKA_CL("Chyba s připojením k databázi");
      }
      return false;
}


//QUERY - slouzi k vykonani dotazu. Pokud je dotaz vykonan, metoda vrati
ukazatel na vysledek dotazu typu MYSQL_RES.
//Pokud dotaz neprobehl je vracena hodnota NULL
RESULT_MYSQL * MYSQL_CL::Query(std::string queryIn) throw (VYJIMKA_CL)
```

```
{
	if(init && conn)
		if(mysql_real_query(&mysql,queryIn.c_str(),queryIn.length())
== 0){
		RESULT_MYSQL *vysledek;
			vysledek = mysql_store_result(&mysql);
			return vysledek;
		}
		else{
			throw new VYJIMKA_CL("Chyba při provádění dotazu");
		}
	throw new VYJIMKA_CL("Chyba s připojením k databázi");
}

//FREERESULT - uvolnuje pamet alokovanou pro vysledky dotazu
void MYSQL_CL::FreeResult(RESULT_MYSQL *res) throw (VYJIMKA_CL)
{
	if(init && conn)
		mysql_free_result(res);
	else{
			throw new VYJIMKA_CL("Chyba s připojením k databázi");
		}
}

//GETLASTID - vraci hodnotu posledniho vlozeneho ID
long long MYSQL_CL::GetLastID() throw (VYJIMKA_CL)
{
	if(init && conn)
		return mysql_insert_id(&mysql);
	else
		throw new VYJIMKA_CL("Chyba s připojením k databázi");
}

//ESCAPESTRING - zabezpecuje DB pred SQL injection, odstranuje nebezpecne
znaky
string MYSQL_CL::EscapeString(std::string text) throw (VYJIMKA_CL)
{
	if(init && conn){
		char query[1000];

			mysql_real_escape_string(&mysql,
query,text.c_str(),text.length());

			return query;
	}
	else
		throw new VYJIMKA_CL("Chyba s připojením k databázi");
}

//DISCONNECT - odpoji od databaze a dojde k dealokaci atributu mysql.
//Pred novym pouzitim znova inicializat atribut
void MYSQL_CL::Disconnect() throw (VYJIMKA_CL){
	if(init){
		mysql_close(&mysql);
		init = false;
		conn = false;
	}
	else
		throw new VYJIMKA_CL("Chyba s připojením k databázi");
}
```

```
//INIT - inicializuje atribut mysql
void MYSQL_CL::Init() throw (VYJIMKA_CL){
      if(init = false)
            if(mysql_init(&mysql)) init=true;
            else
                  throw new VYJIMKA_CL("Chyba s připojením k databázi");
}

//NumFields - vraci pocet sloupcu ve vysledku
unsigned int MYSQL_CL::NumFields(RESULT_MYSQL *res) throw (VYJIMKA_CL){
      if(init && conn)
            return mysql_num_fields(res);
      else
            throw new VYJIMKA_CL("Chyba s připojením k databázi");
}

//FetchRow - vraci dalsi radek vysledku
ROW_MYSQL MYSQL_CL::FetchRow(RESULT_MYSQL *res) throw (VYJIMKA_CL){
      if(init && conn)
            return mysql_fetch_row(res);
      else
            throw new VYJIMKA_CL("Chyba s připojením k databázi");
}

long long MYSQL_CL::NumRows(RESULT_MYSQL * res) throw (VYJIMKA_CL){
  if(init && conn)
            return mysql_num_rows(res);
  else
       throw new VYJIMKA_CL("Chyba s připojením k databázi");
}
```