Česká zemědělská univerzita v Praze

Technická fakulta

Katedra elektrotechniky a automatizace

Bakalářská práce

Propojení aplikace .NET Framework s databází typu MySQL

Vít Řezníček

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra elektrotechniky a automatizace

Technická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Vít Řezníček

Informační a řídicí technika v agropotravinářském komplexu

Název práce

Propojení aplikace .NET Framework s databází typu MySQL

Název anglicky

Connecting the .NET Framework application with database MySQL.

Cíle práce

Propojení aplikace .NET Framework s databází typu MySQL

Metodika

1. seznámení s objektově orientovaným programovacím jazykem .NET Framework

2. seznámení s databází typu MySQL

3. možnosti propojení – syntaxe, transakce, rozdílné table engine, DB operace na straně serveru/klienta, ...

4. testování – program komunikující s testovací databází, testování zpracování dotazů při různých stupních zatížení serveru

Doporučený rozsah práce

35 str.

Doporučené zdroje informací

MySQL 5.6 Reference Manual

Předběžný termín obhajoby

2015/05 (květen)

Vedoucí práce

Ing. René Neděla

Elektronicky schváleno dne 25. 3. 2015

prof. Ing. Jaromír Volf, DrSc.

Vedoucí katedry

Elektronicky schváleno dne 7. 4. 2015

prof. Ing. Vladimír Jurča, CSc.

Děkan

V Praze dne 07. 04. 2015

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Propojení aplikace .NET Framework s databází typu MySQL" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 05.04.2015 _____

Poděkování

Rád bych touto cestou poděkoval Ing. René Nedělovi za jeho vstřícnost a ochotu při konzultacích.

Propojení aplikace .NET Framework s databází typu MySQL

Connecting the .NET Framework application with database MySQL

Souhrn

V této práci je hlavním cílem seznámit se způsoby propojení platformy .NET a databázového systému MySQL. V práci je nastíněn vývoj platformy .NET, její součásti a vnitřní fungování. Jsou zde stručně představeny i nástroje pro vývoj aplikací pracující s touto platformou. Část práce o MySQL se zaobírá architekturou systému, jednotlivými součástmi a souvisejícími programy. Dále jsou popsány možné programátorské přístupy k připojování databází, používané knihovny platformy .Net pro spojení s databázemi a nástroje MySQL, poskytující tyto knihovny. V závěrečné části se práce věnuje praktickému využití poznatků o způsobech spojení v ukázkové databázové aplikaci.

Summary

In this thesis the main goal is to familiarize with means of connecting .NET platform to MySQL database system. In the thesis is depicted historical development of .NET platform, its components and inner workings. There are also briefly described application development tools working with this platform. A part of the thesis dedicated to MySQL describes architecture of the system, individual components and associated programs. Further there are pointed out possible programming choices to connect to databases, related .NET common libraries and also MySQL tools providing specific connection libraries. In the last part the thesis focuses on practical use of acquired knowledge about means of connection in exemplary database application.

Klíčová slova: MySQL, .NET, Visual Basic, Visual Studio, databázová aplikace

Keywords: MySQL, .NET, Visual Basic, Visual Studio, database application

Obsah

1. Úvod

V dnešní době, kdy je na jedné straně velký výběr komerčních databázových systémů (DBMS) i objektově orientovaných programovacích (OOP) jazyků, a na straně druhé snaha či potřeba zjednodušit nebo zefektivnit práci, je zde jasný potenciál pro rozvoj uživatelsky přívětivých řešení jakými jsou databáze (představující systém tabulek) propojené s aplikacemi (dodávající podle potřeby v modifikovatelné formě datové vstupy a výstupy).

Databázové systémy z komerční sféry jako Oracle nebo MS SQL jsou řešením, která nabízí komplexní možnosti, avšak neposkytují své produkty zdarma. Naproti tomu systém MySQL má dvojí licencování – komerční licence a GNU open source – jež dává možnosti i pro vytvoření databází s minimálními náklady. Jako další možnost bezplatného užívání databáze je nutno zmínit PostgreSQL (PSQL), jež ve srovnání s MySQL trpěla v rychlosti, avšak měla rozšířenou funkčnost, což se však v posledních verzích srovnalo, např. replikace dat u PSQL se s poslední verzí 9.1 výrazně zrychlila. MySQL je unikátní v možnosti používání různých table engine, čímž se dá podle analýzy četnosti a typu dotazů optimalizovat rychlost databáze. Výhodou MySQL oproti PSQL je také diagnostika databázového serveru v nativním prostředí Workbench, kde je možné sledovat mimo jiné průběh zatížení příkazy nebo četnost čtení/zapisování . Hlavní devizou je však velmi dobrá podpora pro začínající správce a vývojáře databází, kteří hledají systém oplývající funkčností, jako je víceuživatelská podpora, jakou zas nenabízí SQLite, nebo škálovatelností, a zároveň silný v jednoduché manipulaci s databází prostřednictvím vizuálních nástrojů i v integraci do aplikací.

Práce s databází by však byla pro člověka, který ji nikdy neviděl, vcelku obtížná. Učení se specifických příkazů pro zobrazení hledaných výsledků, kdy navíc je nutné znát strukturu normalizované databáze, není optimálním řešením. Nabízí se řešení grafické nadstavby, která slouží jako prvek ulehčující ovládání vstupů a výstupů databáze. Tato nadstavba, zvaná databázová aplikace, skýtá mnoho možností, co se týče práv uživatelů a kontroly jejich přístupu, rychlých a snadných změn v databázi anebo exportovatelnosti do formátů, se kterými je možno dále pracovat. Pro vývoj databázových aplikací lze jako v případě databází vybírat z mnoha OOP jazyků. Jako kritéria lze uvažovat dostupná vývojová prostředí, podporu knihoven, složitost syntaxe, multiplatformní podporu či rychlost běhu aplikace v různých operačních systémech (OS). Vzhledem k faktu, že na drtivé většině dnešních stolních počítačů běží jako OS různé verze Microsoft Windows, který nabízí platformu VB.NET jako

svůj primární nástroj pro vývoj aplikací, jeví se jako jedním z řešení program Microsoft Visual Studio. Ten kromě možností psaní aplikace v .NET podporuje i C# nebo J#, což usnadňuje přechod od jazyků C++ nebo Java a lze díky tomu např. implementovat knihovny různých jazyků.

Cílem této práce bylo vytvořit stručný přehled o platformě .NET a databázovém systému MySQL, obeznámit s nezbytnými nástroji a objekty pro uskutečnění jejich spojení a znázornit jejich využití na ukázkové databázové aplikaci. Aplikace navržená v této práci demonstruje několik možných způsobů jak propojit VB.NET platformu a MySQL databázový systém při kladení důrazu na funkční možnosti. Výběr tématu pro tuto práci vycházel z osobních zkušeností s propojením zmíněné platformy a systému.

2. Seznámení s platformou .NET

Tato platforma vznikla na základě COM (Component Object Model), který si kladl za cíl používat volně mezi sebou knihovny napsané v různých programovacích jazycích. Nevýhodou COMu byla mimo jiné složitá infrastruktura, kde jednotlivé knihovny byly zapisovány do registru, a také nemožnost použít ho mimo prostředí Windows. S nástupem .NET Framework přišlo jednodušší a flexibilnější řešení.

2.1 Vývoj. NET Frameworku

.NET Framework je s námi již zhruba od roku 2000. První verze 1.0 a 1.1 se dnes již moc nepoužívají. Od roku 2005 máme verzi 2.0, která přinesla nové běhové prostředí a mnoho nových vlastností. Zhruba o rok později přišla verze 3.0. Verze 3.0 a 3.5 jsou v podstatě rozšíření postavená nad stabilní verzí běhového prostředí (runtime) 2.0, běhové prostředí 3.0 neexistuje. Běhové prostředí .NET Framework 4.5 aktualizuje a nahrazuje běhové prostředí verze .NET Framework 4.0 (hlavní číslo verze běhového prostředí zůstává 4.0). Aktuální verzí je verze 4.5.1, která přinesla kromě dalších přidaných knihoven i rozšíření dvou hlavních jazyků Visual Basic .NET a C#. Vývoj

Zde je výčet několika vlastností .NET platformy:

- Spolupráce s existujícími COM knihovnami
- Podpora více programovacích jazyků – C#, JavaScript, VB.NET,F#, atd.
- Společné běhové prostředí pro všechny .NET jazyky
- Možnosti napříč jazyky – ošetření chyb, dědění, debugging, rozšiřování, atd.
- Mohutná kolekce základních knihoven, společná všem .NET jazykům
- Podpora více verzí jedné knihovny na jednom počítači

Tabulka 1 Vývoj platformy .NET

Verze	**Vývojové prostředí**	**Verze VB.NET**	**Verze C#**	**Vlastnosti**
1.0	Visual Studio 2002	Visual Basic 7	C# 1	- první verze frameworku
1.1	Visual Studio 2003			- dramatické zvýšení výkonu - rozšíření a změny API
2.0	Visual Studio 2005 Visual Basic 2005 Express Visual C# 2005 Express Visual C++ 2005 Express	Visual Basic 8	C# 2	- nová verze runtime - podpora generiky - plná podpora 64 bitů - výrazná vylepšení knihoven - vylepšení ASP.NET - parciální třídy - anonymní metody
3.0				- Windows Communication Foundation - Windows Presentation Foundation - Windows Workflow Foundation - Windows CardSpace
3.5	Visual Studio 2008 Visual Basic 2008 Express Visual C# 2008 Express Visual C++ 2008 Express	Visual Basic 9	C# 3	- nové verze VB.NET a C# - expression trees, lambda výrazy - LINQ - extension metody - podpora AJAXu v ASP.NET - Entity Framework
4.0	Visual Studio 2010	Visual Basic 10	C# 4	- dynamické datové typy - vylepšení stávajících technologií
4.5	Visual Studio 2012	VB.NET 11.0	C# 5.0	-asynchronní metody
4.5.1	Visual Studio 2013	VB.NET 11.0	C# 5.0	

(Zdroj: Vlastní úprava)

Pro správu paměti má .NET specifické třídy, ovšem není nutnost je používat. O automatickou správu se stará Garbage Collector, který z paměti odstraňuje objekty bez

reference, čímž uvolňuje místo a umožňuje v programu pracovat delší dobu bez nutnosti restartu.

2.2 Obsah .Net frameworku

Zde je jen stručný a nekompletní seznam toho, co .NET Framework obsahuje a jaké technologie v něm můžeme využívat. V prvním sloupci jsou názvy jmenných prostorů (funkce jsou řazeny po skupinách podle oblastí, které se týkají), ve druhém sloupci popis, co daný jmenný prostor obsahuje.

Tabulka 2 Příklady jmenných prostorů

System	Základní třídy a funkce
System.Addin	Třídy a funkce pro tvorbu pluginů a rozšíření aplikací
System.Collections	Implementace mnoha různých datových struktur a kolekcí
System.Configuration	Třídy a funkce pro práci s konfigurací aplikací
System.Data	Třídy a funkce pro práci s databázemi
System.Drawing	Pokročilé grafické funkce pro vykreslování nad rozhraním GDI+
System.Globalization	Třídy a funkce reprezentující národní zvyklosti a kulturně specifické záležitosti
System.IO	Třídy a funkce pro práci se souborovým systémem
System.Management	Třídy a funkce pro správu systému a zařízení (WMI)
System.Net	Třídy a funkce pro práci se sítí
System.Runtime.InteropServices	Třídy a funkce pro spolupráci s Windows API a COM
System.Runtime.Serialization	Třídy a funkce pro serializaci a deserializaci objektů
System.Security	Třídy a funkce pro zabezpečení a kryptografii (šifrování dat)
System.Text	Třídy a funkce pro práci s textem a regulárními výrazy
System.Threading	Třídy a funkce pro práci s vlákny
System.Web	Třídy a funkce pro tvorbu webových aplikací v technologii ASP.NET
System.Windows.Forms	Třídy a funkce pro tvorbu okenních aplikací
System.Xml	Třídy a funkce pro práci s XML

(Zdroj: Vlastní úprava)

V tabulce jsou uvedeny pouze některé jmenné prostory v rámci .NET Framework. Platforma kromě toho obsahuje několik dalších technologií, o kterých je dobré se zmínit:

ASP.NET – technologie určená pro vývoj webových aplikací nad .NET Frameworkem

Windows Communication Foundation – framework pro snadnou komunikaci aplikací nad různými komunikačními protokoly a technologiemi

Windows Presentation Foundation – framework pro vytváření bohatého hardwarově akcelerovaného uživatelského rozhraní s využitím různých grafických efektů

Windows Workflow Foundation – framework pro vývoj aplikací spravujících složité procesy

Windows Sync Framework – framework pro usnadnění synchronizace dat

Entity Framework – ORM mapper, který umožňuje reprezentovat data v databázi jako objekty

LINQ – syntaktické konstrukce v programovacích jazycích, které zjednodušují dotazování nad různými typy dat

2.3 Vnitřní fungování .NET

Pro přiblížení toho jakým způsobem platforma .NET pracuje je třeba definovat si tři základní vrstvy, představující mechanismus mezi sebou propojených částí.

Jako první si představme **CLR** (**Common Language Runtime**), které reprezentuje běhové prostředí a jehož úkolem je nalézt, nahrát do paměti a spravovat objekty. Tato součást je odpovědná za operace nízké úrovně jako přeložení z IL do strojového kódu, správa paměti, koordinace vláken, hostování aplikací a bezpečnostní kontroly.

Další součástí je **CTS**, neboli **Common Type System**, který definuje základní typy objektů a programové konstrukce podporované běhovým prostředím a jejich vzájemnou interakci. Také popisuje jak jsou tyto entity reprezentovány metadaty.

Aby spolehlivě fungovala možnost kooperace více knihoven napsaných v různých .NET jazycích, je třeba si vysvětlit co znamená zkratka **CLS**, jinak také **Common Language Specification**. CLS vymezuje podmnožinu entit z CTS, která splňuje podmínku zaměnitelnosti mezi jazyky, což v praxi umožňuje přepsat program psaný ve VB.NET např. v J# nebo C# bez obav ze ztráty funkčnosti nebo kompatibility.

2.4 Kompilace

Po zkompilování zdrojového kódu v daném programovacím prostředí vzniká soubor s příponami .exe nebo .dll, který však není přeložen do finální podoby pro cílovou platformu. Tento celek, zvaný assembly, se sestává z MSIL kódu (Microsoft Intermediate Language),

informací o typech neboli metadat, a manifestu, což jsou metadata popisující dané assembly. Tento soubor je pak, v závislosti na zvolené platformě dále přeložen do strojového kódu až ve chvíli, kdy je odkazováno na konkrétní blok instrukcí .NET běhovým prostředím. Nástroj, který toto umožňuje se nazývá JIT(Just-In-Time) kompilátor. Jeho specifické vlastnosti jsou přizpůsobení kompilace podle dostupné paměti a uložení kompilovaných částí kódu pro pozdější použití.

2.5 Struktura .NET assembly

Ať už budeme programovat okenní nebo konzolovou aplikaci, web či nějakou knihovnu, výsledkem bude jedna či více .NET assemblies, a to s příponou dll či exe. Assembly může být buď single-file, kdy je vše pohromadě, nebo multifile, kde assembly tvoří několik souborů.

Je nutné si uvědomit, že v druhém případě soubory Util.netmodule aGraphic.bmp nejsou součástí souboru assembly, ale jsou to oddělené soubory, které se k assembly propojují pouze informací v manifestu. Visual Studio standardně vytváří single-file assemblies.

2.5.1 Manifest

Manifest obsahuje mnoho důležitých informací o samotné assembly a jejím obsahu, konkrétně jméno assembly, číslo verze, výchozí kulturu(jazykové a národní nastavení), strong name (vysvětlíme později), seznam souborů patřících k assembly, informace o datových typech a implementacích jejich metod a informace o všech referencovaných assembly.

Referencované assemblies jsou takové assemblies, které obsahují datové typy, které naše assembly používá, ale sama je nedeklaruje. Určitě sem patří například základní knihovny .NET Frameworku, které obsahují definice vestavěných datových typů a které určitě každá naše assembly bude používat, proto potřebuje mít referencované knihovny, kde jsou tyto typy definovány.

2.5.2 Moduly

Modul je část assembly, která obsahuje kód a definice datových typů. Modul je víceméně zkompilovaný jeden soubor kódu, analogie např. objektových modulů v C/C++. Assembly vznikne slinkováním modulů dohromady linkerem. V praxi se však o toto nemusíme starat, vše za nás udělá vývojové prostředí Visual Studio

2.6 Programovací jazyky

Nad .NET Frameworkem existuje mnoho různých programovacích jazyků, zdaleka nejpoužívanější jsou ale Visual Basic .NET a C#. Tyto dva jazyky jsou si velmi podobné, přestože mají dost odlišný původ.

2.7 Visual Basic .NET

Jazyk BASIC vznikl zhruba v 60. letech minulého století a byl určen především pro výuku programování. Tento starý jazyk neměl funkce ani procedury, řádky se číslovaly a vzniklo mnoho jeho různých větví, z nichž žádná se moc neprosadila. Velké popularity se dočkal QBasic, který byl zaintegrován v systému MS-DOS, a který umožňoval psát strukturovaně – uměl procedury, funkce, struktury a datové typy. Pořád to ale byl jazyk interpretovaný, což vyřešil až QuickBasic.

S nástupem Windows se objevil Visual Basic. Napsat okenní aplikaci pro Windows v C/C++ nebylo právě jednoduché, zatímco Visual Basic umožnil díky svému vývojovému prostředí "naklikat" uživatelské rozhraní a dopsat kód. O mnoho let později tuto myšlenku převzala jiná vývojová prostředí, například Borland Delphi.

Poslední verzí klasického Visual Basicu byla verze Visual Basic 6 z roku 1998. Visual Basic jako takový již docela zaostával za konkurencí, nenabízel mnoho výhod a nebyl právě nejrychlejší. Dosáhl i tak ale velké obliby (zvláště na západě) a mnoho aplikací bylo napsáno právě v něm. Podporoval částečně objektově orientované programování, cílený byl především na databázové aplikace.

S nástupem platformy .NET se Microsoft rozhodl Visual Basic vzkřísit a vytvořil verzi Visual Basic 7 (což je verze jazyka, ne vývojového prostředí). Tento jazyk přinesl mnoho novinek, mnoho věcí se změnilo. Konečně se v jazyce objevila pořádná a kompletní podpora objektově orientovaného programování, díky .NETu jazyk dostal nepřeberné množství knihovních funkcí a zachována byla samozřejmě i kompatibilita s COM a Windows API voláními. Programy napsané ve VB.NET mohou být díky .NET Frameworku rychlejší, samozřejmě ale záleží na tom, jak je aplikace napsána. V současné době je aktuální verzí jazyka Visual Basic 9, která mimo generických typů a jiných vylepšení podporuje technologii LINQ.

2.8 Syntaxe VB.NET

Volba jazyka v němž bude program napsán je důležitým aspektem celého procesu tvorby aplikace. V případě aplikace vytvářené v této práci se setkáme se syntaxí kódu Visual Basic .NET, jehož typickou vlastností je používání klíčových slov oproti symbolům v jiných jazycích, což zvyšuje čitelnost kódu. Všechny názvy jsou case-insensitive, což znamená, že nezáleží na velikosti písmen v klíčových slovech, názvech proměnných, metod a typů.

Konec řádku je signifikantní znak, používá se k oddělování příkazů. Na rozdíl od většiny jazyků na konci příkazů nejsou středníky. Pokud chceme na jeden řádek dát více příkazů, můžeme, stačí je oddělit dvojtečkou. Pokud naopak chceme rozdělit příkaz do více řádků, použijeme sekvenci znaků mezera podtržítko. V nové verzi jazyka Visual Basic 10, která se chystá, se tato sekvence může vynechat, kompilátor podle kontextu pozná, jestli se jedná o konec řádku, nebo o jeden řádek rozdělený na více částí (kdy je možné řádky dělit je přesně definováno ve specifikaci a pravidla jsou to celkem jednoduchá a intuitivní).
Komentáře existují pouze jednořádkové, začínají znakem apostrof a končí společně s koncem řádku.

2.9 Struktura kódu

V jazycích VB.NET a C# nejsou globální funkce ani proměnné. Veškeré metody (tedy funkce a procedury) a proměnné musí být uzavřeny uvnitř nějakého datového typu. Datový typem myslíme jednu z těchto položek (ne ve všech mohou být metody a proměnné):

- třída
- rozhraní
- struktura
- výčtový typ (enum)
- delegát
- vestavěný typ

Každý typ je zařazen do nějakého jmenného prostoru (namespace). Jmenné prostory tvoří hierarchické rozdělení typů podle jejich funkce a oblasti použití. Například ve jmenném prostoru System se nachází základní datové typy (třeba Integer, String, DateTime) a třídy (např. třída Math obsahující matematické funkce atp.). Vše, co se týká grafiky, se nachází ve jmenném prostoru System.Drawing; vše, co se týká okenních aplikací, je ve jmenném prostoru System.Windows.Forms a podobně.

Všechny datové typy, které uvedeme dovnitř sekce namespace, budou zařazeny ve jmenném prostoru VbNet.Tutorials. V kódu můžeme používat všechny datové typy, které jsou ve stejném jmenném prostoru, nebo které jsou ve výchozím jmenném prostoru.

Pokud tedy uvnitř našeho jmenného prostoru nadefinujeme třídy A a třídy B, budeme se ze třídy A odvolávat na třídu B pouhým napsáním názvu třídy, tedy B. Pokud chceme použít typ z jiného jmenného prostoru, musíme se na něj odvolat celou cestou tohoto prostoru, například VbNet.C, pokud by třída C byla ve jmenném prostoru VbNet.

Psát celý název typu včetně jmenného prostoru je samozřejmě zdlouhavé, proto máme možnost si jmenné prostory naimportovat. To se dělá takto pomocí klíčového slova Imports resp. using, které platí pro celý soubor a píše se hned na začátek.

Ke zjištění využitelných typů, jejich členů a popisů spolu s ukázkovým kódem je možné využít webové stránky MSDN, což je vlastně internetový portál pro vývojáře veškerých produktů firmy Microsoft.

Nástrojů pro podporu jazyků .NET je mnoho, z freeware stojí za to zmínit SharpDevelop nebo jEdit, které nabízí zvýraznění syntaxe C# a kompilaci do IL, ovšem podpora VB.NET tu není. Existují i webové konvertory kódu mezi .NET jazyky, které lze použít, ovšem největší smysl má využití oficiálního vývojového prostředí jak z hlediska funkčnosti tak z hlediska podpory.

2.10 Microsoft Visual Studio

Jde o nejrozšířenější IDE neboli vývojové prostředí pro tvorbu v .NET obsahuje velkou škálu nástrojů umožňujících efektivní tvorbu nejen databázových aplikací. Abyste mohli vyvíjet aplikace běžící na .NET Frameworku, potřebujete vývojové prostředí. Microsoft se o platformu .NET stará velice dobře a vývojové prostředí s bohatou dokumentací dává k dispozici zdarma.

Pro vývoj v .NET Frameworku můžete využít Visual Studio Professional, což je plnohodnotné profesionální vývojové prostředí, které podporuje programovací jazyky VB.NET, C# a C++. Tento produkt je placený a obsahuje kromě základních funkcí i pokročilé nástroje pro testování aplikací, profilování a týmový vývoj.

Kromě komerčního Visual Studia Professional Microsoft uvolnil i jeho odlehčené verze (Express edition) pro nekomerční i komerční užití zdarma. Obsahují skoro vše až pro nástroje pro týmový vývoj a testování, ale to stejně ze začátku nevyužijete, jediné znatelnější omezení

je, že nemáte jeden produkt pro všechny jazyky, ale pro každý programovací jazyk je samostatná aplikace.

- Zde je několik základních nástrojů, které prostředí obsahuje:
- Object Browser – výčet dostupných knihoven a jejich obsahu
- Toolbox – obsahuje komponenty pro stavbu aplikací typu Windows Forms, ASP.NET
- Solution Explorer – seznam položek v projektu, jako jsou třídy, moduly, komponenty, složky, datové zdroje, atp.
- Server Explorer – prohlížení seznamu serverů a datových spojení
- Properties – vlatnosti daného objektu a jejich hodnoty
- Error List – zobrazuje chyby znemožňující kompilaci
- Locals – jen v módu Debug, umožňuje sledovat obsah proměnných za běhu programu
- Code Analysis - návrhy pro zlepšení kódu

2.11 Výběr projektu

Pro uživatelsky přívětivé řešení je vhodné vybrat projekt s vizuální prezentací, v níž se snadno orientuje a je čitelná i pro uživatele s minimální zkušeností práce na daném zařízení, ať už PC nebo jiném. Pro tento účel je vhodná aplikace Windows Forms, jejíž rozhraní je totožné s nativním rozhraním Windows Explorer.

3. Seznámení s databází typu MySQL

Existuje několik základních typů systémů pro správu databází, zkráceně DBMS (Database Management System). Nejjednodušší je prostý databázový soubor, většinou ve formě textu, bez větších možností manipulace s daty. Následuje hierarchický systém se vztahy rodič-potomek ve stromové struktuře, který se však příliš nehodí pro aplikaci na operace z reálného světa. Dalším typem DBMS posouvajícím možnosti vpřed je síťový databázový systém, který se od hierarchického liší hlavně možností vazeb M:N a již pro manipulaci s daty uplatňuje SQL (Structured Query Language), tedy jazyk typický pro dnešní databáze. Tento systém byl však uplatňován hlavně v letech 60. a 70. letech minulého století a dnes, kdy byl nahrazen relačním modelem, je již málokdy využíván. Relační systém je specifický využitím cizích klíčů pro zachování konsistence databáze. Posledním typem je objektově relační systém, držící se zásad objektově orientovaného paradigmatu, v jehož modelu je k objektům jako datovým celkům přistupováno skrze ukazatele. *(Zdroj: 1)*

V případě MySQL jde o relační databázový systém, stavějící na velkém množství funkcí (jak v SQL dotazech tak na samotném databázovém serveru, modifikace dotazů, manipulace s tabulkami, aj.), velmi dobré rychlosti vykonávání dotazů a v neposlední řadě také možnosti provozovat bezplatnou verzi serveru díky licenci GNU. Pro databázovou aplikaci je klíčová podpora propojení serveru s klientskou aplikací a možnosti převodu zpracování dat mezi nimi. Příkladem tohoto může být vytvoření dočasné tabulky na serveru, výpočet dat na klientském počítači a uložení do této tabulky, nad kterou lze již vytvářet dotazy a dále s ní pracovat.

3.1 Structured Query Language

Je třeba definovat uživatelský prostředek, kterým bude sloužit k manipulaci s daty v databázi. Tímto prostředkem je programovací jazyk Structured Query Language, zkráceně SQL, který je standartním jazykem pro relační databázové systémy. Jazyk jako takový vznikl ve snaze odstranit nutnost znalosti programátorů skutečného fyzického uspořádání dat v databázi. Díky SQL není třeba znát množství bytů v záznamu nebo stopovací značky, stačí vědět jaké tabulky jsou v databázi a jak jsou mezi sebou propojeny. Dělí se do kategorií, v názvu popisujících jakých operací se příkazy v dané kategorii týkají.

DDL – Data Definition Language – obsahuje příkazy pro vytváření a změny databází, tabulek či jiných databázových objektů pomocí příkazů CREATE, ALTER, TRUNCATE a DROP, určuje a mění jejich vlastnosti, definuje vztahy mezi nimi

DML – Data Manipulation Language – zde jsou příkazy pro manipulaci se samotnými daty, uloženými v tabulkách. Příkazy jsou INSERT pro vložení, UPDATE pro změnu a DELETE pro smazání. Pro transakce slouží příkazy BEGIN, COMMIT a ROLLBACK

DCL – Data Control Language – je důležitá kategorie z hlediska bezpečnosti, obsahující příkazy pro přidělování a odebírání uživatelských práv GRANT a REVOKE

3.1.1 Architektura MySQL Serveru

MySQL má vlastnost díky, které umožňuje vybírat z mnoha specializovaných storage engine (dále SE) pro danou aplikaci, aniž by v samotné aplikaci bylo třeba měnit kód. Architektura MySQL serveru umožňuje izolovat programátora databázové aplikace a administraci databáze od veškerých detailů implementace nízké úrovně na úrovni ukládání, čímž poskytuje konsistentní datový model. Pokud jsou tedy rozdílné vlastnosti napříč několika různými SE, aplikace žádný rozdíl nepozná. V této architektuře je společný standartní soubor řídících a podpůrných služeb všem SE, které jsou jeho součástí. Jako takový je SE komponentou databázového serveru, která provádí reálné operace nad daty udržovanými na fyzické úrovni serveru. Výhody této architektury v praxi jsou znatelné zejména, při požadavcích aplikace na změnu SE, nebo aby v rámci rozšíření funkčnosti aplikace byl přidán jeden či více SE.

Díky zásuvným SE, zodpovědným za veškeré vstupně-výstupní operace s daty a zajištění specifického souboru funkcí je možné soustředit se na konkrétní požadavky aplikace. Ve výsledku lze správným výběrem SE snížit režijní nároky databáze, v důsledku čehož pak zefektivní běh databáze a navýší rychlost zpracování dotazů. Podívejme se na některé odlišnosti jednotlivých SE, definující rozdíly v infrastruktuře tabulek:

- Souběžnost – u některých typů databázových aplikací je vyžadováno zamykání na úrovni řádků, zejména z důvodu možnosti manipulace více uživatelů s tabulkou. Proto je třeba zvolit správný způsob zamykání, jež může mít také vliv na režii a tudíž celkový výkon databáze.

- Transakčnost – pokud je u aplikace třeba spolehlivého zpracování více příkazů jako jednoho celku, je dobré využít SE podporujícího standard ACID či jiný, zajišťující spolehlivé provedení transakce
- Referenční Integrita – vynucení referenční integrity relační databáze skrze systém cizích klíčů definovaných v DDL (Data Definition Language) je vhodný pro zajištění konsistence dat bez nutnosti vykonávat více dotazů a zatěžovat tak server připojeními
- Fyzické uložení – zde jsou zahrnuty velikosti stránkovacích souborů tabulek a indexů, a také použitý formát, v němž jsou data uložena na disku
- Podpora Indexů – různé možnosti aplikačních řešení těží z rozmanitých indexačních strategií. Každý SE má zpravidla své vlastní indexační metody, ačkoli některé jako B-Stromový index jsou společné téměř všem SE
- Paměťové caches – opět zde záleží na konkrétní aplikaci a její reakci na způsob ukládání do paměti. Jisté vybrané caches jsou společné všem SE (např. ty které spravují uživatelská připojení či MySQL high-speed query cache), jiné unikátní na úrovni jednotlivých SE.
- Výkonnostní podpora – sem patří vstupně-výstupní vlákna pro paralelní operace, souběh vláken, databázové mezistavy, zpracování dávkových insertů, atd.
- Dodatečné cílové funkce – podpora pro operace systému geologických souřadnic, bezpečnostní omezení určitých funkcí pro manipulaci s daty, a jiné

3.2 Storage Engine

Základem databází v systému MySQL je tato kritická součást databázového systému, která hraje klíčovou roli při návrhu databáze. SE je softwarový modul, který je databázovým systémem využíván k tvorbě, čtení a změně (mazání) dat. Základní rozdělení v MySQL vyčleňuje dva typy SE – transakční a netransakční.

Tabulky transakční nebo také transakčně bezpečné nabízí oproti netransakčním tabulkám řadu výhod. Pokud se zajímáme o bezpečí dat, transakce zde mají výhodu.
V případě, že dojde k pádu MySQL nebo se vyskytnou hardwarové problémy, lze se dostat zpět ke ztraceným datům díky automatické obnově ze zálohy spolu s logem transakcí. Také lze zkombinovat několik příkazů a vykonat je ve stejnou chvíli pomocí příkazu COMMIT

(případně jde zapínat a vypínat tuto funkci automaticky). Změny, ke kterým mělo v transakci dojít mohou být podmíněně ignorovány příkazem ROLLBACK. K němu také dojde automaticky, selže-li z různých důvodů (timeout pro transakci, přerušení spojení) vykonání alespoň jednoho z příkazů v transakci. Transakčně-bezpečné SE poskytují také lepší souběh pro tabulky, na nichž dochází k mnoha zápisům a čtením současně.

Co se týče netransakčních tabulek, je třeba se zmínit o některých výhodách, které nabízejí oproti tabulkám transakčním. Všechny tyto výhody vyplývají z faktu, že u příkazů na netransakční tabulky není třeba vyčlenit žádné systémové prostředky na režii. Jednou z těchto výhod je rychlost vykonání příkazů, která je zde podstatně vyšší než u příkazů v transakcích. Požadavky prostoru na pevném disku jsou také sníženy, stejně jako menší potřebné množství paměti u příkazů pro vykonávání změn v tabulce.

V případě potřeby lze kombinovat transakční a netransakční tabulky v rámci jednoho příkazu pro získání co nejlepších výsledků. Nicméně i přes podporu MySQL mnoha transakčně bezpečných SE se nedoporučuje kombinovat různé SE uvnitř jedné transakce s vypnutou automatickou funkcí COMMIT. Pokud totiž dojde k vykonání transakce, změny k nimž došlo v tabulce nepodporující transakce budou permanentní bez možnosti rollbacku.

3.3 Jednotlivé typy SE

Výběr specifických SE může probíhat na několika úrovních – serverové úrovni, databázové úrovni a na úrovni jednotlivých tabulek. Díky systému úrovní lze správnou volbou vyladit možnosti tabulek a docílit individuální funkčnosti. Pro představu z jakých SE lze volit, zde je několik základních typů:

- MyISAM – výchozí SE až do verze MySQL 5.5, kdy byl nahrazen InnoDB. Je využíván zejména pro web databáze, datové sklady, a jiná aplikační prostředí
- InnoDB – je SE pro všeobecné účely, který vyvažuje vysokou spolehlivost a vysoký výkon
- Merge – umožňuje vývojáři či administrátorovi logicky shlukovat identické MyISAM tabulky a odkazovat na ně jako na jeden objekt, čehož se využívá ve VL (very large - velkých) databázích
- Memory – uchovává veškerá data v paměti RAM pro rychlé přístupy v prostředí, kde je třeba rychle vyhledat méně kritická data. Vzhledem

k paměťovému prostoru bufferů u InnoDB je však tento SE využíván stále méně

- Federated – nabízí možnost propojit více MySQL serverů k vytvoření jedné logické databáze z více fyzických serverů. Využití v distribuovaných prostředích a datových tržištích
- Example – tento engine slouží jen jako ukázka v MySQL zdrojovém kódu jak napsat vlastní tabulkový engine. Je zajímavý zejména pro vývojáře. Lze v něm vytvářet tabulky, do nichž ale nelze ukládat data
- Archive – kompaktní a neindexované tabulky v tomto engine jsou určeny ke skladování a získávání velkých množství málokdy odkazovaných dat, jako jsou historické informace či výsledky bezpečnostních auditů
- CSV – tyto tabulky jsou ve skutečnosti textové soubory, v nichž uložené hodnoty jsou jako textové řetězce či znaky oddělené středníkem. Lze pomocí nich importovat nebo exportovat data ve formátu CSV a vyměňovat tyto data se skripty a aplikacemi, které tento formát podporují. Vzhledem k absenci indexů se obvykle používají jen při vstupu nebo výstupu dat z tabulek v jiném engine, např. InnoDB
- Blackhole – speciální druh SE, který přijímá data ale žádná nevrací. Dotazy na tabulky v tomto engine vrací vždy prázdnou množinu. Používají se při replikaci, kdy DML příkazy jsou posílány slave serverům, přičemž master server nezachovává vlastní kopie dat

SE uvedené výše jsou jen některými ze seznamu běžně dostupných, v případě potřeby má MySQL Server příkaz pro zjištění, jaké SE lze na dané instalaci použít. Z popisů některých SE je patrné, že je běžnou praxí zkombinovat více tabulek s různými vlastnostmi v jedné databázi a v případě, že žádný typ SE nevyhovuje konkrétnímu požadavku, je možné vymyslet a implementovat vlastní. V této práci se však dále budeme soustředit na dva konkrétní SE, kterými jsou MyISAM a jeho nástupce InnoDB.

První byl engine MyISAM, vycházející z kódu ISAM obohaceným o některá rozšíření. Tabulka MyISAM je na disku rozložena do tří souborů, z nichž každý má příponu napovídající typ souboru. V souboru s příponou .frm je uložen formát tabulky, soubor kde jsou uložena

data má koncovku .myd a posledním definičním souborem je ten, který obsahuje indexy, mající příponu .myi. Jedním ze specifických znaků je ukládání datových hodnot s nulovým prvním bytem, což zajišťuje možnost kopírovat data nezávisle na různých operačních systémech. Naopak všechny číselné klíčové hodnoty jsou ukládány s jednotkovým prvním bytem, čímž je docíleno lepší komprese indexů. Zajímavou vlastností je také souběžné vkládání dat u více vláken, kdy v případě že tabulka nemá žádné volné bloky (způsobené např. smazáním řádku) v datovém souboru, dají se do datového souboru vložit nová data, zatímco v tabulce dochází ke čtení. Ve scénáři, kdy je třeba plánovat volné místo se dá využít vlastnost rozdělení souborů s indexy a datových souborů do různých složek, která se nastavuje při vytváření tabulky. Za zmínku ještě stojí booleovský atribut v indexačním souboru, který ukazuje, zda-li byla tabulka správně ukončena a slouží k určení, zda se mají indexy v tabulce opravit.

Nejpoužívanější SE v rámci MySQL je v dnešní době InnoDB, který svého předchůdce překonává ve většině ohledů, jak si ukážeme ve srovnání. Organizace tabulek je zde v modifikovatelné formě s možnostmi uložení tabulky a jejích indexů do jednoho souboru nebo uložení více tabulek s indexy do jednoho tablespace, který může obsahovat jednu či více tabulek. Způsob zpracování příkazů je zajišťován v rámci transakčního ACID modelu, zajišťujícího atomicitu (transakce mají jen stavy úspěšně provedena a neprovedena), konsistentnost (zajištění konsistentnosti tabulky za všech okolností), izolace (oddělení jednotlivých transakcí skrz systém uzamykání) a trvanlivost dat (při úspěšném provedení transakce jsou provedené změny zachovány za všech okolností, jako např. výpadek proudu nebo pád systému). Systém uzamykání tabulky probíhá na úrovni jednotlivých řádků, což má výhody zejména pokud příkazy spojené s danou tabulkou ovlivňují pouze jeden či dva řádky, v případě více řádků se zpracování příkazu značně zpomaluje. Tabulky InnoDB uzpůsobují data na disku s ohledem na zefektivnění často používaných dotazů založených na primárním klíči. Každá tabulka má svůj index, zvaný clustered index, který organizuje data pro minimalizaci vstupů a výstupů spojených s hledáním primárního klíče. Pro udržení datové integrity umožňuje tento SE také závislosti cizích klíčů, což je vlastně realizace relací. Díky cizím klíčům lze synchronizovat operace v odkazující a závislé tabulce, kdy se příkazy UPADTE či DELETE v odkazující tabulce provedou i na příslušných řádcích v tabulce závislé. Jelikož jsou cizí klíče automaticky indexovány, dotazy SELECT u příslušných tabulek probíhají velice rychle.

Tabulka 3 Srovnání MyISAM a InnoDB storage engine

	MyISAM	InnoDB		MyISAM	InnoDB		MyISAM	InnoDB
Max. velikost úložiště	64 TB	256 TB	Podpora transakcí	Ne	Ano	Úroveň zamykání	Tabulka	Řádek
Multiversion kontrola souběhu	Ne	Ano	Podpora systému geolog. souřadnic	Ano	Ano	Podpora indexování geolog. souřadnic	Ano	Ano
B-Strom indexy	Ano	Ano	T-Strom indexy	Ne	Ne	Hash indexy	Ne	Ne
Indexy Full-textového vyhledávání	Ano	Ano	Clustered Indexy	Ne	Ano	Datové caches	Ne	Ano
Indexové cache	Ano	Ano	Podpora komprimace dat	Ano	Ano	Podpora cizích klíčů	Ne	Ano
Podpora cluster databází	Ne	Ne	Podpora query cache	Ano	Ano	Aktualizace statistik pro datový slovník	Ano	Ano

(Zdroj: 3)

3.4 Datový slovník

Každá relační databáze obsahuje nějakou formu datového slovníku či systémového katalogu. Datový slovník je systémová oblast uvnitř databázového prostředí, která obsahuje informace o složkách databáze. Datové slovníky obsahují informace o návrhu databáze, uloženém kódu jazyka SQL, uživatelských statistikách, databázových procesech, růstu databáze a výkonových statistikách databáze. K naplnění tabulek datového slovníku dochází již při provádění SQL příkazů z kategorie DDL, a ukládají se do nich informace o tabulkách - o jejich atributech, sloupcích, velikostech, právech a růstu. Mezi další objekty uložené v datovém slovníku patří indexy, spouštěče, procedury, balíčky a pohledy.

V MySQL je datovým slovníkem virtuální databáze s názvem INFORMATION_SCHEMA. Pro zobrazení informací relevantních k dané námi vytvořené databázi složí příkazy SHOW, což jsou vlastně příkazy SELECT z tabulek obsahujících příslušné informace, které nás zajímají. Z hlediska SE je INFORMATION_SCHEMA typ Memory, což znamená, že při ukončení MySQL Serveru jsou všechny tabulky v datovém slovníku smazány. Po restartu jsou znovu

vytvořeny jako dočasné tabulky a obsazeny metadaty pro jednotlivé tabulky v uživatelské databázi.

Tabulka 4 Příklady tabulek z INFORMATION_SCHEMA databáze

Název Tabulky	Obsažené informace
CHARACTER_SETS	Informace o dostupných znakových sadách
COLUMNS	Informace o sloupcích v tabulkách, jako např. v jaké jsou tabulce, nastavení práv pro sloupce, název sloupce, datový typ, atd.
ENGINES	Informace o storage engines
PARTITIONS	Informace o tabulkových partitions, pokud je tabulka rozdělena do více částí
PROCESSLIST	Velmi užitečná tabulka zobrazující procesy, jsou zde vidět jednotlivá vlákna i s dobou otevření přípojení
ROUTINES	Obsahuje informace funkcích a uložených procedurách, jejich parametrech, typu, názvu, atd.
SCHEMATA	Informace o databázích
TABLES	Informace o tabulkách – engine, počet řádků, délka indexů, jméno, atd.
TRIGGERS	Informace o spouštěčích (triggerech), podmínkách spuštění, starých/nových hodnotách řádcích a tabulkách, atd.

(Zdroj: Vlastní úprava)

3.5 MySQL programy

Programů nebo také programových nástrojů spojených s během a údržbou MySQL databáze je celá řada. Tyto jsou ovládány přes příkazový řádek jazykem shell a lze u nich nastavit parametry dle specifických požadavků nastavení. Příkladem za všechny je program mysqld, což je vlastně klíčová aplikace obstarávající většinu serverových funkcí, včetně přístupu do datového adresáře, obsahujícího databáze a tabulky. Programové nástroje jsou rozděleny do několika kategorií:

- Programy pro vytvoření a zapnutí jednoho serveru či více serverů na jednom stroji
- Programy spojené s instalací serveru, např. pro správu zásuvných modulů, generování chybových hlášení, inicializace datového adresáře, a jiné
- Klientské programy, mezi které patří ty pro zálohu databáze, importování dat, zobrazení informací o databázi a jejích členech, mysql příkazový řádek, atd.
- Administrativní programy a programy pro údržbu, jako například pro generování komprimovaných MyISAM tabulek, konfigurační editor, programy pro ukončení procesů nebo pro kontrolu uživatelských práv, atd.

- Vývojové nástroje pro konverzi mSQL programů k použití s MySQL nebo pro zobrazení informací pro kompilaci databázové aplikace
- Ostatní programy zahrnující program vysvětlující chybové hlášky či program pro nahrazování řetězců

3.6 MySQL Workbench

Grafický nástroj pro práci s MySQL databázemi, využívající programových nástrojů MySQL ke zjednodušení a zpřehlednění návrhu, modelování, vývoje, migrování a správy databází. Po připojení se na specifický server je možné buď přímo psát sql kód do mysql příkazového řádku anebo vytvářet a spravovat databáze čistě graficky v okně navigator, to vše na základě přidělených práv pro daného uživatele. Při dostatečných oprávněních je také možné sledovat a spravovat uživatelské účty či sledovat zatížení serveru. Zejména zajímavá je možnost zpětného inženýrství (reverse engineering) databáze, která generuje diagram tabulek a jejich relací z příslušné databáze.

4. Možnosti propojení platformy .NET a MySQL databáze

Platforma .NET definuje množství jmenných prostorů, jež umožňují komunikovat s relačními databázovými systémy. Tyto jmenné prostory jsou souhrnně označovány jako ADO.NET. Díky tomu je umožněna podpora spousty datových poskytovatelů, z nich každý je přizpůsoben pro komunikaci se specifickým databázovým systémem. Zkratka ADO (Active Data Objects) byla používána již v souvislosti s COM modelem k přístupu dat, ovšem s ADO.NET má společný jen koncept objektů Connection a Command, přičemž objekt jako recordset v ADO.NET již není. Naopak je zde mnoho nových objektů, které ve starším modelu ADO nemají ekvivalent. Rozdíl je ale zejména v myšlence na níž byl ADO.NET vybudován, což je systém, založený na typu DataSet. Tento typ reprezentuje objekt, který obsahuje místní kopie příslušných tabulek z databáze, a každá z tabulek obsahuje kolekce sloupců a řádků. S využitím DataSetu je volající assembly (např. webová stránka nebo aplikace typu exe) schopno manipulovat s DataSetem a měnit jeho obsah, přičemž není ve spojení s databází. K odesílání modifikovaných dat z DataSetu do databáze pro zpracování pak dochází přes spřízněný data adapter.

Z pohledu programátora je jádro ADO.NET reprezentováno základním assembly s názvem System.Data, které jsme si představili v úvodní části seznámení s .NET platformou. Uvnitř této knihovny je množství jmenných prostorů, které specifikují konkrétní typy datových poskytovatelů. Většina šablon projektů, jako např. Windows Forms již v základu tuto knihovnu importuje, takže pro použití DataSetů není třeba ji přidávat do referencí projektu. V případě komunikace s MySQL databází je však třeba přidat knihovnu Mysql.Data, zprostředkující ADO.NET datové poskytovatele pro MySQL.

ADO.NET lze pro spojení s databází použít třemi unikátními způsoby, a to přes spojenou vrstvu, odpojenou vrstvu a Entity Framework (EF). Při použití spojené vrstvy dochází ke spojení a odpojení databáze přes explicitní definice v kódu. V tomto případě jsou k příslušným operacím s databází využívány Connection, Command a Data Reader objekty. Odpojená vrstva umožňuje manipulovat s množinou objektů typu DataTable (uloženou v objektu DataSet), která funguje jako kopie tabulek z databáze na straně klienta. Zde je rozdíl ve způsobu připojování, kdy objekt Connection, zajišťující připojení, je pro otevření a zavření připojení volán implicitně při získávání dat objektem data adapter. Tento přístup umožňuje rychle uvolňovat připojení pro ostatní uživatele a je zejména užitečný při možném

zvětšování databáze. Poté co je DataSet získán je připojení ihned uzavřeno a další manipulace s daty lze provádět bez jakékoli síťové komunikace. Pokud klient chce použít změny provedené v DataSetu, je znovu volán data adapter, který otevře připojení, zjistí rozdíly mezi databází a DataSetem, provede příslušné SQL příkazy pro změnu a ihned po provedení změn připojení zavře. Tohoto způsobu komunikace je využíváno i ve třetím způsobu užívání ADO.NET, kterým je EF. Ten přidává zapouzdření databázových příkazů do metod silně typových DataSetů a použití příkazů LINQ.

4.1 Datoví Poskytovatelé

Jak již bylo zmíněno na začátku této části, existuje více knihoven v rámci VB.NET, z nichž každá nabízí množinu datových poskytovatelů uzpůsobenou pro komunikaci s konkrétními DBMS. Jedna výhoda tohoto přístupu je, že lze naprogramovat specifického datového poskytovatele pro přístup k unikátním funkcím, jež daný DBMS nabízí. Druhá výhoda je, že se poskytovatel může přímo na příslušný SE zmiňovaného DBMS bez nutnosti zprostředkující mapovací vrstvy. Jednoduše lze říci, že datový poskytovatel je souborem typů v určitém jmenném prostoru, který ví jak komunikovat se specifickým typem datového zdroje. Nehledě na použitého datového poskytovatele je v každém definována množina typů tříd zajišťujících základní funkčnost.

Tabulka 5 Objekty pro komunikaci s databázemi z assembly System.Data

Název objektu	Základní třída	Související rozhraní	Popis
Connection	DbConnection	IDbConnection	Poskytuje možnost připojení do a odpojení od databáze. Objekty Connection také poskytují přístup k souvisejícím objektům transakcí
Command	DbCommand	IDbComand	Reprezentuje SQL příkaz či dotaz, nebo uloženou proceduru. Také poskytuje přístup poskytovatele k Data Readeru
DataReader	DbDataReader	IDataReader, IDataRecord	Umožňuje číst data pouze dopředným přístupem, využívajícím kurzor na straně serveru
DataAdapter	DbDataAdapter	IDataAdapter, IDbDataAdapter	Stará se o transfer DataSetů mezi klientem a databází. Součástí je objekt Connection a soubor čtyř vnitřních objektů, obsahujících odpovídající DML příkazy – select, insert, update a delete
Parameter	DbParameter	IDataParameter, IDbDataParameter	Reprezentuje pojmenovaný parametr uvnitř parametrického příkazu
Transaction	DbTransaction	IDbTransaction	Zapouzdřuje databázovou transakci

(Zdroj: 5)

4.2 Spojená vrstva

Při použití této vrstvy jsou klíčovými objekty datového poskytovatele objekty Connection, Command a Data Reader. Samotný proces komunikace se skládá z několika kroků, které je třeba vykonat pro spojení z databází a přečtení dat pomocí Data Reader objektu:

- Alokace (vyčlenění paměti), konfigurace a otevření našeho objektu Connection

- Alokace a konfigurace objektu Command, specifikující objekt Connection jako argument konstruktoru nebo jako vlastnost s názvem Connection
- Volání metody ExecuteReader() v nakonfigurované třídě Command
- Zpracování každého záznamu s použitím Read() metody Data Reader objektu

S výstupem z Data Readeru můžeme pracovat na úrovni jednotlivých záznamů, kde v rámci cyklu podmíněném návratovou hodnotou Read() metody lze záznamy např. ukládat do pole či tabulky.

4.2.1 Connection objekt

Objekt, bez kterého by nemohlo ke spojení s datovým zdrojem (databází) dojít se nazývá Connection, a slouží k vytvoření seance (session) mezi klientem a datovým zdrojem. Objekty pro připojení u .NET platformy mají formátovaný řetězec Connection string pro připojení, který obsahuje množství párů typu název/hodnota oddělených středníkem. Tato informace slouží k identifikaci stroje, ke kterému se chceme připojit, nezbytná bezpečnostní nastavení, název cílové databáze, a jiné specifické informace pro datového poskytovatele.

Klíčové metody:

- Open() – metoda zodpovědná za otevření připojení a vytvoření seance
- Close() – slouží k ukončení seance a zavření připojení
- Dispose() – odstranění instance objektu z paměti
- BeginTransaction() – zahájení transakce s databází
- ChangeDatabase() – změna databáze, se kterou komunikujeme
- GetSchema() – metoda vrací objekt data table, obsahující informace o schématu databáze

Klíčové vlastnosti:

- ConnectionTimeout – vrací dobu, po kterou se snaží klient navázat spojení s datovým zdrojem. Po jejím uplynutí již pokus o navázání neopakuje a zahlásí chybu
- Database – obsahuje název databáze s níž je ve spojení
- DataSource – zde je umístění serveru ve formě IP adresy nebo URL názvu serveru
- State – indikuje stav připojení

Protože jsou vlastnosti objektu Connection typicky pouze pro čtení, jsou užitečné jen jako informace o charakteristikách připojení za běhu programu. Pokud chceme tyto vlastnosti změnit, je třeba tyto změny provést v Connection stringu.

4.2.2 Command objekt

Již sme si představili, co je třeba provést pro úspěšné navázání spojení s databází. Nyní můžeme začít předávat příkazy či dotazy, podle toho zda chceme databázi upravovat nebo z ní vybírat. Pro tento účel slouží objekt Command, který se dá označit jako objektově orientované vyjádření SQL kódu, jež může být kromě příkazu či dotazu i název tabulky nebo uložené procedury. Při vytváření objektu Command lze vložit příkaz SQL jako parametr konstruktoru spolu s připojením, nebo ho nastavit přímo jako vlastnost objektu.

Klíčové metody:

- Cancel() – zruší vykonání příkazu
- ExecuteQuery() – vykoná SQL dotaz, tedy select
- ExecuteReader() – vykoná SQL dotaz a vrátí objekt typu Data Reader příslušného datového poskytovatele, z něhož je možné číst data dopředným směrem
- ExecuteNonQuery() – vykoná jeden z SQL příkazů, kterým může být insert, update, delete nebo příkaz z kategorie DDL
- ExecuteScalar() – tato metoda složí k vykonání dotazu, který vrací pouze jednu hodnotu
- Prepare() – vytvoří připravenou (či zkompilovanou) verzi příkazu pro datový zdroj, která je vykonána rychleji než nezkompilovaná verze a slouží zejména pro často opakované příkazy s jinými parametry

Klíčové vlastnosti:

- CommandType – specifikuje, jak bude interpretována vlastnost CommandText
- CommandText – obsahuje ve formátu string kód SQL, který se má vykonat
- Transaction – specifikuje instanci transakce, ve které se jeden či více příkazů vykoná
- Connection – vrátí nebo přiřadí instanci objektu Connection používanou touto instancí objektu Command

- Parameters – vrací kolekci objektů typu parameter použitých pro příkazy s parametry
- CommandTimeout – nastaví nebo vrátí čas čekání na vykonání příkazu, po jehož uplynutí ukončí pokusy a vrátí chybovou hlášku

Použití parametrů v objektu Command je běžná praktika, která slouží ke zvýšení bezpečnosti databázové aplikace, zejména ochranou před útoky typu SQL injection. Ty spočívají v nežádoucí modifikaci vlastnosti CommandText v místech uživatelských vstupů, např. v komponentách TextBox a v některých případech mohou mít za následek např. mazání nebo modifikaci řádků, tabulek i celých databází. Tomuto lze předejít přidáním parametrů v místech uživatelských vstupů.

4.2.3 Data Reader Objekt

Po navázání aktivního připojení a odeslání SQL dotazu je třeba získat výsledek dotazu od datového zdroje. Způsobů jak toho docílit je více, z nichž nejrychlejší a nejjednodušší je použití objektu Data Reader. Ten vrací výsledek dotazu jako dopředný tok dat, který čte záznam po záznamu. Jak je patrné, je využíván pouze v případě, kdy databáze vrací výsledek dotazu, tedy výsledek příkazu select v SQL kódu. Data Reader je užitečný pokud je třeba rychle procházet velké množství dat bez nutnosti je udržovat v paměti. Jako příkladem může být uložení řádově desítek tisíc záznamů z tabulky do textového souboru, které by bylo v případě uložení do DataSetu velice náročné (DataSet uchovává v paměti celý výsledek dotazu). K získání objektů Data Reader z objektu Command dochází pomocí metody ExecuteReader(). Každý takový objekt reprezentuje aktuální přečtený záznam z databáze. Data Reader má indexační metodu, která umožňuje přistupovat ke sloupci aktuálního záznamu, což lze přes název sloupce nebo jeho ordinální číslo pozice. Při čtení záznamů je po každém přečteném záznamu volána metoda Read(), která určuje zda-li šlo o poslední záznam (návratovou hodnotou false). Je však třeba zmínit, že objekty Data Reader udržují otevřené připojení k datovému zdroji dokud není explicitně zavřeno.

4.2.4 Transakce

Transakce probíhají voláním metody BeginTransaction() instance Connection objektu, která vrací objekt SQLTransaction. V rámci tohoto objektu lze pak mezi jednotlivými metodami pro vykonání příkazů objektu Command určovat, jak se má transakce chovat. Toto

chování je definováno metodami Rollback(), Save() a Commit().Při volání metody Rollback() dojde ke zrušení celé transakce a návratu databáze do původního stavu. Metoda Save() umožňuje definovat záchytné body do kterých se transakce vrací v případě selhání, aby nedošlo ke zrušení všech změn. Pro potvrzení hromadného vykonání všech příkazů v transakci se volá metoda Commit().

4.3 Odpojená vrstva

Tato vrstva pracující na úrovni klienta umožňuje modelovat data v paměti s použitím členů jmenného prostoru System.Data (zejména jde o členy DataSet, DataTable, DataRow a DataColumn). Pro uživatele připojeného k databázi přes databázovou aplikaci vytváříme tímto přístupem iluzi, že je neustále ve spojení s databází, přičemž ve skutečnosti pracuje s místní kopií relačních dat. Existuje i možnost využívat odpojenou vrstvu ADO.NET, aniž by došlo ke spojení s databází, pokud by například byl požadavek udělat program s modifikovatelným GUI a výstupy nebo by nevyhovovala žádná z dostupných aplikací pro účely uchovávání dat. V případě databázové aplikace jsou data získávána naplněním objektu DataSet pomocí objektu data adapter příslušného datového poskytovatele. Data adapter tak funguje jako můstek mezi klientskou aplikací a databázovým systémem. Po naplnění DataSetu je pak možné manipulovat s jeho obsahem, měnit řádky v tabulkách a výsledky změn posílat opět pomocí data adapteru zpět. S obsahem DataSetu pak také úzce souvisí datové svazování (data binding) tabulek s komponenty pro uživatelskou interakci Forms controls.

Typy obsažené v System.Data jmenném prostoru umožňují kromě tabulkového systému sloupců a řádků také reprezentovat relace mezi jednotlivými tabulkami, závislosti sloupců, primární klíče, pohledy, a jiné specificky databázové typy. Po vytvoření datového modelu je možné aplikovat filtry, provádět příkazy nad tabulkami v paměti, a konvertovat data do formátu XML nebo jiných. To lze udělat i bez připojení na databázi nahráním dat z XML souboru či ručním vytvořením DataSetu v kódu.

4.3.1 DataSet objekt

Jak již bylo naznačeno, objekt DataSet představuje místní kopii dat z databáze. Může obsahovat libovolné množství DataTable objektů (které reprezentují databázové tabulky), z nichž každý obsahuje objekty DataRows reprezentující řádky a DataColumn s reprezentující sloupce. Důležitá vlastnost Tables umožňuje přístup do kolekce objektů DataTable s názvem

DataTableCollection. Další kolekce v rámci DataSetu je DataRelationCollection, v níž jsou uchovávány relace mezi jednotlivými tabulkami. Například je možné vytvořit relaci mezi dvěma tabulkami s použitím závislosti cizího klíče pomocí typu DataRelation. Toho se docílí přes vlastnost DataSetu s názvem Relations. Poslední kolekcí v rámci DataSetu je Property Collection, přistupovaná přes vlastnost ExtendedProperties. V této kolekci mohou být uloženy jakékoli dodatečné informace ve formě párů název/hodnota. Tyto informace nemusí být ani nijak přímo spojeny s databází a mohou obsahovat např. jméno firmy, jejíž databáze je spravována, časové značky, zašifrované heslo pro přístup k DataSetu, apod. Některé další vlastnosti zahrnují název DataSetu, rozlišování malých/velkých písmen v tabulkách, vynucování závislostí, a jiné.

Klíčové metody:

- AcceptChanges() – provede všechny změny provedené v tomto DataSetu od doby jeho nahrání do paměti nebo od doby, kdy byla tato metoda volána posledně
- Clear() – Vymaže veškerý obsah tabulek v DataSetu
- Clone() – kopíruje infrastrukturu DataSetu, včetně tabulek a relací mezi nimi, ovšem nekopíruje data
- Copy() – kopíruje infrastrukturu i obsah DataSetu
- GetChanges() – vrací kopii DataSetu, obsahující všechny změny od doby nahrání nebo od doby, kdy byla volána metoda AcceptChanges(). Metoda může být přetížena, takže je možné získat pouze nové, pouze změněné nebo pouze smazané řádky
- HasChanges() – vrací hodnotu true, pokud došlo k nějakým změnám v řádcích
- Merge() – spojení dvou DataSetů
- RejectChanges() – zruší všechny změny od doby nahrání DataSetu nebo od doby, kdy byla volána metoda AcceptChanges().

4.3.2 DataTable objekt

Tyto objekty se dají nazvat reprezentací tabulek databáze. Jejich hlavní součástí jsou kolekce objektů Columns (obsahující objekty DataColumn , což jsou sloupce) a Rows (obsahující objekty DataRow, představující řádky). Přes metody těchto kolekcí dochází k přidávání, odebírání a manipulaci s řádky a sloupci tabulky. Přes metody kolekce Rows lze

navíc i přiřazovat hodnoty jednotlivých buňek tabulky. DataTable má mnoho identických vlastností s objektem DataSet, ale i pár specifických, mezi které patří ParentRelations a ChildRelations, které vrací kolekce vazeb typu rodič a potomek, Constraints pro výčet závislostí cizích klíčů tabulky nebo PrimaryKey, který vrací pole sloupců, sloužících jako primární klíč.

4.3.3 DataColumn objekt

Objekt DataColumn představuje jeden sloupec uvnitř tabulky a všeobecně řečeno, množina všech objektů tohoto typu svázaných s danou tabulkou je základem pro informace ve schématu tabulky. Pokud je třeba do tabulky přidat sloupec, vytvoří se instance tohoto objektu s vlastnostmi, které navolíme, jako název, datový typ, závislost na jiném sloupci, jen pro čtení, atd. Poté se přidá pomocí metody Add() kolekce Columns tabulky, do které chceme sloupec přidat.

4.3.4 DataRow objekt

Zatímco kolekce objektů DataColumn reprezentovala schéma tabulky, konkrétní hodnoty jednotlivých buňek, tedy data v tabulce jsou uložena jako pole hodnot v objektech DataRow. Počet prvků v poli je závislý na počtu sloupců v tabulce. Je třeba vědět, že přidávání nových řádků nelze udělat přes konstruktor, ale přes metodu NewRow() objektu DataTable, po jejímž zavolání se u daného řádku zadají hodnoty v jednotlivých sloupcích. Samotné přidání řádku do kolekce Rows se provede nakonec metodou Add(). Mezi členy (vlastnosti a metody) objektu DataRow patří členové na hlášení a opravu chyb, metody pro přijmutí či odmítnutí změn a zejména vlastnost ItemArray, což je pole představující jednotlivé hodnoty v řádku.

4.4 Entity Framework

Tento programový model se snaží zmenšit mezeru mezi databázovými a objektově orientovanými programovými konstrukcemi. Model jako takový nemá za cíl nahradit spojenou či odpojenou vrstvu, představuje spíše jen zjednodušující prvek při práci s nimi. EF umožňuje komunikovat s databází bez nutnosti napsat jakýkoliv SQL kód, což lze v případě aplikace LINQ na soubor silně typovaných objektů ADO.NET přímým generováním SQL příkazů díky LINQ to Entities. Rozložení entit (silně typovaných objektů) v EF se nazývá EDM (Entity Data Model) a vyjadřuje fyzické schéma rozložení databáze. Komunikaci EF s databází

zprostředkovávají datoví poskytovatelé, k nimž je však nutno dodat potřebné služby, spojující je s EF. Tyto služby jsou obsaženy v knihovně System.Data.Entities.

4.5 MySQL Connectory

Až doposud zde byly zmiňovány nezbytné součásti pro spojení s databázemi na straně platformy .NET, ovšem nástroje pro spojení se specifickým DBMS jsou dodávány jeho vývojáři, v tomto případě společností Oracle. Zmiňovaný nástroj, zvaný Connector, je důležitý zejména sadou jmenných prostorů (knihoven), které jsou registrovány pro dané vývojové prostředí a slouží k poskytnutí objektů a jejich metod pro spojení s MySQL databází. MySQL Connectory nabízí konektivitu pro aplikace a nástroje, které jsou kompatibilní se standarty ODBC a JDBC.

Obrázek 1 Seznam jednotlivých sad ovladačů Connector

Developed by MySQL
ADO.NET Driver for MySQL (Connector/NET)
ODBC Driver for MySQL (Connector/ODBC)
JDBC Driver for MySQL (Connector/J)
Python Driver for MySQL (Connector/Python)
C++ Driver for MySQL (Connector/C++)
C Driver for MySQL (Connector/C)
C API for MySQL (mysqlclient)

(Zdroj: 3)

5. Testovací databázová aplikace

Volba vývojového nástroje pro .NET platformu vychází z předchozích zkušeností s programem Visual Studio, kdy je k vývoji použita verze Visual Studio Professional 2012 se studentskou licencí. Použité knihovny obsahující datové poskytovatele pro MySQL jsou součástí Net/Connectoru v.6.9.6. Pro ověření možností spojení platformy .NET a databázového systému MySQL volíme aplikaci typu Windows Forms, zejména díky jednoduchému a intuitivnímu způsobu vývoje aplikace.

Databáze je vytvořena přes webhosting Web4u.cz, na kterém také běží MySQL Server. Pro správu databáze je použit nástroj MySQL Workbench ve verzi 6.1. Model databáze je vytvořen pomocí součásti Reverse engineer nástroje Workbench.

5.1 Struktura databáze

Model databáze, kterou jsme nazvali **warehouse_test**, je vytvořen podle fiktivní struktury fungování malé firmy, zabývající se kompletováním zásilek smíšeného zboží. Je založen zejména na relacích tabulek s InnoDB Storage Engine, kvůli převažujícímu množství tabulek, na které jsou často posílány DML příkazy jazyka SQL.

Jako první je tabulka Users, která jako jediná používá SE MyISAM, což je z důvodu častých select příkazů v případě logování uživatelů do databáze z více míst a také proto, že tato tabulka nepotřebuje být vázána cizími klíči. Jejím obsahem jsou uživatelské přístupové údaje a některé doplňující údaje o uživateli.

Další tabulkou jsou Employees, uchovávající údaje o zaměstnancích, jako jméno, pozice, datum nástupu do firmy. Tato tabulka má autoinkrementující (AI) číslo jako primární klíč, čímž je automaticky indexováno.

V tabulce Places jsou uchovávány informace o vnitřním firemním označení skladiště, obsahující specifické číslo a typ skladiště, doplněné o sekci, v níž jsou objednávky zpracovány. Opět je použito jako primární klíč AI číslo.

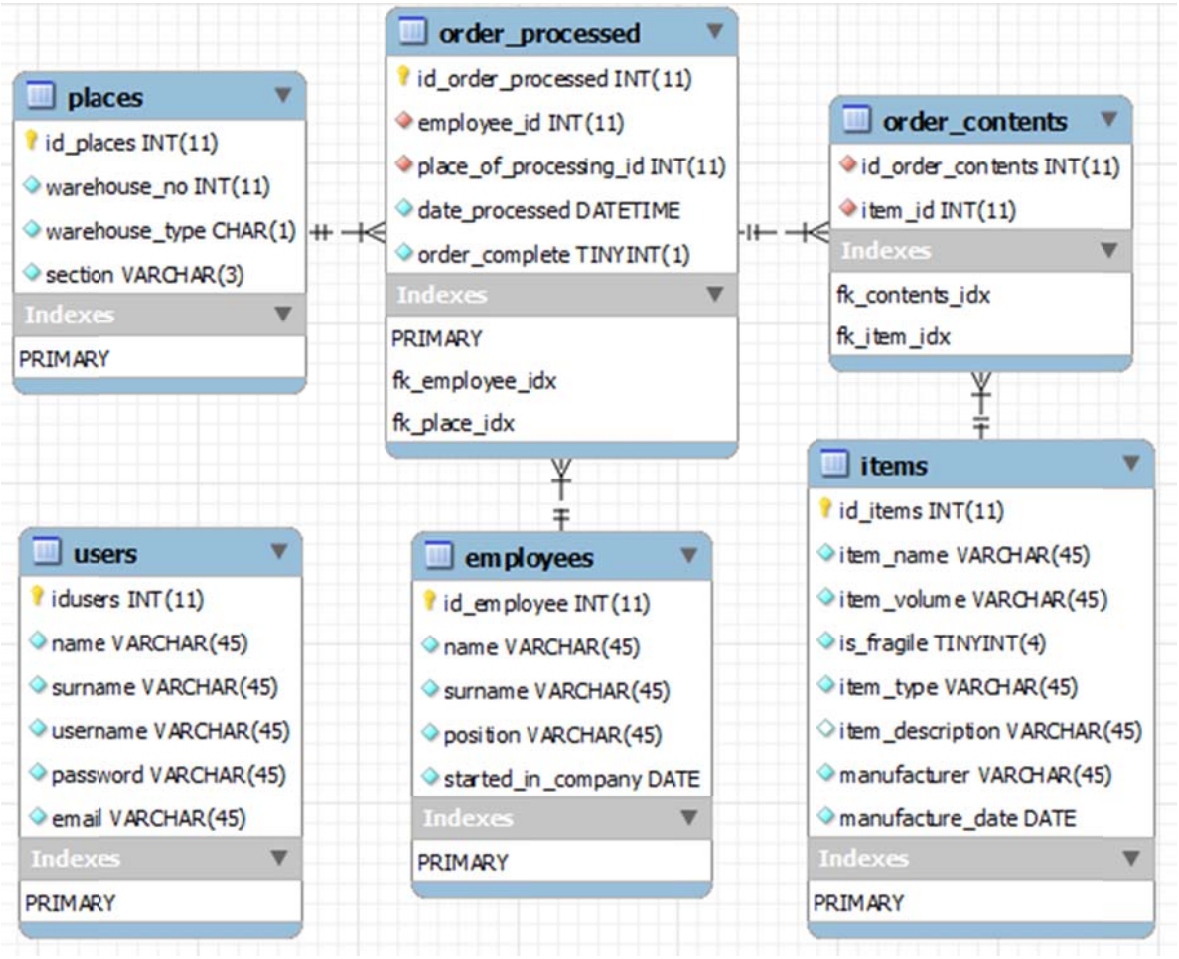
Pro skladování dostupného inventáře výrobků slouží tabulka Items, kde najdeme informace o názvu výrobku, jeho rozměru v m^3, typu výrobku, případně popisu, údaj o výrobci a datum výroby. Také je zde označení, zda-li je výrobek křehký.

Nejdůležitější tabulkou je Order_processed, která přes cizí klíče sdružuje tabulky Employees a Places a přidává datum zpracování objednávky.

Tabulka zajišťující obsah jednotlivých zásilek s názvem Order_contents slouží ke spojení tabulek Items a Order_processed. Díky ní je možná realizace vazby M:N pro jednotlivé zásilky, tedy různá množství výrobků v zásilkách.

Tabulky InnoDB v této databázi představují normalizovaný model, spoléhající zejména na závislosti cizích klíčů a na rychlé čtení pomocí spojování tabulek.

Obrázek 2 Grafický model testovací databáze

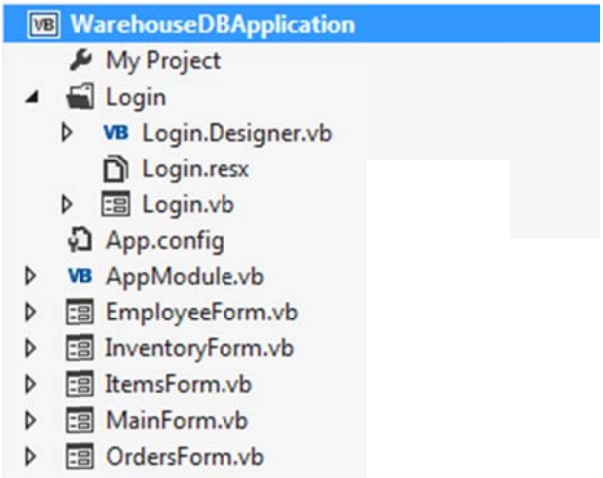


(Zdroj: Vlastní úprava)

5.2 Struktura aplikace

Program komunikující s databází, pojmenovaný **WarehouseDBApplication**, je uzpůsoben zejména k zadávání dat do jednotlivých tabulek. Sestává se z několika formulářů, sloužících jako grafické uživatelské rohraní. Data do databáze jsou zadávána buď přes textová pole, výběrem položky ze seznamu položek (Comboboxu) nebo označením řádků v datové mřížce (DataGridView). Všechny datové vstupy jsou parametrizovány.

Obrázek 3 Seznam tříd v testovací aplikaci



(Zdroj: Vlastní úprava)

5.2.1 Implementace datového poskytovatele

Komunikace s databází v kódu je řešena pomocí blokového příkazu Using, který umožňuje efektivně pracovat s pamětí. Deklarace objektů je vyřešena klíčovým slovem New následovaným typem objektu. Tím se vytvoří nová instance objektu, kterou je možné uvnitř bloku dále používat. Po skončení užívání objektů uvnitř bloku koncovým příkazem End Using jsou objekty z paměti odstraněny.

Po vytvoření objektů přiřazujeme objektům potřebné vlastnosti, definované v jejich konstruktoru. U objektu Connection je parametr v jednoduchých závorkách, u Command objektu používáme klíčové slovo With a názvy vlastností s tečkovou notací.

Poté, co jsou nadefinovány vlastnosti, používáme blokový příkaz Try k zavolání metod pro komunikaci s databází, kdy otevřeme připojení a zavoláme příslušné metody. V rámci Try používáme klíčové slovo Catch, které slouží k ošetření vyjímek, zpravidla zobrazením chybové hlášky. Před ukončovacím slovem End Try ještě uvolníme objekt Connection pro odstranění v rámci bloku Finally.

5.2.2 AppModule

Tento modul slouží k deklaraci globálních proměnných objektů datového poskytovatele MySQL.Data, využívaných ke spojení s databází.

5.2.3 LoginForm

Formulář sloužící ke vstupu do databázové aplikace. Jeho funkce spočívá v získání tabulky Users, porovnání se vstupními řetězci reprezentujícími jméno a heslo uživatele s řetězci v tabulce Users a v případě shody otevření MainFormu.

Obrázek 4 Kód pro získání dat z tabulky Users ve formuláři LoginForm

```
Using MysqlConn As New MySqlConnection(connString)
    Using MyCommand As New MySqlCommand()
        Using MyAdapter As New MySqlDataAdapter
            With MyCommand
                .Connection = MysqlConn
                .CommandText = "select * from users where username=@admin" & _
                               "and password=@pass"
                .Parameters.AddWithValue("@admin", txtUser.Text)
                .Parameters.AddWithValue("@pass", txtPass.Text)
                .CommandType = CommandType.Text
                MyAdapter.SelectCommand = MyCommand
            End With
            Try

                MyAdapter.Fill(dtAdmin)

            Catch ex As MySqlException
                MessageBox.Show(ex.Message)
            Finally
                MysqlConn.Dispose()
            End Try
        End Using
    End Using
End Using
```

(Zdroj: Vlastní úprava)

5.2.4 MainForm

Je to rozcestník mezi ostatními formuláři, zobrazující jednotlivé formuláře při kliknutí na příslušná tlačítka. Na tomto formuláři lze taktéž ukončit aplikaci.

5.2.5 EmployeeForm

Zde je řešena správa zaměstnanců přes objekty RadioButton, kdy je možné vybírat mezi přidáváním do databáze a vyhledáváním v databázi pomocí klikání na příslušně označené políčko. V případě zadávání je odemčeno tlačítko pro uložení do databáze a po zadání hodnot se provede příkaz pro vložení do databáze s parametry, které mají hodnoty vyplněných polí. Vzhledem k nastavení sloupců tabulky Employees je nutné pro úspěšné uložení vyplnit všechna pole.

Obrázek 5 Část kódu pro vložení do tabulky Employees z formuláře EmployeeForm

```
With MyCommand
    .Connection = MysqlConn
    .CommandText = "insert into employees " & _
        "(name,surname,position,started_in_company) " & _
        "values (@nm,@snm,@pos,@sic)"
    .CommandType = CommandType.Text
    .Parameters.AddWithValue("@nm", empNameBox.Text)
    .Parameters.AddWithValue("@snm", empSurnameBox.Text)
    .Parameters.AddWithValue("@pos", empPosBox.Text)
    .Parameters.AddWithValue("@sic", _
                             String.Format _
                             ("{0:yyyy-MM-dd}", _
                              targetDate))
End With
Try
    MysqlConn.Open()

    MyCommand.ExecuteNonQuery()

    MysqlConn.Close()
```

(Zdroj: Vlastní úprava)

5.2.6 ItemsForm

V tomto formuláři je možné zadávat do databáze výrobky a jejich specifické popisující vlastnosti. Zadávání zde funguje obdobně jako v případě EmployeeForm, tedy přes zadávání textových hodnot nebo výběr datumu přes objekt DateTimePicker, případně výběr dvouhodnotové vlastnosti křehkost skrz objekty RadioButton. Změna zde je v zobrazení hodnot v databázi, která je zprostředkována přes zavolání formuláře InventoryForm.

5.2.7 InventoryForm

Jedinou funkcí tohoto formuláře je zobrazení obsahu tabulky Items pomocí vytvoření kopie v DataSetu a datového svázání této kopie s objektem DataGridView. V objektu DataGridView lze následně vybírat řádky a pracovat s nimi, čehož je využito při vybírání výrobků do zásilky. Kromě této klíčové vlastnosti, lze také např. uzpůsobit zobrazení řádků seřazení podle hodnot v jednotlivých sloupcích. Pod objektem DataGridView na formuláři je tlačítko, které má různé funkce na základě toho, odkud byla zavolána metoda pro zobrazení InventoryFormu. Pokud došlo k jejímu zavolání z formuláře ItemsForm, dojde po kliknutí na tlačítko k návratu na tento formulář. V případě volání z formuláře OrdersForm dojde k odblokování tohoto formuláře a k přidání řádků vybraných v InventoryForm do jeho DataGridView s výrobky pro přidání do zásilky.

5.2.8 OrdersForm

Klíčový formulář pro zadávání zpracovaných zásilek je řešený zcela bez textových vstupů ze strany uživatele. Ten zadává hodnoty z čtecích Comboboxů obsahujících data načtená z databázových tabulek. Pro výběr datumu opět slouží objekt DateTimePicker. Určení kompletnosti zásilky je řešeno přes dvojici objektů RadioButton. Seznam výrobků, které mají být přidány do tabulky Order_contents pro spojení se zadávanou zásilkou je spravován přes objekt DataGridView. Klepnutím pravým tlačítkem myši se otevře kontextové menu, kde je možné vybírat jednotlivé úkony s výrobky. V případě označených výrobků je možné odebrat vybrané výrobky klepnutím na volbu Odebrat Výrobek. Po kliknutí na volbu Přidat Výrobek se otevře formulář InventoryForm, v němž je možné vybrat položky a potvrdit přidání tlačítkem.

Obrázek 6 Část kódu pro příkazy v transakci

```
Try

    MyCommand.ExecuteNonQuery()

    MyCommand.CommandText = "select max(id_order_processed) from order_processed"

    x = MyCommand.ExecuteScalar()

    MyTransaction.Commit()

    MysqlConn.Close()
Catch ex As MySqlException

    If MyTransaction IsNot Nothing Then
        MyTransaction.Rollback()
    End If

    MessageBox.Show(ex.Message)
Finally
    MysqlConn.Dispose()
    MyTransaction.Dispose()
End Try
```

(Zdroj: Vlastní úprava)

Po stisknutí tlačítka pro uložení na formuláři proběhne transakce obsahující dva příkazy, kdy první vloží hodnoty charakterizující tuto zpracovanou objednávku do tabulky Order_processed a druhý získá právě vygenerované autoinkrementující číslo, představující primární klíč. Tato hodnota je pak předána jako parametr dalšímu příkazu pro vložení, tentokrát do tabulky Order_contents. Řetězec pro vložení do Order_contents má dynamický charakter, závislý na počtu řádků v DataGridView.

6. Závěr

V této práci jsme byli obeznámeni jak se specifickými vlastnostmi platformy .NET a databázového systému MySQL, tak i se způsoby, jakými mohou komunikovat. V teoretické části jsme se setkali se strukturou platformy .NET, jejím historickým vývojem, jejími součástmi a vývojovými prostředími. V části o MySQL byla představena unikátní architektura zásuvných storage engines, konkrétní typy storage engines a jejich specifikace, datový slovník a programy nezbytné pro běh a správu serveru. Další část obsahovala informace o datových poskytovatelích pro platformu .NET, obsahujících objekty nezbytné ke komunikaci s databázovými systémy, dále pak vrstvy, skrz které lze s databázemi komunikovat, a nakonec konkrétní programové rozhraní Connector, obsahující datového poskytovatele pro spojení s MySQL. V praktické části jsme popsali testovací databázi a s ní spojenou aplikaci. V databázi jsme použili dva různé storage engine pro databázový model fungování fiktivní firmy. Na tomto modelu jsme ukázali způsob provázání tabulek se storage engine podporujícím cizí klíče. V aplikaci jsme pak ukázali kromě práce s objekty spravujícími data, jak lze používat objekty datového poskytovatele pro komunikaci s databází, jak nastavit vlastnosti objektů a jak volat metody, které vykonají dané dotazy. Také jsme si popsali, jak poslat více dotazů v rámci jedné transakce.

Seznam Zdrojů

[1] BrightHub Science & Technology Articles. *http://www.brighthub.com/.* [Online] [Citace: 01. 02 2015.] http://www.brighthub.com/internet/web-development/articles/110654.aspx.

[2] DigitalOcean Community. *https://www.digitalocean.com/community/.* [Online] [Citace: 01. 02 2015.] https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems.

[3] *MySQL Documentation: MySQL Reference Manuals.* [Online] [Citace: 01. 02 2015.] http://dev.mysql.com/doc/.

[4] **Ryan Stephens, Ron Plew, Arie D. Jones.** *Naučte se SQL za 28 dní.* 2010. ISBN 978-80-251-2700-1.

[5] **Troelsen, Andrew.** *Pro C# and the .NET 4.5 Framework.* 2012. ISBN 978-1-4302-4233-8.

[6] Úvod do .NET Frameworku. *http://www.dotnetportal.cz.* [Online] [Citace: 01. 02 2015.] http://www.dotnetportal.cz/clanek/125/Uvod-do-NET-Frameworku.

Seznam Obrázků

Seznam Tabulek