UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Aplikace pro práci s daty o výrobě automobilového závodu

Ladislav Češpiva

Bakalářská práce
2013

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

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Ladislav Češpiva**
Osobní číslo:	**I09007**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Komunikační a mikroprocesorová technika**
Název tématu:	**Aplikace pro práci s daty o výrobě automobilového závodu**
Zadávající katedra:	**Katedra elektrotechniky**

Zásady pro vypracování:

Úkolem této bakalářské práce je tvorba aplikace a databáze pro ŠKODA AUTO a.s.. Cílem této aplikace, napsané v jazyce C, je ukládat data o výrobě a tvorba hlášení o výrobě. Data se budou ukládat na vzdálený server do databáze MS SQL. Teoretická část práce bude obsahovat popis a rozbor problematiky, objektový návrh struktury programu. V praktické části bude vytvořena zadaná aplikace, součásti práce bude kompletní dokumentace k aplikaci.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

STEPHENS, Ryan K, Ronald R PLEW a Arie JONES. Naučte se SQL za 28 dní. Vyd. 1. Překlad Lukáš Krejčí. Brno: Computer Press, 2010, 728 s. ISBN 978-80-251-2700-1.
HANÁK, Ján. C 3.0: Programování na platformě .NET 3.5. Vyd. 1. Brno: Zoner Press, 2009, 282 s. ISBN 978-80-7413-046-5.
BISHOPOVÁ, Judith. C: návrhové vzory. Vyd. 1. Brno: Zoner Press, 2010, 323 s. ISBN 978-80-7413-076-2.

Vedoucí bakalářské práce: **Ing. Pavel Rozsíval**
Katedra elektrotechniky

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

prof. Ing. Simeon Karamazov, Dr.
děkan

L.S.

Ing. Zdeněk Němec, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

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 16.8.2013 Ladislav Češpiva

Poděkování

Tímto bych rád poděkoval svému vedoucímu bakalářské práce panu Ing. Pavlu Rozsívalovi za pomoc, ochotu a dobré rady při zpracování této práce. A také své přítelkyni, která mi pomáhala formulovat myšlenky.

Anotace

Úkolem této bakalářské práce je tvorba aplikace a databáze pro ŠKODA AUTO a.s.. Cílem této aplikace, napsané v jazyce C#, je ukládat data o výrobě a tvorba hlášení o výrobě. Data se budou ukládat na vzdálený server do databáze MS SQL.

Klíčová slova

aplikace, databáze, server, C#

Title

Application for working with automobile factory’s data of production

Annotation

The task of this bachelor thesis is to create an application and database for ŠKODA AUTO a.s.. The objective of this application, written in C# language, is to store data of production and to create reports of production. The data will be stored in the remote server in MS SQL database.

Keywords

application, database, server, C#

Obsah

Seznam zkratek

ANSI	American National Standards Institute
CRM	Customer Relationship management
DBMS	Database managment systém
ECMA	European Computer Manufacturers Association
ER	Entity- relationship
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
IT	Informační technologie
MIT	Massachusetts Institute of Technology
POOP	Paralelní objektově orientované programování
SQL	Structured Query Language
VB.NET	Visual Basic.NET

Seznam obrázků

Seznam tabulek

Úvod

Pro společnost jako je ŠKODA AUTO a.s. je důležité vědět, kolik kusů bylo vyrobeno, kolik zaměstnanců bylo v provozu, jaké byly závady na strojích, jaké prostoje strojů vznikly a tyto údaje mít rozděleny podle data, směny, výrobní oblasti atd. Z těchto dat se dá jednoduše sledovat výroba, vytvářet statistiky a vypočítat produktivita. Na základě těchto informací podnik upravuje výrobní plán. Pro tyto účely je nejlepší použít pro uložení dat databázi, a na práci s touto databází, pak program s uživatelsky snadným a příjemným rozhraním. Tato aplikace je vytvořena pro pobočku ŠKODA AUTO a.s. ve Vrchlabí, kde se vyrábí převodovky do automobilů.

V první kapitole jsou základní informace o databázích, databázových systémech a pravidlech pro návrh a tvorbu databáze.

Druhá kapitola se zabývá programovací jazykem C#. Je zde popsán jeho vývoj spolu s jeho vlastnostmi a možnostmi použití.

Ve třetí kapitole je popsána databáze Vyroba, což je databázová část aplikace. Jsou zde uvedeny všechny tabulky, vztahy mezi tabulkami a funkce, kterou v aplikaci zastávají.

Ve čtvrté kapitole je popsána programová část aplikace. Je zde popsána struktura programu a detailněji popsány třídy, důležité části kódu a propojení se vzdálenou databází Vyroba.

1 Databáze

Databáze představuje nástroj pro shromáždění a uspořádání informací. Do databází lze ukládat informace o osobách, produktech, objednávkách nebo čemkoli jiném. Mnoho databází vzniká jako seznam v textovém editoru nebo tabulkovém procesoru. Jak seznam narůstá, začínají se v datech objevovat redundance a nekonzistence. Data ve formě seznamu se stávají obtížně srozumitelnými a je jen málo způsobů, jak vyhledávat nebo načítat jejich podmnožiny pro prohlížení. Jakmile se začnou objevovat problémy, je vhodné data přenést do databáze vytvořené v systému pro správu databází DBMS. Aby mohl být nějaký programový systém označený za DBMS, musí být jednak schopen efektivně pracovat s velkým množstvím dat, ale také musí být schopný řídit (vkládat, modifikovat, mazat) a definovat strukturu těchto perzistentních dat.

1.1 Relační databáze

Nejpopulárnějším modelem datového úložiště je relační databáze, která se zrodila z klíčové studie „A Relational Model of Data for Large Shared Data Banks“ (relační model dat pro rozsáhlé sdílené banky dat), kterou napsal Dr. E. F. Codd v roce 1970. Jazyk SQL se vyvinul tak, aby sloužil principům relačního modelu databáze. Dr. Codd definoval pro relační model 13 pravidel, kterým se kupodivu říká 12 pravidel Dr. Codda:

0. Relační databázový systém musí být schopen spravovat databáze jen s využitím svých relačních schopností.
1. **Informace:** Všechny informace v relační databázi (včetně názvů tabulek a sloupců) jsou reprezentovány explicitně jako hodnoty v tabulkovém formátu.
2. **Zaručený přístup:** U každé hodnoty v relační databázi je zaručeno, že bude přístupná pomocí kombinace názvu tabulky, primárního klíče a názvu sloupce.
3. **Systematická podpora nulitní hodnoty:** Databázový systém poskytuje systematickou podporu s nulitními hodnotami (neznámá či nepoužitelná data), které jsou odlišné od výchozích hodnot a nezávislé na jakékoliv doméně.
4. **Aktivní relační katalog dostupný online:** Popis databáze a jejího obsahu je reprezentován na logické úrovni v tabulkové formě, a proto lze nad ním spouštět dotazy pomocí databázového jazyka.
5. **Ucelený datový podjazyk:** Alespoň jeden podporovaný jazyk musí mít dobře definovanou syntaxi a musí být ucelený. Musí podporovat definici dat, manipulaci s daty, integritní pravidla, autorizaci a transakce.
6. **Aktualizace pohledů:** Veškeré pohledy, které lze teoreticky aktualizovat, mohou být aktualizovány pomocí systému.
7. **Vkládání, aktualizace a mazání na úrovni množin:** Databázový systém podporuje nejen získávání dat na úrovni množin, ale také vkládání, aktualizace a mazání na úrovni množin.
8. **Fyzická datová nezávislost:** Změna fyzických přístupových metod nebo úložných struktur nemá z logického hlediska na aplikační a ad hoc programy žádný vliv.
9. **Logická datová nezávislost:** Změna tabulkových struktur nemá z logického hlediska na aplikační a ad hoc programy v maximální možné míře vliv.

10. **Integritní nezávislost:** Databázový jazyk musí být schopen definovat integritní pravidla. Tato pravidla musejí být uložena v katalogu dostupném online a nesmí existovat možnost pro jejich obejití.
11. **Nezávislost distribuce:** První distribuce ne redistribuce dat nemá na aplikační programy a ad hoc požadavky z logického hlediska žádný vliv.
12. **Nenarušitelnost:** Nesmí existovat možnost umožňující obejití integritních pravidel definovaných v databázovém jazyku pomocí jazyků nižší úrovně.[1]

1.2 Databázové systémy

Pro tvorbu databázové části aplikace byla možnost velkého výběru databázových systémů. Ty nejpoužívanější databázové systémy si představíme v této podkapitole. Tyto systémy jsou si velice podobné, protože všechny využívají jazyk SQL, ale existují zde malé odlišnosti.

1.2.1 Microsoft SQL Server

Společnost Sybase má na svědomí původní implementaci databáze SQL Server, která byla původně navržena pro operační systém OS/2. Později uzavřela dohodu o vývoji kódu se společností Microsoft, která přenesla aplikaci pro OS/2 na svou platformu Windows. V roce 1993 se tyto dvě společnosti rozhodly jít každá vlastní cestou.

1.2.2 Oracle Database

Oracle je systém řízení báze dat, moderní multiplatformní databázový systém s velice pokročilými možnostmi zpracování dat, vysokým výkonem a snadnou škálovatelností. Přesněji Oracle Corporation je název firmy, oficiální název databázové platformy je Oracle Database. Oracle Corporation je jedna z hlavních společností vyvíjejících relační databáze, nástroje pro vývoj a správu databází či CRM systémů. Byla založena v roce1977. Aktuální verzí je Oracle Database 11g. Tento systém podporuje nejen standardní relační dotazovací jazyk SQL podle normy SQL92, ale také proprietární firemní rozšíření, imperativní programovací jazyk PL/SQL rozšiřující možnosti vlastního, dále podporuje objektové databáze a databáze uložené v hierarchickém modelu dat

1.2.3 MySQL

MySQL je databázový systém, vytvořený švédskou firmou MySQL AB, nyní vlastněný společností Sun Microsystems, dceřinou společností Oracle Corporation. Je považován za úspěšného průkopníka dvojího licencování – je k dispozici jak pod bezplatnou licencí GPL, tak pod komerční placenou licencí. MySQL je multiplatformní databáze. Podobně jako u ostatních SQL databází se jedná o dialekt tohoto jazyka s některými rozšířeními. Pro svou snadnou implementovatelnost, výkon a především díky tomu, že se jedná o volně šiřitelný software, má vysoký podíl na v současné době používaných databázích.

1.2.4 PostgresSQL

PostgreSQL (Postgres), je objektově-relační databázový systém (ORDBMS). Vydáván je pod licencí typu MIT a tudíž se jedná o bezplatný software. Stejně jako v případě mnoha dalších bezplatných programů, PostgreSQL není vlastněn jedinou firmou, ale na jeho vývoji se podílí globální komunita vývojářů a firem. PostgreSQL je primárně vyvíjen

pro Linux resp. pro unixové systémy obecně, nicméně existují i balíčky pro platformu win32 a win64.

1.3 Normalizace databáze

Normalizace je proces redukce opakování a redundancí dat v databázi. Pro normalizaci databáze existuje šest normálových forem, ale v praxi se využívají první čtyři. Normalizace databáze poskytuje databázi četné výhody. Mezi ty nejdůležitější patří:

- celkově lepší uspořádání databáze,
- nižší redundance dat,
- konzistence dat v databázi,
- flexibilnější návrh databáze,
- lepší zacházení a zabezpečení databáze.

1.3.1 První normálová forma

Cílem první normální formy je rozdělit data do logických jednotek, tabulek. Po navržení tabulek je většině z nich přiřazen primární klíč. Primární klíč v tabulce reprezentuje jeden nebo více sloupců, kterým je každý řádek v tabulce jedinečný.

1.3.2 Druhá normálová forma

Cílem druhé normální formy je vzít data, která jsou jen částečně závislá na primárním klíči, a uložit je do jiné tabulky.

1.3.3 Třetí normální forma

Cílem třetí normální formy je odstranit z tabulky data, která nejsou závislá na primárním klíči.

1.3.4 Boyce-Coddova normální forma

Pro Boyce-Coddovu normální formu existuje definice: „Relace R je v Boyce-Coddově normální formě, pokud pro každou funkční závislost X→Y, kde X a Y jsou množiny atributů a zároveň, kde Y není podmnožinou X, platí, že X je nadmnožina nějakého klíče, nebo X je klíčem relace R.“

1.3.5 Čtvrtá normálová forma

Tabulka je ve čtvrté normální formě, je-li ve třetí a popisuje pouze příčinnou souvislost (jeden fakt).

1.3.6 Pátá normálová forma

Tabulka je v páté normální formě, pokud je ve čtvrté a není možné do ní přidat nový sloupec nebo skupinu sloupců tak, aby se vlivem skrytých závislostí rozpadla na několik dílčích tabulek.

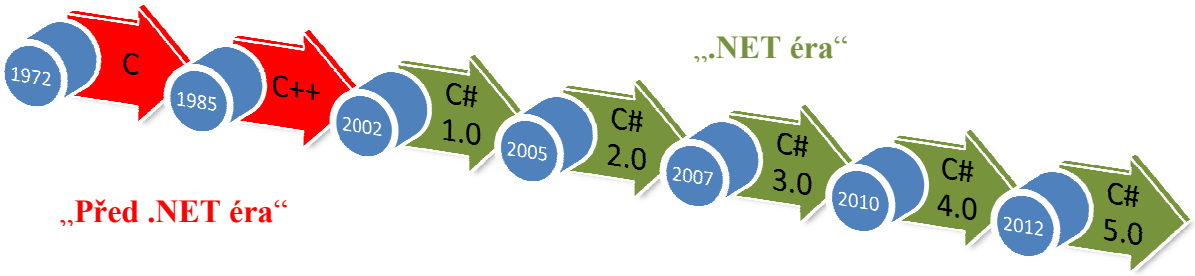
1.4 Jazyk SQL

Historie jazyka SQL začala v laboratoři společnosti IBM v San Jose v Kalifornii. Zde byl na konci sedmdesátých let dvacátého století jazyk SQL vyvinut. Zkratka SQL znamená Structured Query Language (strukturovaný dotazovací jazyk). SQL je jazyk neprocedurální

a množinově orientovaný. Slovo neprocedurální znamená, že nepopisuje, jak se má něco provést, ale spíše to, co se má provést. Množinová orientace jazyka SQL znamená, že zpracovává data jako množiny nebo skupiny. Díky tomu se jazyk SQL využívá v relačních databázových systémech. Dvě standardizační organizace ANSI a ISO propagují jazyk SQL jako průmyslový standard.[1]

2 Jazyk C#

C# je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft zároveň s platformou .NET Framework, později schválený standardizačními komisemi ECMA (ECMA-334) a ISO (ISO/IEC 23270). Implementace .NET byla natolik hluboká, že se začalo dění v IT průmyslu dělit na „před .NET éru“ a „.NET éru“. Microsoft založil C# na jazycích C++ a Java (je nepřímím potomkem jazyka C, ze kterého čerpá syntaxi). C# lze využít k tvorbě databázových programů, webových aplikací a stránek, webových služeb, formulářových aplikací ve Windows, softwaru pro mobilní zařízení atd. [3]



Obrázek 1: Od jazyka C až k jazyku C# 5.0

2.2 Aspekty jazyka C#

Hlavní aspekty jazyka C# můžeme charakterizovat následujícím způsobem:

- **C# je hybridní objektově orientovaný jazyk.** Ačkoliv pořád hybridní, jazyk C# zavádí kompletní podporu pro všechny stěžejní principy všeobecné teorie objektově orientovaného programování, k nimž patří abstrakce, zapouzdření, ukrývání dat, ukrývání implementace, dědičnost a polymorfismus.
- **C# je standardizovaný organizacemi ECMA International a ISO.** Jedná se o tyto dokumenty: ECMA:334 C# - Language Specification a ISO/IEC 23270 Information Technology – Programming Languages – C#.
- **C# používá pro návrh fragmentů zdrojového kódu velice podobná syntaktická pravidla, jež uplatňují jazyky C a C++.** To je výhodné zejména tehdy, když jste v minulosti programovali v C/C++ a nyní byste rádi přešli směrem k C#. Programátoři jsou vždycky nadšení, když mohou v novém programovacím jazyce okamžitě využít něco, co již dávno dobře znají. Filosofie psaní zdrojového kódu jazyka C# vychází z jazyků C a C++, což napomáhá rychlé asimilaci v novém prostředí.
- **C# se vyznačuje vysokou pracovní produktivitou.** Nárůst pracovní produktivity lze pozorovat ve dvou rovinách. Za prvé, díky optimálně neprojektované jazykové specifikaci mohou vývojáři psát algoritmy v jazyce C# rychleji a s menší námahou než v jiných programovacích jazycích. Za druhé, proces vytváření aplikací je v jazyce C# velice rychlý a komfortní.
- **Jazyková specifikace C# je ve srovnání s C++ méně košatá.** Tuto skutečnost nesmíme považovat za stinnou stránku jazyka C#. Právě naopak, tím, že se C#

odpoutává od mnohdy zbytečně náročných a spletitých programových konstrukcí jazyka C++, eliminuje možnosti výskytu početné množiny chyb, k nimž v minulosti s velkou frekvencí docházelo. Zjednodušeně se dá říct, že specifikace jazyka C# je oproti C++ kompaktnější a především bezpečnější.

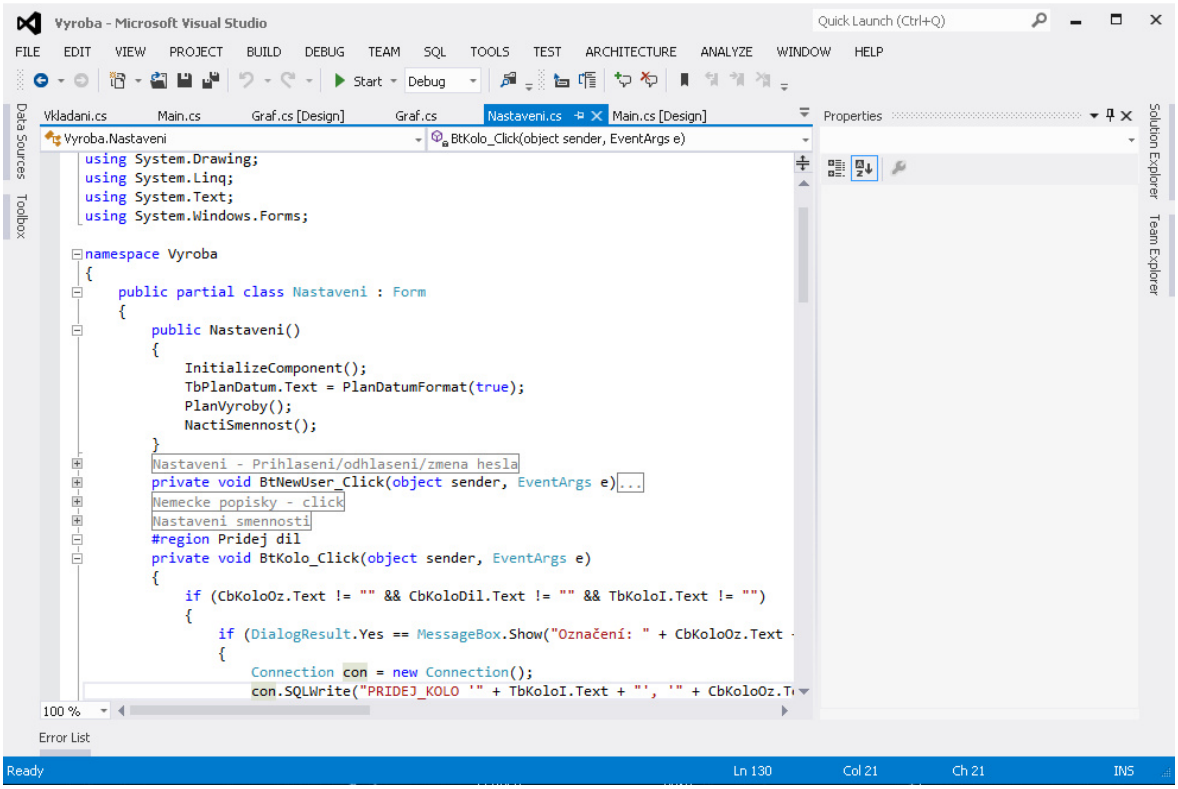
- **C# podporuje generické programování.** Generické programování neboli též parametrizované programování bylo po dlouhá léta doménou jazyka C++ a jeho šablon. V C++ můžeme pomocí šablon navrhovat generické funkce nebo rovnou generické datové typy, což jsou, zjednodušeně řečeno, všeobecné vzory, sloužící ke konstrukci konkrétních funkcí nebo typů, jež byly podle těchto vzorů automaticky vytvořeny. Přestože C# nepodporuje práci se šablonami, umí spolupracovat s generiky. Generiky mohou mít variabilní povahu a syntaktickou interpretaci. Vývojáři v jazyce C# mohou pracovat s generickými metodami, generickými třídami, generickými delegáty a generickými rozhraními. Pro úplnost si dovoluji poznamenat, že podpora generik byla do jazyka C# zapracována ve verzi 2.0.
- **C# implementuje vybrané prvky funkcionálního programování.** Funkcionální programování nabízí jiné paradigma než imperativní programování, s nímž se programátoři nejčastěji seznamují. Styl vývoje funkcionálního programování má spíše deklarativní povahu a jeho filosofie se více přibližuje k matematice. Pokud bychom chtěli rychle diferencovat mezi funkcionálním a imperativním paradigmatem programování, pak bychom konstatovali následující. Funkcionální programování se věnuje spíše popisu procesu, jenže má uskutečnit, ovšem oprošťuje se od nutnosti exaktní a mnohdy i velice detailní analýzy toho, jak technicky daný proces provést. Velice zjednodušeně řečeno, funkcionální paradigma se zaměřuje na to, co se má udělat, nicméně neříká, jak přesně se to má udělat. Styl zápisu funkcionálních programů je velice symbolicky a deterministický. Vzhledem k tomu, že je C# původně imperativní jazyk, počínaje verzí 2.0 byly do něj pomalu zanášeny prvky, které jsou známy z funkcionálního programování. Například anonymní metody, čili metody, které nemají programátorem přisouzený identifikátor, jsou hezkou ukázkou toho, jak si C# vypůjčil jeden aspekt funkcionálního programování.
- **C# podporuje paralelní objektově orientované programování.** S příchodem procesorů s více výpočetními jádry se programátoři snaží využít veškerý disponibilní výkon těchto hardwarových platforem. Je zřejmé, že cílem vývojářů je psát optimalizovaný a především škálovatelný software, jenž má tendenci prokazovat přinejmenším sublineární nárůst výkonnosti při běhu na vícejádrových procesorech a víceprocesorových strojích. Vytvářením takovýchto programů se zabývá odvětví moderní informatiky, které nazýváme paralelní objektově orientované programování. POOP nabízí metodiku, jak naprogramovat aplikaci tak, aby uměla vykonávat více úloh paralelně s tím, že zpracování každé úlohy je delegováno na samostatné výpočetní jádro vícejádrového procesoru. Tímto se promění sekvenční aplikace na aplikaci paralelní. Paralelizace má různé praktické implikace, k nimž patří: snížení vykonávacího času na řešení jisté problémové úlohy, schopnost vyřešit za stejný čas složitější úlohy, možnost progresivně využít veškerou hardwarovou sílu počítače a mnohé další.[2]

2.3 Vývojové prostředí

Pro C# existuje několik vývojových prostředí. Některá jsou dostupná zdarma, ale ty jsou spíše pro studentské účely. Jmenujme třeba tyto PSPad, SharpDevelop, Visual C#. Pro profesionální účely doporučuji Visual Studio.

2.3.1 Visual Studio 2012

Vývojové prostředí Visual Studia od Misrosoftu je velmi oblíbené. Visual Studio se prodává v několika edicích, ale jen edice Express je bezplatná. Visual Studio může být použito pro vývoj konzolových aplikací a aplikací s grafickým rozhraním spolu s aplikacemi Windows Forms, webovými stránkami, webovými aplikacemi a webovými službami, jak ve strojovém kódu, tak v řízeném kódu. Podporuje psaní programu v různých jazycích. Mezi vestavěné jazyky patří C/C++, VB.NET, C#. Ostatní programovací jazyky je možno přidat pomocí jazykových služeb. Dobrý pomocník v programu je editor kódu, který zvýrazňuje syntaxi a chybně napsaný kód.[4]



Obrázek 2: Vývojové prostředí Visual Studio 2012

3 Návrh aplikace

Pro návrh aplikace je důležité provést analýzu problematiky, kvůli které je tvořena. Tato aplikace byla tvořena pro záznam dat z výroby převodovek, proto je nutné mít základní znalosti o provozu výroby. Dále bylo nutné přesně specifikovat, s jakými informacemi má aplikace pracovat, funkce, které má aplikace dělat, vzhled aplikace, programovací jazyk a databázový systém, s kterým se pracuje. Funkce a informace se odvíjejí od výroby převodovek. Vzhled, programovací jazyk a databázový systém od požadavků IT oddělení ŠKODA AUTO a.s..

3.1 Základní informace o výrobě převodovek

Výroba převodovek ve ŠKODA AUTO a.s. ve Vrchlabí je rozdělena do výrobních oblastí, to jsou:

- kalírna,
- výroba ozubených kol,
- výroba hřídelí,
- montáž mechatronik pro sériovou výrobu,
- montáž mechatronik mimo sériovou výrobu,
- montáž převodovek.

Každá výrobní oblast má své mistry a směnnost. Směnnost je dána druhem směn, při kterých výrobní oblast vyrábí. Existují zde tyto typy směn:

- Ranní
- Odpolední
- Noční
- BU1
- BU2
- BU3
- BU4

Každá výrobní oblast zaznamenává data z výroby. Některá zaznamenaná data jsou pro všechny výrobní oblasti ve stejném formátu. Jsou to prostoje strojů, závady ve výrobě, přítomnost a produktivita. Záznamy o počtu kusů se v různých oblastech výroby samozřejmě liší.

Každá výrobní oblast má určený výrobní plán, kterým se řídí a pokud ho nesplní, tak se dohledává, čím to bylo způsobeno. Výrobní plán vytváří vedoucí výrobních oblastí podle požadavků ŠKODA AUTO a.s. v Mladé Boleslavi na vyrobené kusy.

3.2 Funkce aplikace

Aplikace pracuje s databázovým systémem Microsoft SQL Server, protože ho ŠKODA AUTO a.s. již používá. Jelikož ŠKODA AUTO a.s. patří společnosti Volkswagen International Finance N.V., tak jedním z požadavků na aplikaci byla dvojjazyčnost. Aplikace je tedy s českými i německými popisky. Pomocí této aplikace se ukládají záznamy o výrobě do databáze. Dále se z těchto záznamů vytváří reporty. V aplikaci je možné upravovat parametry výroby jednotlivých výrobních oblastí, jako je změna směnnosti a mistrů, plánování výroby, přidání vyráběných dílů. Aplikace se dělí na čtyři základní sekce. Těmito sekcemi jsou:

- zadávání výroby,
- tabulkové reporty,
- grafické reporty,
- nastavení.

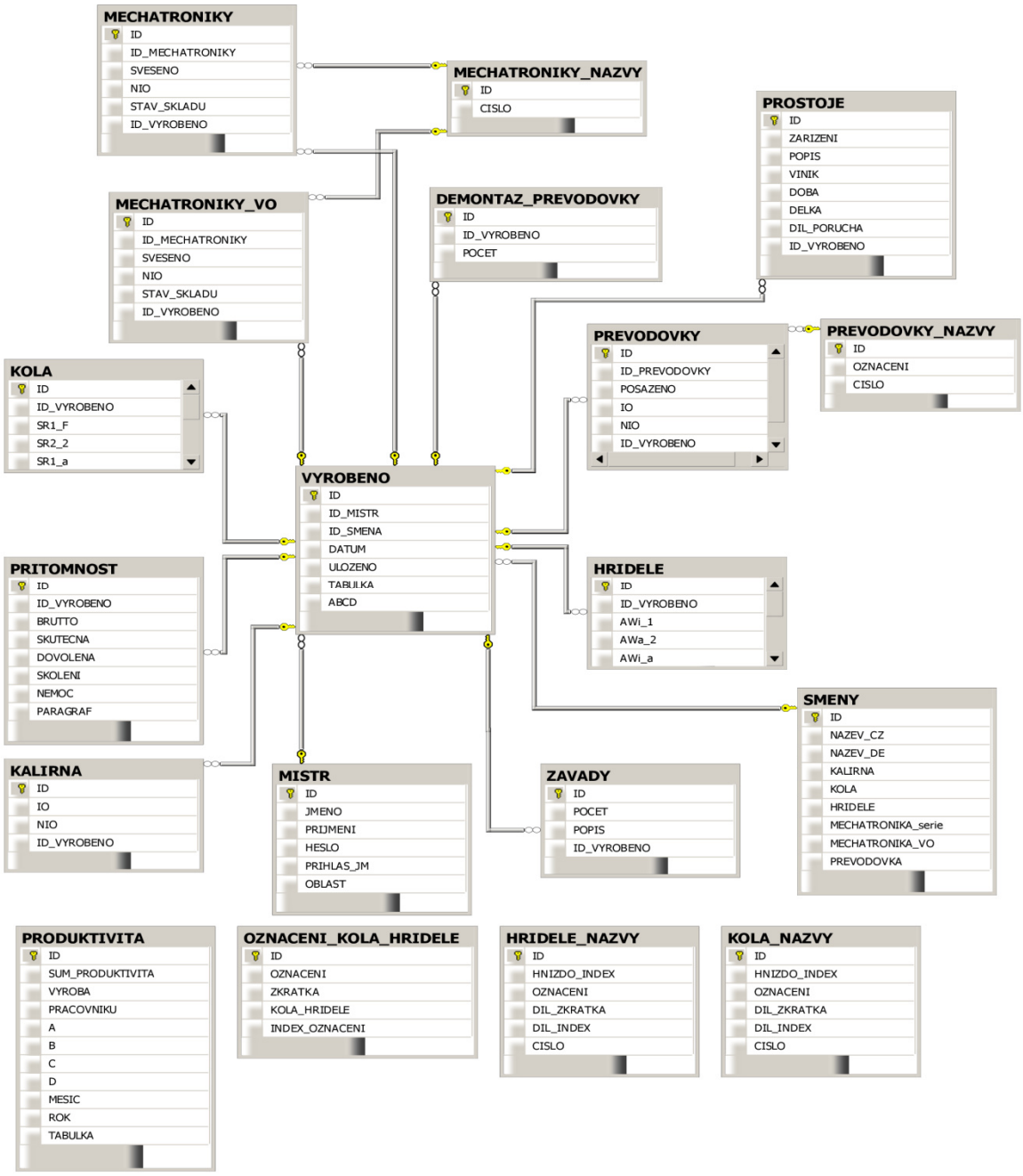
Sekce nastavení je přístupná pouze správci aplikace a je společná pro všechny výrobní oblasti. Ostatní sekce jsou přístupné všem uživatelů a mají rozdílný vzhled pro různé výrobní oblasti. Tabulkové reporty jsou sumarizací počtů vyrobených kusů, přítomnosti, produktivity a výčtem závad a prostojů v určeném časovém období. Grafické reporty jsou grafy za jednotlivé měsíce zobrazující produktivitu, počty kusů nebo přítomnost zaměstnanců. Pro reporty tabulkové i grafické je zde možnost tisku.

4 Databáze Výroba

Databázová část programu je tvořena databází Výroba. V této databázi jsou uložena všechna data potřebná pro funkci celé aplikace.

4.1 ER diagram

Na ER diagramu je znázorněna struktura celé databáze Výroba a vztahy mezi jednotlivými tabulkami. Vazby mezi tabulkami a detailní informace jsou popsány v dalších podkapitolách.



Obrázek 3: ER diagram databáze Výroba

4.2 Popis tabulek a jejich vztahů

Z ER diagramu jsme mohli zjistit základní informace o tabulkách a vazbách mezi nimi. V této podkapitole jsou popsány více do hloubky.

4.2.1 Vyrobeno

Tabulka Vyrobeno je stěžejní pro celou databázi. Na tuto tabulku se odkazuje většina ostatních tabulek, protože obsahuje základní informace o záznamech. Sloupec ID je primárním klíčem této tabulky a s každým záznamem se jeho hodnota automaticky zvyšuje o jedničku. Sloupec ID_MISTR je informace o mistrovi, ke které je záznam přiřazen. Je též cizím klíčem s vazbou na primární klíč tabulky Mistr. Sloupec ID_SMENA je informace o směně z hlediska času, ke které je záznam přiřazen. Je též cizím klíčem s vazbou na primární klíč tabulky Smena. Sloupec DATUM je informace o dni, ke kterému má být záznam přiřazen. Sloupec ZALOZENO je kontrolní informace o tom, kdy byl záznam opravdu uložen. Sloupec TABULKA je číslo odkazující na oblast výroby, ke které záznam patří. Sloupec ABCD je informace o tom, ke které směně z hlediska produktivity patří.

Tabulka 1: Vyrobeno

Vyrobeno		
ID	Int	Identity, Primary key
ID_MISTR	Int	Foreign key
ID_SMENA	Int	Foreign key
DATUM	Date	
ULOZENO	DateTime	
TABULKA	Smallint	
ABCD	Char(1)	

4.2.2 Mistr

V této tabulce se ukládají informace o mistrech, jako je jméno, příjmení, přihlašovací jméno, heslo a výrobní oblast, do které patří. Podle oblasti, do které patří, se v aplikaci přiřazují přístupová práva. ID těchto záznamů se také přiřazují do tabulky Vyrobeno, dle toho kdo záznam uložil.

Tabulka 2: Mistr

Mistr		
ID	Int	Identity, Primary key
JMENO	Nvarchar(30)	
PRIJMENI	Nvarchar(30)	
HESLO	Nvarchar(50)	
PRIHLAS_JM	Nvarchar(50)	
OBLAST	Smallint	

4.2.3 Smeny

Tabulka obsahuje názvy směn v českém a německém jazyce a výrobní oblasti. V této tabulce najdeme informace, o druhu směn, v kterých každá oblast vyrábí. ID těchto směn se také přiřazují do tabulky Vyrobeno, dle toho kdo záznam uložil. Tato tabulka musí být správně vyplněna pro správnou funkci aplikace. Záznamy se z ní pomocí aplikace nedají mazat ani přidávat. Záznamy se mohou změnit pouze ve sloupcích výrobních oblastí.

Tabulka 3 : Smeny

Smeny		
ID	Int	Identity, Primary key
NAZEV_CZ	Nvarchar(50)	
NAZEV_DE	Nvarchar(50)	
KALIRNA	Bit	
KOLA	Bit	
HRIDELE	Bit	
MECHATRONIKA_serie	Bit	
MECHATRONIKA_VO	Bit	
PREVODOVKA	Bit	

4.2.4 Produktivita

Do tabulky produktivita se ukládají záznamy o plánované produktivitě. Jsou to záznamy o počtu plánovaných pracovníků na směnu a celkem, plánovaných kusů celkem a vypočtené produktivitě. Dále pak měsíc a rok a tabulka. Tabulka znázorňuje výrobní oblast, pro kterou je záznam určen. Tato tabulka není provázána s jinou tabulkou.

Tabulka 4: Produktivita

Produktivita		
ID	Int	Identity, Primary key
SUM_PRODUKTIVITA	Decimal(10,3)	
VYROBA	Int	
PRACOVNIKU	Smallint	
A	Smallint	
B	Smallint	
C	Smallint	
D	Smallint	
MESIC	Smallint	
ROK	Smallint	
TABULKA	Smallint	

4.2.5 Pritomnost

Zde se ukládají záznamy o počtu pracovníků plánovaných a skutečně přítomných a absence. Absence se dělí na nemoc, dovolenou, paragraf a školení. Tato tabulka má cizí klíč ID_VYROBENO a odkazuje na ID v tabulce Vyrobeno.

Tabulka 5: Pritomnost

Pritomnost		
ID	Int	Identity, Primary key
ID_VYROBENO	Int	Foreign key
BRUTTO	Smallint	
SKUTECNA	Smallint	
DOVOLENA	Smallint	
SKOLENI	Smallint	
NEMOC	Smallint	
PARAGRAF	Smallint	

4.2.6 Zavady

Do tabulky Zavady se ukládají záznamy o závadách. Je v ní uložen počet a popis závad a odkazuje se cizím klíčem ID_VYROBENO na ID v tabulce Vyrobeno.

Tabulka 6: Zavady

Zavady		
ID	Int	Identity, Primary key
POCET	Int	
POPIS	Nvarchar(50)	
ID_VYROBENO	Int	Foreign key

4.2.7 Prostoje

V této tabulce jsou data o prostojích ve výrobě. Prostoje se dělí na prostoje strojů a zařízení a na prostoje dílů vadných nebo chybějících. Toto rozdělení je informace ve sloupci DIL_PORUCHA. Pro díl je to logická hodnota 0 a pro poruchu logická hodnota 1. Záznam se skládá z popisu, viníka, zařízení, délky prostoje v minutách, doby prostoje v minutách a informaci, jestli je to prostoj stroje nebo dílu.

Tabulka 7: Prostoje

Prostoje		
ID	Int	Identity, Primary key
ZARIZENI	Nvarchar(50)	
POPIS	Nvarchar(50)	
VINIK	Nvarchar(50)	
DOBA	Int	
DELKA	Int	
DIL_PORUCHA	Bit	
ID_VYROBENO	Int	Foreign key

4.2.8 Kalirna

Do této tabulky se ukládají počty kusů vyrobených v kalírně. Je tvořena informacemi o počtu dobře vyrobených kusů, počtu špatně vyrobených kusů a cizím klíčem ID_VYROBENO, který odkazuje na ID v tabulce Vyrobeno.

Tabulka 8: Kalirna

Kalirna		
ID	Int	Identity, Primary key
IO	Smallint	
NIO	Smallint	
ID_VYROBENO	Int	Foreign key

4.2.9 Kola

Do této tabulky se přidávají sloupce dle toho, jaké další typy ozubených kol se přidávají do výroby. Další sloupce se vytvářejí pomocí procedury PRIDEJ_KOLO. Do těchto sloupců se ukládají počty vyrobených kusů ozubených kol. Vytvořené sloupce mají název tvořený podle vzorce:

$$[1] \qquad název\ sloupce = zkratka\ označení + \_ + \ index\ kola$$

ID_VYROBENO je cizí klíč, který odkazuje na ID v tabulce Vyrobeno.

Tabulka 9: Kola

Kola		
ID	Int	Identity, Primary key
ID_VYROBENO	Int	Foreign key

4.2.10 Kola_nazvy

V této tabulce se ukládají informace o typech ozubených kol. Záznamy se sem vkládají pomocí procedury PRIDEJ_KOLO. Záznamy se skládají z indexu označení, označení, zkratky označení, čísla dílu a indexu dílu.

Tabulka 10: Kola_nazvy

Kola_nazvy		
ID	Int	Identity, Primary key
HNIZDO_INDEX	Smallint	
OZNACENI	Nvarchar(30)	
DIL_ZKRATKA	Nvarchar(10)	
DIL_INDEX	Nvarchar(10)	
CISLO	Nvarchar(50)	

4.2.11 Hridele

S touto tabulkou se pracuje stejně jako s tabulkou Kola, ale sloupce se přidávají pomocí procedury PRIDEJ_HRIDEL. Názvy sloupců jsou tvořeny podle vzorce:

[2] $$název\ sloupce = zkratka\ označení + \_ + \ index\ hřídele$$

Tabulka 11: Hridele

Hridele		
ID	Int	Identity, Primary key
ID_VYROBENO	Int	Foreign key

4.2.12 Hridele_nazvy

Tato tabulka je stejná jako tabulka Kola_nazvy. Záznamy se do ní ukládají pomocí procedury PRIDEJ_HRIDEL.

Tabulka 12: Hridele_nazvy

Hridele_nazvy		
ID	Int	Identity, Primary key
HNIZDO_INDEX	Smallint	
OZNACENI	Nvarchar(30)	
DIL_ZKRATKA	Nvarchar(10)	
DIL_INDEX	Nvarchar(10)	
CISLO	Nvarchar(50)	

4.2.13 Mechatroniky

V tabulce Mechatroniky jsou záznamy o vyrobených mechatronikách pro sériovou výrobu automobilky. Záznamy se vedou o počtu správně a nesprávně vyrobených kusů a stavu skladu. V této tabulce je také cizí klíč ID_MECHATRONIKY odkazující na ID v tabulce Mechatroniky_nazvy, kde jsou uloženy typy mechatronik. Dalším cizím klíčem je ID_VYROBENO odkazující na ID v tabulce Vyrobeno.

Tabulka 13: Mechatroniky

Mechatroniky		
ID	Int	Identity, Primary key
ID_MECHATRONIKY	Int	Foreign key
SVESENO	Smallint	
NIO	Smallint	
STAV_SKLADU	Smallint	
ID_VYROBENO	Int	Foreign key

4.2.14 Mechatroniky_VO

V tabulce Mechatroniky_VO jsou záznamy vyrobených mechatronik, které se nepoužívají pro sériovou výrobu automobilky. Struktura této tabulky je totožná s tabulkou Mechatroniky. Tabulka je vytvořena pouze z důvodů přehlednosti databáze.

Tabulka 14: Mechatroniky_VO

Mechatroniky_VO		
ID	Int	Identity, Primary key
ID_MECHATRONIKY	Int	Foreign key
SVESENO	Smallint	
NIO	Smallint	
STAV_SKLADU	Smallint	
ID_VYROBENO	Int	Foreign key

4.2.15 Mechatroniky_nazvy

Do této tabulky se vkládají typy mechatronik, které se vyrábějí. Obsahuje pouze sloupce s identifikačním číslem ID a označením mechatroniky.

Tabulka 15: Mechatroniky_nazvy

Mechatroniky_nazvy		
ID	Int	Identity, Primary key
CISLO	Nvarchar(50)	

4.2.16 Prevodovky

Tato tabulka je velmi podobná tabulce Mechatroniky a Mechatroniky_VO. Vkládají se sem záznamy o smontovaných převodovkách. Záznamy se skládají z informací o počtu správně a nesprávně vyrobených kusů a počtu osazených převodovek. V tabulce je také cizí klíč ID_PREVODOVKY odkazující na ID v tabulce Prevodovky_nazvy, kde jsou uloženy typy převodovek. Dalším cizím klíčem je ID_VYROBENO odkazující na ID v tabulce Vyrobeno.

Tabulka 16: Prevodovky

Prevodovky		
ID	Int	Identity, Primary key
ID_PREVODOVKY	Int	Foreign key
POSAZENO	Smallint	
IO	Smallint	
NIO	Smallint	
ID_VYROBENO	Int	Foreign key

4.2.17 Prevodovky_nazvy

Tabulka obsahuje záznamy o vyráběných převodovkách. A to ID převodovky, označení převodovky a číslo převodovky.

Tabulka 17: Prevodovky_nazvy

Prevodovky_nazvy		
ID	Int	Identity, Primary key
OZNACENI	Nvarchar(50)	
CISLO	Nvarchar(50)	

4.2.18 Demontaz_prevodovky

Zde se ukládá pouze počet kusů demontovaných převodovek a cizí klíč ID_VYROBENO, který odkazuje na ID v tabulce Vyrobeno.

Tabulka 18: Demontaz_prevodovky

Demontaz_prevodovky		
ID	Int	Identity, Primary key
ID_VYROBENO	Int	Foreign key
POCET	Int	

4.2.19 Oznaceni_kola_hridele

V této tabulce jsou informace o základním rozdělení kol a hřídelí. Podle těchto informací se vytváří názvy sloupců v tabulkách Kola a Hridele.

Tabulka 19: Oznaceni_kola_hridele

Oznaceni_kola_hridele		
ID	Int	Identity, Primary key
OZNACENI	Nvarchar(30)	
ZKRATKA	Nvarchar(10)	
KOLA_HRIDELE	Bit	
INDEX_OZNACENI	Smallint	

4.3 Uložené procedury v databázi

Databáze obsahuje uložené procedury, což jsou objekty, které neobsahují data, ale část programu, který se vykonává nad databázovými daty. Jedná se o část programu, který je jasně funkčně oddělený od svého okolí, má seznam parametrů pro komunikaci s jinými moduly programu. Může mít vlastní lokální proměnné neviditelné pro ostatní části programu.

4.3.1 PRIDEJ_KOLO

Tato procedura se volá kvůli přidání nového ozubeného kola do databáze. Volá se s parametry @NEWINDEX (nvarchar(10)), @NEWOZNACENI (nvarchar(30)), @NEWDIL (nvarchar(30)). Pomocí těchto parametrů se najdou další odpovídající záznamy z tabulky Oznaceni_kola_hridele. Pomocí všech těchto proměnných se vloží záznam do tabulky Kola_nazvy a vloží se sloupec s korektním názvem do tabulky Kola. Na konci procedury musí být příkaz COMMIT TRANSACTION, který potvrdí změny v databázi.

```
ALTER PROCEDURE PRIDEJ_KOLO
(
      @NEWINDEX NVARCHAR(10),
      @NEWOZNACENI NVARCHAR(30),
      @NEWDIL   NVARCHAR(30)
)
AS
BEGIN TRANSACTION
      DECLARE @NEWHNIZDO INT
      DECLARE @NEWZKRATKAOZNACENI NVARCHAR(30)
      DECLARE @PRIDEJ NVARCHAR(200)
      SET @NEWHNIZDO = (SELECT INDEX_OZNACENI FROM
OZNACENI_KOLA_HRIDELE WHERE OZNACENI=@NEWOZNACENI)
      SET @NEWZKRATKAOZNACENI=(SELECT ZKRATKA FROM
OZNACENI_KOLA_HRIDELE WHERE OZNACENI=@NEWOZNACENI)
      INSERT INTO KOLA_NAZVY (HNIZDO_INDEX, OZNACENI,
DIL_ZKRATKA, DIL_INDEX, CISLO) VALUES(@NEWHNIZDO ,
@NEWOZNACENI, @NEWZKRATKAOZNACENI, @NEWINDEX, @NEWDIL)
      SET @PRIDEJ = 'ALTER TABLE [KOLA] ADD  [' +
@NEWZKRATKAOZNACENI + '_' + @NEWINDEX + '] SMALLINT'
      EXECUTE (@PRIDEJ)
COMMIT TRANSACTION
```

4.3.2 PRIDEJ_ HRIDEL

Toto je obdobná procedura jako PRIDEJ_KOLO, ale pracuje s daty o hřídelích.

```
ALTER PROCEDURE PRIDEJ_HRIDEL
(
      @NEWINDEX NVARCHAR(10),
      @NEWOZNACENI NVARCHAR(30),
      @NEWDIL   NVARCHAR(30)
)
AS
BEGIN TRANSACTION
      DECLARE @NEWHNIZDO INT
      DECLARE @NEWZKRATKAOZNACENI NVARCHAR(30)
      DECLARE @PRIDEJ NVARCHAR(200)
      SET @NEWHNIZDO = (SELECT INDEX_OZNACENI FROM
OZNACENI_KOLA_HRIDELE WHERE OZNACENI=@NEWOZNACENI)
      SET @NEWZKRATKAOZNACENI=(SELECT ZKRATKA FROM
OZNACENI_KOLA_HRIDELE WHERE OZNACENI=@NEWOZNACENI)
      INSERT INTO HRIDELE_NAZVY (HNIZDO_INDEX, OZNACENI,
DIL_ZKRATKA, DIL_INDEX, CISLO) VALUES(@NEWHNIZDO ,
@NEWOZNACENI, @NEWZKRATKAOZNACENI, @NEWINDEX, @NEWDIL)
      SET @PRIDEJ = 'ALTER TABLE [HRIDELE] ADD [' +
@NEWZKRATKAOZNACENI + '_' +  @NEWINDEX + '] SMALLINT'
      EXECUTE (@PRIDEJ)
COMMIT TRANSACTION
```

5 Aplikace Vyroba

V této kapitole je popsána aplikace Vyroba. Každá aplikace obsahuje třídy, které jsou na sobě nějak závislé. V této kapitole jsou popsány vazby mezi třídami a jejich funkce. Dále jsou zde popsány některé důležité metody. Také je zde vysvětlení propojení databáze s aplikací.

Protože tato aplikace byla tvořena pro ŠKODA a.s., musí splňovat zadaná kritéria týkající se rozložení, barev a bezpečnosti. Jsou zde dvě úrovně uživatelů. Obě úrovně mají jiná práva přístupu do aplikace, tím pádem i do databáze. Těmito úrovněmi jsou:

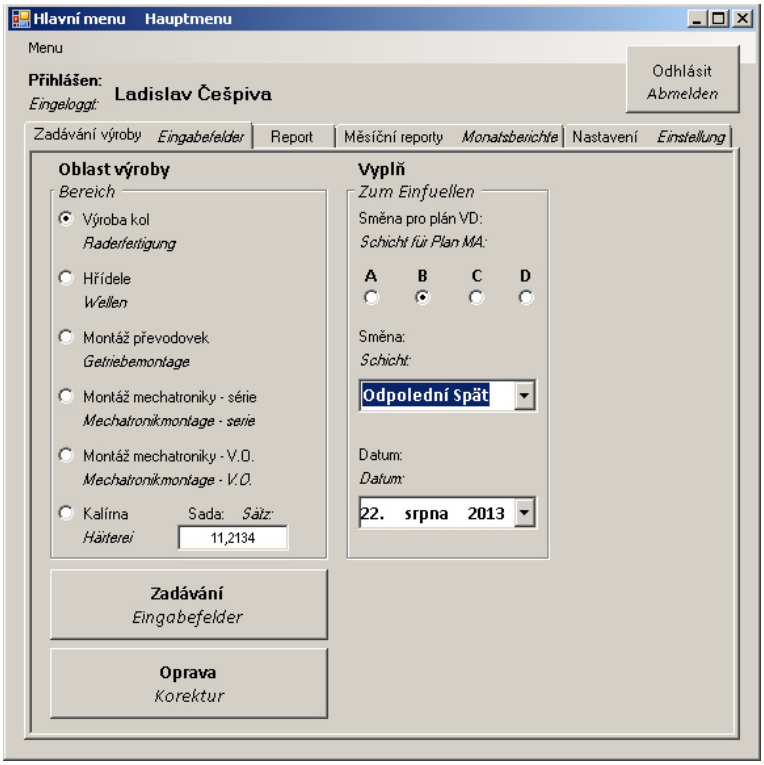
- Mistr má právo pouze na vkládání záznamů a opravu do jednoho dne po uložení záznamu.
- Správce má právo měnit nastavení, vkládat, opravovat a mazat záznamy.

5.1 Třída Main

Tato třída je dědicem třídy Windows.Forms a funguje jako hlavní menu celé aplikace. V tomto menu si můžete vybrat ze záložek, které odkazují na čtyři základní sekce aplikace. Jsou to zadávání výroby, tabulkové reporty, grafické reporty a nastavení. V těchto záložkách je vždy několik parametrů, které se ovládají různými ovládacími prvky.

- `private void IntervalReset()` – Nastaví možnost zadávání pouze ovládacím prvkům podle toho, který interval je zvolen. Je zde možno nastavit interval denní, měsíční a libovolný.
- `private int WeeksInYear(DateTime date, int add)` – Vrátí číslo týdne v roce, který je dán přičtením parametru add, který znázorňuje počet týdnů, k parametru date, který znázorňuje datum.
- `private DateTime DateOfWeek(int KW)` – Vrátí datum pondělí týdne určeného parametrem KW, který znázorňuje číslo týdne v roce.
- `private void BtPrihlasit_Click(object sender, EventArgs e)` – Spojí se s databází a porovná zadané heslo s heslem v databázi. Pokud jsou stejná zavolá konstruktor třídy Nastaveni a otevře ji v okně.
- `private void BtUnloggin_Click(object sender, EventArgs e)` – Odhlásí aktuálního uživatele. Poté zavolá třídu Login a otevře ji v okně.
- `private void BtReport_Click(object sender, EventArgs e)` – Zjistí, která výrobní oblast je zvolena a zavolá konstruktor třídy pro zobrazení záznamu této výrobní oblasti s parametry určenými ovládacími prvky.
- `private bool SmenaExist(int tabulka)` – Spojí se s databází a zjistí, zda zde existuje nějaký záznam výroby ze dne a směny, určené ovládacími prvky, pro oblast výroby určenou parametrem tabulka. Pokud záznam existuje, vrátí logickou 1. Tato metoda se používá na to, aby se do databáze nevkládaly duplicitní záznamy.
- `private string ABCD()` – Vrátí znak směny, která je zadána ovládacími prvky (Radiobutton).

- `private void BtOprav_Click(object sender, EventArgs e)` – Pomocí metody SmenaExist zjistí, zda k danému dni a směně existuje záznam. Dále zjistí, zda se tento záznam vztahuje k období posledního dne metodou MoznastOpravy. Pokud jsou tyto podmínky splněny, tak zavolá příslušnou třídu a její metody pro zobrazení tohoto záznamu, ve formátu, který je možné upravovat.
- `private bool MoznostOpravy(int tabulka)` – Tato metoda zjistí, zda záznam není starší než jeden den. Starší nesmí běžný uživatel opravovat.
- `private void BtLogUpdate_Click(object sender, EventArgs e)` – Tato metoda podle zadaných údajů v k tomu určených ovládacích prvcích změní údaje přihlášeného uživatele.
- `private void NactiSmeny(string SQL)` – Metoda, která načte směny do ComboBoxu, jako jeho nabídku.
- `private void menuProhlizec_Click(object sender, EventArgs e)` – Metoda, která zavolá třídu Prohlizec a otevře jí v okně.
- `private void menuExit_Click(object sender, EventArgs e)` – Ukončí celou aplikaci.
- `private int TabulkaGraf()` – Vybere výrobní oblast z ovládacích prvků (RadioButton), pro kterou se má graf zobrazit.
- `private void BtPlan_Click(object sender, EventArgs e)` – Zavolá konstruktor třídy Graf s parametry pro zobrazení vyrobených kusů ve výrobní oblasti a otevře ji v okně.
- `private void BtProd_Click(object sender, EventArgs e)` – Zavolá konstruktor třídy Graf s parametry pro zobrazení produktivity ve výrobní oblasti a otevře ji v okně.
- `private void BtPrit_Click(object sender, EventArgs e)` – Zavolá konstruktor třídy Graf s parametry pro zobrazení přítomnosti pracovníků ve výrobní oblasti a otevře ji v okně.
- `private void BtVAVPrit_Click(object sender, EventArgs e)` – Zavolá konstruktor třídy Graf s parametry pro zobrazení přítomnosti zaměstnanců všech výrobních oblastí a otevře ji v okně.
- `private void BtVAVProd_Click(object sender, EventArgs e)` – Zavolá konstruktor třídy Graf s parametry pro zobrazení produktivity všech výrobních oblastí a otevře ji v okně.



Obrázek 4: Hlavní menu aplikace

5.2 Třída Nastaveni

Třída Nastaveni je dědicem třídy Windows.Forms a její funkcí je nastavování parametrů výroby a aplikace. V této třídě se nachází metody na přidávání dílů a mistrů. Dále pak metody na práci s výrobním plánem, produktivitou a směnností. V neposlední řadě je zde funkce na změnu parametrů přihlášení, jako je přihlašovací jméno a heslo uživatele.

- `private void NactiSmennost()` – Načte tabulku Smeny, v které je uložena směnnost jednotlivých výrobních oblastí, do DataGridView DGVSmennost, kde se dá směnnost jednoduše upravit.
- `private void PlanVyroby()` – Načte plán výroby z tabulky Produktivita do tabulky a vypočítá produktivitu. Poté zapíše všechny hodnoty do požadovaného formátu do DataGridView DGVPlanVyroby.
- `private void NaplnComboBox(DataTable dt, ComboBox Cb)` – Volá se s parametrem DataTable dt, v kterém jsou uloženy názvy v tabulce a s parametrem Cb, který určuje ComboBox, do kterého se názvy zapíšou jako jeho položky.
- `private string PlanDatumFormat(bool plus)` – Plán výroby je vytvořen vždy na měsíc, který je zapsán v TextBoxu TbPlanDatum ve formátu „MM/RRRR“. Tato metoda se volá s parametrem plus, který určuje, zda se k danému číslu měsíce přičítá, nebo odčítá jedna. Návratová hodnota je upravené datum ve stejném formátu.
- `private int ZjistiMesic()` – Metoda, která zjistí měsíc ze stringu ve formátu „MM/RRRR“.

- `private int ZjistiRok()` – Metoda, která zjistí rok ze stringu ve formátu „MM/RRRR“.
- `private bool IsNumeric(object Expression)` – Metoda, která zjistí zda je parametr Expression číslo.
- `private void BtZmen_Click(object sender, EventArgs e)` – Spojí se databází a změní heslo pro přístup do nastavení.
- `private void BtNewUser_Click(object sender, EventArgs e)` – Spojí se s databází a přidá nového mistra. Mistrovy údaje se načítají z textboxů.
- `private void BtUlozPlan_Click(object sender, EventArgs e)` – Spojí se z databází a vymaže předchozí plán na daný měsíc, pokud existuje, a uloží nový do tabulky Produktivita.
- `private void DGVPlanVyroby_CellValueChanged(object sender, DataGridViewCellEventArgs e)` – Tato metoda se volá při změně hodnoty buňky v DataGridView DGVPlanVyroby. Poté se DGVPlanVyroby přepočítá podle daných pravidel.
- `private void BtSmennost_Click(object sender, EventArgs e)` – Uloží změny ve směnnosti do tabulky Smeny v databázi.
- `private void BtKolo_Click(object sender, EventArgs e)` – Vloží do databáze nový typ ozubeného kola.
- `private void BtHridel_Click(object sender, EventArgs e)` – Vloží do databáze nový typ hřídele.
- `private void BtPrevodovka_Click(object sender, EventArgs e)` – Vloží do databáze nový typ převodovky.
- `private void BtMechatronika_Click(object sender, EventArgs e)` – Vloží do databáze nový typ mechatroniky.

5.3 Třída Login

Třída Login je dědicem třídy Windows.Forms a její funkcí je přihlášení uživatele do aplikace. Tato třída se volá na začátku programu a při odhlášení uživatele. Přihlášení se provádí napsáním uživatelského jména a hesla do textboxů a po zmáčknutí tlačítka BtLogin se aplikace spojí s databází a načte údaje o uživatelském jménu. Pokud uživatelské jméno neexistuje nebo heslo není správné, tak se postup opakuje.

```
private void BtLogin_Click(object sender, EventArgs e)
{
     Connection con = new Connection();
     DataTable dt = con.SQLSelect("SELECT HESLO, JMENO,
               PRIJMENI, ID FROM MISTR WHERE PRIHLAS_JM = '"
               + TbName.Text + "';");
       if (dt.Rows.Count > 0)
       {
         if(Convert.ToString(dt.Rows[0][0])==TbPassword.Text)
             {
                  jmeno = Convert.ToString(dt.Rows[0][1]);
```

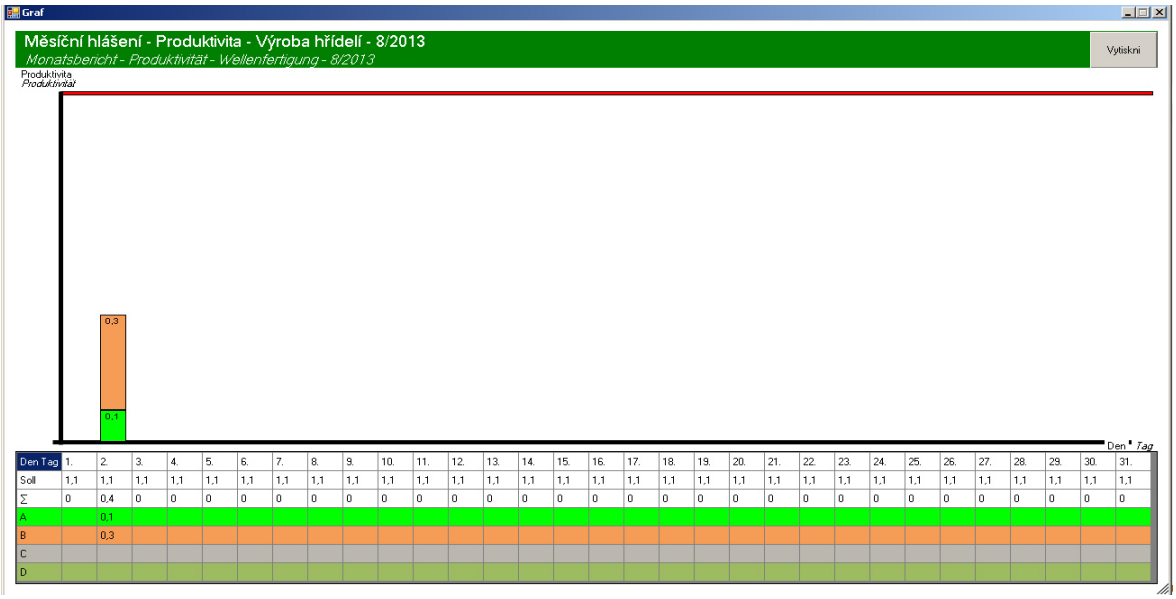
```
                    prijmeni = Convert.ToString(dt.Rows[0][2]);
                    id = Convert.ToInt32(dt.Rows[0][3]);
                    Close();
                }
            else
                {
                  MessageBox.Show("Špatné heslo nebo jméno!");
                }
        }
        else
           MessageBox.Show("Špatné jméno!");
}
```

5.4 Třída Graf

Třída Graf je dědicem třídy Windows.Forms a její funkcí je zobrazování grafických reportů a jejich tisk. Grafy se vykreslují jako sloupcové grafy a pod grafy se vykreslí tabulka s hodnotami. Je volána konstruktorem `public Graf(int typ, int month, int year)` pro zobrazení grafu všech oblastí výroby, nebo `public Graf(int typ, int month, int year, int tabulka)` pro zobrazení grafu jednotlivých výrobních oblastí. Vykreslují se grafy přítomnosti zaměstnanců, vyrobených kusů a produktivity.

- `private int ZjistiPocetDnuVMesici()` – Vrátí číslo odpovídající počtu dnů v daném měsíci.
- `private void GrafProdVAV1(int pocetDnu)`- Nastaví vzhled okna podle požadavků pro zobrazení všech výrobních oblastí. Dále zjistí počty kusů z dané tabulky a přítomnost pracovníků z tabulky Pritomnost v databázi a podle vzorců pro výpočet produktivity výrobní oblasti vypočítá hodnoty pro graf.
- `private void GrafPritVAV1(int pocetDnu)` - Nastaví vzhled okna podle požadavků pro zobrazení všech výrobních oblastí. Dále zjistí přítomnost pracovníků z tabulky Pritomnost v databázi. Tyto data upraví pro zobrazení v grafu.
- `private void GrafPritOblast(int pocetDnu, int tabulka)` - Nastaví vzhled okna podle požadavků pro zobrazení výrobní oblasti. Dále zjistí přítomnost pracovníků z tabulky Pritomnost v databázi. Tyto data upraví pro zobrazení v grafu.
- `private void GrafSadyOblast(int pocetDnu, int tabulka, double kalirna)` - Nastaví vzhled okna podle požadavků pro zobrazení výrobní oblasti. Dále zjistí počty kusů z dané tabulky. Tyto data upraví pro zobrazení v grafu.
- `private void GrafProduktivitaOblast(int pocetDnu, int tabulka, double kalirna)` - Nastaví vzhled okna podle požadavků pro zobrazení výrobní oblasti. Dále zjistí počty kusů z dané tabulky a přítomnost pracovníků z tabulky Pritomnost v databázi a podle vzorců pro výpočet produktivity výrobní oblasti vypočítá hodnoty pro graf.

- `private double[][] IntToDoubleArray(int[][] intArray)` - Tato metoda přetypovává pole datových proměnných integer na pole datových proměnných double.
- `private void VykresliGraf(double maximum, double[][] hodnoty, int mistaZaCarkou, int pocetSloupcu, double plan)`- Tato metoda je univerzální pro vykreslení všech grafů v aplikaci. Maximum znamená maximální zjištěnou hodnotu. Pole hodnoty jsou hodnoty, které se vykreslují. Pole je vícerozměrné, aby metoda byla schopna vykreslit skládaný sloupcový graf. Místa za desetinnou čárkou jsou zapsané v parametru mistaZaCarkou. Počet sloupců je v našem případě totožný s počtem dnů v měsíci a je dán parametrem pocetSloupcu. Pro zjištění počtu dnů v měsíci se volá metoda ZjistiPocetDnuVMesici. V grafu se vykresluje plánovaná hodnota přítomnosti, produktivity nebo vyrobených kusů, právě proto je zde parametr plan.
- `private void VyskaDGV()`- Upraví výšku DataGridView s tabulkou hodnot podle počtu řádků.
- `private string DotazSQLKolaHridele(bool kolaHridele)`- Tato metoda načte z databáze typy kol, nebo hřídelí. To je určeno parametrem kolaHridele a poté z nich vytvoří názvy sloupců. Z těchto názvů vytvoří část SQL dotazu pro zjištění počtu kusů kol nebo hřídelí.
- `private string SpocitejSmeny(int tabulka)` - Tato metoda vrací název tabulky v databázi podle parametru tabulka.
- `private string VyberTabulkuCz(int tabulka)`- Tato metoda vrací název výrobní oblasti v českém jazyce podle parametru tabulka.
- `private string VyberTabulkuDe(int tabulka)` - Tato metoda vrací název výrobní oblasti v německém jazyce podle parametru tabulka.
- `private Color VyberBarvu(int cislo)`- Tato metoda vrátí barvu podle parametru cislo.

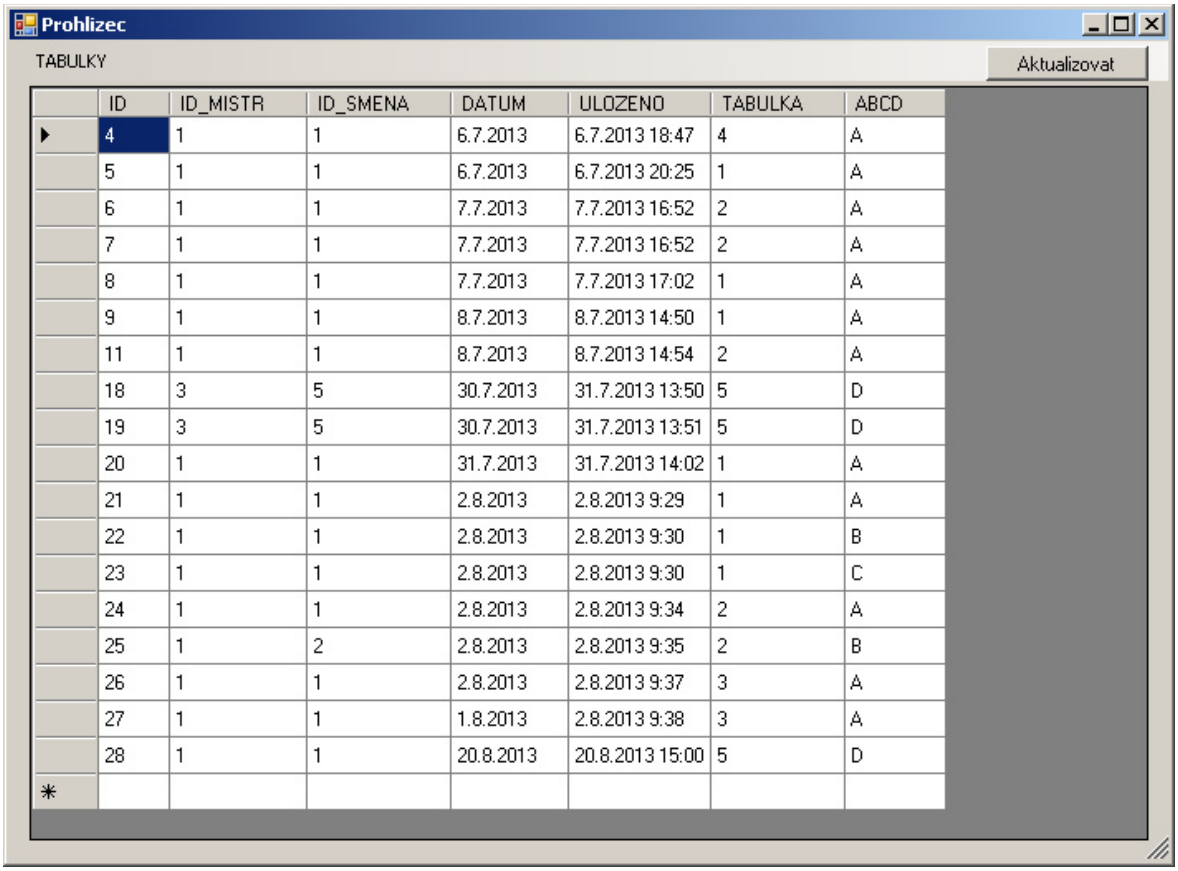


Obrázek 5: Vykreslení grafu

5.5 Třída Prohlizec

Tato třída, která je též dědicem Windows.Forms je vytvořena, jako jednoduchý databázový prohlížeč. Tato třída je určena pouze pro uživatele s přístupovým právem správce. Pomocí ní se dají zobrazit všechny bázové tabulky v databázi. Tyto tabulky se pak dají upravovat až na některé výjimky. Tabulka se načte do proměnné `private DataTable oldData`, která se nemění a zároveň jako `DataSource DataGridView DGVProhlizeni`, v kterém ji uživatel může změnit. Řádky v tabulkách se zde dají mazat, přidávat nebo jen změnit požadované buňky. Po úpravě tabulky se databáze upravuje SQL dotazy.

- `public Prohlizec()` - Konstruktor třídy Prohlizec. Se zavoláním tohoto konstruktoru se do ToolStripMenu načtou všechny bázové tabulky z databáse.
- `private void NactiTabulku()` - Načte požadovanou tabulku z databáze. Název této tabulky je uložen v globální proměnné této třídy `private string tabulka`. Tuto tabulku vloží do proměnné oldData a do DGVProhlizec jako DataSource.
- `private string UpdateRows()` - Tato metoda porovnává stará data v oldData a nová v DGVProhlizec. Pokud se najde stejné ID záznamu, ale jiné buňky se liší, tak vytvoří SQL příkaz pro změnu záznamu. Tyto dotazy se zaznamenávají do stringu za sebe. Po dokončení se vrací tento string.
- `private string DeleteRows()` - Tato metoda porovnává stará data v oldData a nová v DGVProhlizec. Pokud se ve starých datech najde záznam s ID, který v nových není, tak vytvoří SQL příkaz pro smazání záznamu. Tyto dotazy se zaznamenávají do stringu za sebe. Po dokončení se vrací tento string.
- `private string AddRows()` - Tato metoda porovnává stará data v oldData a nová v DGVProhlizec. Pokud se v nových datech najde záznam bez ID, tak vytvoří SQL příkaz pro zapsání tohoto záznamu. Tyto dotazy se zaznamenávají do stringu za sebe. Po dokončení se vrací tento string.



	ID	ID_MISTR	ID_SMENA	DATUM	ULOZENO	TABULKA	ABCD
▶	4	1	1	6.7.2013	6.7.2013 18:47	4	A
	5	1	1	6.7.2013	6.7.2013 20:25	1	A
	6	1	1	7.7.2013	7.7.2013 16:52	2	A
	7	1	1	7.7.2013	7.7.2013 16:52	2	A
	8	1	1	7.7.2013	7.7.2013 17:02	1	A
	9	1	1	8.7.2013	8.7.2013 14:50	1	A
	11	1	1	8.7.2013	8.7.2013 14:54	2	A
	18	3	5	30.7.2013	31.7.2013 13:50	5	D
	19	3	5	30.7.2013	31.7.2013 13:51	5	D
	20	1	1	31.7.2013	31.7.2013 14:02	1	A
	21	1	1	2.8.2013	2.8.2013 9:29	1	A
	22	1	1	2.8.2013	2.8.2013 9:30	1	B
	23	1	1	2.8.2013	2.8.2013 9:30	1	C
	24	1	1	2.8.2013	2.8.2013 9:34	2	A
	25	1	2	2.8.2013	2.8.2013 9:35	2	B
	26	1	1	2.8.2013	2.8.2013 9:37	3	A
	27	1	1	1.8.2013	2.8.2013 9:38	3	A
	28	1	1	20.8.2013	20.8.2013 15:00	5	D
*							

Obrázek 6 : Prohlížeč tabulek

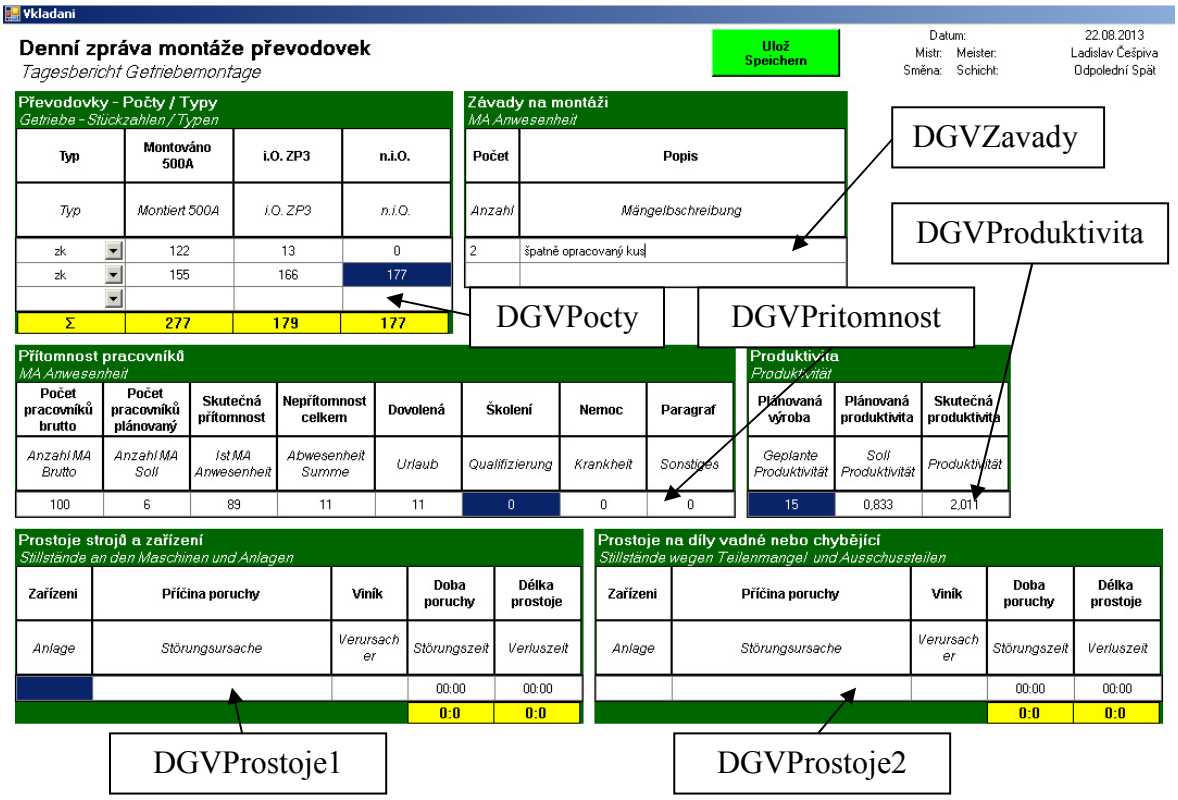
5.6 Třída Vkladani

Třída Vkladani je dědicem třídy Windows.Forms a její funkcí je zobrazení zadávací sekce i sekce tabulkových reportů. V této třídě se vyskytují metody společné pro všechny nebo většinu výrobních oblastí. Těmito metodami jsou ukládání a načítání záznamů prostojů, závad, přítomnosti pracovníků, grafické rozložení celého formu, tisk reportu a nastavení implicitních hodnot tabulek.

- `public Vkladani(string mistr, string smena, DateTime datum, int smenaId, int mistrId, int tabulka, string ABCD)` - Konstruktor třídy pro vkládání dat do databáze. V tomto konstruktoru se volají všechny metody potřebné pro nastavení okna.
- `public Vkladani(DateTime mindate, DateTime maxdate, int[] smeny, int tabulka)` - Konstruktor třídy pro načítání dat z databáze. V tomto konstruktoru se volají všechny metody potřebné pro nastavení okna.
- `protected void DGVSize(DataGridView dgv, bool sumy)` – Upraví velikost DataGridView z parametru dgv, tak aby odpovídala počtu řádků. V této metodě se volá druhá metoda SetPanelSize, pro kterou je důležitý parametr sumy.

- `protected void SetPanelSize(bool sumy)` – Upraví velikost všech panelů ve formu, tak aby odpovídaly velikosti DataGridView a byly zobrazeny v odpovídajícím formátu. Pokud je parametr sumy v logické „1“, tak se panel zvětší o velikost Textboxu pro sumy počtů.
- `private void ProstojeRowsAdded(DataGridView dgv)` – Nastaví formát poslednímu přidanému řádku v DataGridView podle parametru dgv. Používá se DGVProstoje1 a DGVProstoje2.
- `protected void SetTabulky(bool prostoj2, bool Bzavady)` – Nastaví formát všech panelů formu pomocí volání jednotlivých metod s odpovídajícími parametry.
- `private void PlanovaneHodnoty()`– Načte z databáze plánované hodnoty. Jsou to plánovaná produktivita, přítomnost a výroba. Poté je vloží do odpovídajících buněk v DataGridView.
- `protected void Nadpis(string[,] nadpis, DataGridView dgv,Panel pnl)` – Protože tato aplikace vyžaduje určitý formát, kterého nelze dosáhnout pouhými nadpisy sloupců DataGridView, tak se zde jako nadpisy sloupců vytváří Labely. Tyto Labely se vytváří ve dvou jazycích a určitém formátu. Parametr nadpis obsahuje názvy, dgv obsahuje odpovídající DataGridView a pnl odpovídající panel. Podle těchto informací se správně umístí a naformátují.
- `protected void FormatStandart(DataGridView dgv)` – Formátuje poslední přidaný řádek v dgv, aby se text zobrazoval vprostřed a výchozí hodnota byla 0.
- `protected string UlozProstoje(bool prostoje1_2)` – Generuje příkazy na uložení dat z DGVProstoje1 nebo DGVProstoje2. To je určeno parametrem prostoje1_2. Tyto generované příkazy se ukládají za sebe a jsou návratovou hodnotou metody.
- `protected string UlozPritomnost()`– Generuje příkaz na uložení dat z DGVPritomnost. Tento generovaný příkaz je návratovou hodnotou metody.
- `private int CasToInt(string str)` – Vtabulce Prostoje se uchovává doba i délka prostoje v integeru odpovídající minutám. V aplikaci se zobrazuje ve formátu „hh:mm“, proto je potřeba převést tento string na minuty a to je funkcí této metody. Str je ve výše zmíněném formátu a návratová hodnota jsou minuty.
- `protected void Nacteni(bool prostoje2, bool zavady)` – Tato metoda formátuje vzhled formu do formátu pro zobrazení tabulkových reportů a načte data společná pro všechny nebo většinu výrobních oblastí. Pokud oblast obsahuje prostoje dílů musí být parametr prostoje2 logická „1“. Pokud oblast obsahuje tabulku závad musí být parametr zavady logická „1“.
- `protected void NastavSirkuSloupce(DataGridView dgv, int[] sirka)` – Tato metoda nastavuje šířky sloupců, které jsou v parametru sirka, v DataGridView, určené parametrem dgv.
- `private void VymazSloupce(DataGridView dgv)` – Tato metoda vymaže sloupce z DataGridView, určené parametrem dgv.

- `private string SQLSmeny(int[] smeny)` – Tato metoda vrací část SQL dotazu, která odkazuje na požadované směny, které jsou zapsané v parametru smeny.
- `private void BtSaveTisk(object sender, EventArgs e)` – Po stisknutí tohoto tlačítka se nastaví základní nastavení tisku a spustí tisk.
- `private void printDocument1_PrintPage(object sender, PrintPageEventArgs e)` – Nastaví jednotlivé vlastnosti tisknutých stránek, jako číslo stránky, datum tisku a tisknutou oblast, která je určena metodou StranaTisk.
- `private Bitmap StranaTisk()` – Vrátí Bitmapu, na které jsou zobrazené požadované prvky z formu ve formátu pro tisk.



Obrázek 7: Ukázka rozložení ovládacích prvků třídy Vkladani

5.7 Třída KalirnaZ

Tato třída je dědicem třídy Vkladani a je určena pro vkládání a načítání dat výrobní oblasti kalírny. V této třídě se nacházejí metody, které jsou vhodné pouze pro práci s daty této výrobní oblasti.

- `public KalirnaZ(string mistr, string smena, DateTime datum, int smenaId, int mistrId, int tabulka, string ABCD, double koeficient): base(mistr, smena, datum, smenaId, mistrId, tabulka, ABCD)` - Konstruktor třídy pro zobrazení ve formátu pro vkládání záznamů výrobní oblasti kalírny.

- `public KalirnaZ(DateTime mindate, DateTime maxdate, int[] smeny, int tabulka) : base(mindate, maxdate, smeny, tabulka)` - Konstruktor třídy pro zobrazení ve formátu pro načítání tabulkových reportů výrobní oblasti kalírny.
- `private void NactiPocty()` - Metoda pro načtení počtů vyrobených kusů v kalírně a naformátování DGVPocty.
- `private void NactiProduktivitu()` - Načte plánovanou produktivitu a naformátuje DGVProduktivita pro zobrazení produktivity.
- `private void Suma(object sender,EventArgs e)` – Počítá součet správně a nesprávně vyrobených dílů při změně jejich počtu a zapisuje tuto hodnotu do tabulky.
- `private void VypocetProduktivity()` - Vypočte produktivitu z dat v tabulce DGVPritomnost a DGVPocty podle následujícího vzorce.

$$[3] \qquad produktivita\ kalírny = \frac{správně\ vyrobené\ kusy}{počet\ přítomných\ pracovníků}$$

- `private void LoadSet()` - Naformátuje DGVPocty a jeho popisky na vkládání hodnot z výroby.
- `private void FormatPoctyK()` - Naformátování řádků v DGVPocty pro zobrazení údajů z kalírny.
- `private void Uloz(object sender, EventArgs e)` - Tato metoda uloží záznam do tabulky Vyrobeno a zjistí jeho ID. Poté vygeneruje příkaz pro uložení vyrobených kusů kalírny a pomocí zděděných metod pro uložení ostatních údajů a následně je uloží.

5.8 Třida HridKolaZ

Tato třída je dědicem třídy Vkladani a je určena pro vkládání a načítání dat výrobních oblastí výroby ozubených kol a výroby hřídelí. V této třídě se nacházejí metody, které jsou vhodné pouze pro práci s daty těchto výrobních oblastí.

- `public HridKolaZ(string mistr, string smena, DateTime datum, int smenaId, int mistrId, int tabulka, string ABCD): base(mistr, smena, datum, smenaId, mistrId, tabulka, ABCD)` - Konstruktor třídy pro zobrazení ve formátu pro vkládání záznamů výrobní oblasti výroby hřídelí a ozubených kol.
- `public HridKolaZ(DateTime mindate, DateTime maxdate, int[] smeny, int tabulka): base(mindate, maxdate, smeny, tabulka)` - Konstruktor třídy pro zobrazení ve formátu pro načítání tabulkových reportů výrobní oblasti hřídelí a ozubených kol.
- `private void Produktivita()` - Vypočte produktivitu z dat v tabulce DGVPritomnost a DGVPocty podle následujících vzorců.

$$[4] \qquad produktivita\ výroby\ ozubených\ kol = \frac{správně\ vyrobené\ kusy/13}{počet\ přítomných\ zaměstnanců}$$

$$[5] \quad produktivita\ výroby\ hřídelí = \frac{správně\ vyrobené\ kusy/5}{počet\ přítomných\ zaměstnanců}$$

- `private void Uloz(object sender, EventArgs e)` - Tato metoda uloží záznam do tabulky Vyrobeno a zjistí jeho ID. Poté vygeneruje příkaz pro uložení vyrobených kusů ozubených kol nebo hřídelí a pomocí zděděných metod pro uložení ostatních údajů a následně je uloží.
- `private void LoadSet()` - Naformátuje DGVPocty a jeho popisky na vkládání hodnot z výroby.
- `private void NaplnSloupce()` - Načte typy ozubených kol nebo hřídelí a zapíše je do globálního pole struktury `private Sloupce[] s`.
- `private void PoctySloupce(bool nacitani)` - Z globálního pole `private Sloupce[] s` udělá popisky k DGVPocty.
- `private void NacistPocty()` - Načte z databáze počty kusů do DGVPocty a zformátuje je do požadovaného vzhledu.
- `private void NactiProduktivitu()` - Načte plánovanou produktivitu a naformátuje DGVProduktivita pro zobrazení produktivity.

```
private struct Sloupce
{
      public int pocet;
      public string hnizdo, index;
      public Sloupce(int pocet, string hnizdo, string index)
      {
            this.pocet = pocet;
            this.hnizdo = hnizdo;
            this.index = index;
      }
}
```

5.9 Třída PreMechZ

Tato třída je dědicem třídy Vkladani a je určena pro vkládání a načítání dat výrobních oblastí montáže mechatronik pro sériovou i mimo sériovou výrobu a montáže převodovek. V této třídě se nacházejí metody, které jsou vhodné pouze pro práci s daty těchto výrobních oblastí.

- `public PreMechZ(string mistr, string smena, DateTime datum, int smenaId, int mistrId, int tabulka, string ABCD): base(mistr, smena, datum, smenaId, mistrId, tabulka, ABCD)` - Konstruktor třídy pro zobrazení ve formátu pro vkládání záznamů výrobní oblasti montáže převodovek a montáže mechatronik pro sériovou i mimo sériovou výrobu.
- `public PreMechZ(DateTime mindate, DateTime maxdate, int[] smeny, int tabulka) : base(mindate, maxdate, smeny, tabulka)` - Konstruktor třídy pro zobrazení ve formátu pro načítání

tabulkových reportů výrobní oblasti montáže převodovek a montáže mechatronik pro sériovou i mimo sériovou výrobu

- `private void VypoctiProduktivitu()` - Vypočte produktivitu z dat v tabulce DGVPritomnost a DGVPocty podle následujících vzorců.

$$[6] \quad produktivita\ montáže\ mechatronik = \frac{správně\ vyrobené\ kusy}{počet\ přítomných\ zaměstnanců}$$

$$[7] \quad produktivita\ montáže\ převodovek = \frac{správně\ vyrobené\ kusy}{počet\ přítomných\ zaměstnanců}$$

- `private void Uloz(object sender, EventArgs e)` - Tato metoda uloží záznam do tabulky Vyrobeno a zjistí jeho ID. Poté vygeneruje příkaz pro uložení montovaných kusů převodovek nebo mechatronik a pomocí zděděných metod a metody UlozZavady pro uložení ostatních údajů a následně je uloží.
- `private string UlozZavady()` - Tato metoda generuje SQL příkazy pro uložení záznamů závad do stringu. Tyto záznamy se řadí za sebe a jsou návratovou hodnotou této funkce.
- `public void LoadSet()` - Naformátuje DGVPocty a jeho popisky na vkládání hodnot z výroby.
- `private void AddRowPoctyPM()` - Nastaví požadovaný formát posledního přidaného řádku DGVPocty. Zavolá zděděnou metodu DGVSize, která upraví velikost DGVPocty a posune všechny prvky formu, tak aby byly čitelné.
- `private void NactiPocty()` - Načte z databáze počty kusů do DGVPocty a zformátuje je do požadovaného vzhledu.
- `private void NactiProduktivitu()` - Načte plánovanou produktivitu a naformátuje DGVProduktivita pro zobrazení produktivity.

5.10 Třída Connection

Pro komunikaci s databází je v aplikaci tato třída. Tato třída dále využívá třídu System.Data.SqlClient. Spojení s databází se provádí pomocí třídy System.Data.SqlClient a stringu. Tento string obsahuje uživatelské id, heslo, server nebo adresu nebo data source, důvěrné spojení, timeout, název databáze nebo původní katalog. Pro účely testování na mém počítači s MS SQL server s databází Vyroba lze použít string: „`Server = CZESHA-PC; Database = Vyroba; User Id = sa; Password = masterkey`“. Pro připojení do databáze na vzdáleném serveru je potřeba string: „`Data Source = ipAdresa, port; NetworkLibrary = DBMSSOCN; Initial Catalog = myDataBase; User ID = myUsername; Password = myPassword`“ [5], kde ipAdresa je IP adresa vzdáleného serveru s databází a myDataBase je název databáze. V této třídě se nachází čtyři metody.

- `public void SQLWrite(string SQLstring)` – Tato metoda se používá na zápis dat do databáze nebo jejich update. V SQL jazyce to jsou příkazy INSERT, UPDATE nebo DELETE. Volá se s parametrem SQLstring, který obsahuje příkaz psaný v jazyce SQL. Spojí se s databází a poté provede příkaz voláním metody ExecuteNonQuery().

- `public int SQLWriteID(string SQLstring)` - Tato metoda je téměř stejná jako SQLWrite, ale je doplněna o příkaz „`SELECT @@IDENTITY`“, který vrátí poslední ID vložené do databáze. Tato metoda se používá na vkládání záznamů do tabulky Vyrobeno a vrátí poslední hodnotu ID, která se dále používá v aplikaci. Načtení ID je provedeno metodou ExecuteScalar().

```
public int SQLWriteID(string SQLstring)
{
      SqlConnection sqlConnection = new
      SqlConnection(ConnectionString);
      SqlCommand sqlCommand = new SqlCommand(SQLstring,
      sqlConnection);
      int id = 0;
      try
      {
            sqlConnection.Open();
            sqlCommand.ExecuteNonQuery();
            sqlCommand.CommandText = "SELECT @@IDENTITY";
            id = Convert.ToInt32(sqlCommand.ExecuteScalar());
      }
      catch (SqlException ex)
      {
           MessageBox.Show(ex.ToString());
      }
      sqlConnection.Close();
}
```

- `public DataTable SQLSelect(string SQLstring)` - Metoda SQLSelect je volána s parametrem SQLString, což je dotaz napsaný v jazyce SQL, který vrací více, jak jednu hodnotu, to znamená, že vrácený objekt je DataTable. V SQL jazyce to znamená příkaz SELECT. Příkaz se provede vytvořením a naplněním objektu SqlDataAdapter. Poté je z tohoto objektu naplněn DataTable.
- `public object SQLSel(string SQLstring)` - Metoda SQLSel je obdobná jako metoda SQLSelect, ale vrací pouze jednu hodnotu. Tato hodnota je vrácena, jako objekt a dále se v programu musí přetypovat. Příkaz se vykonává pomocí metody ExecuteScalar().

Závěr

Podařilo se udělat aplikaci pro práci s daty podle požadavků společnosti ŠKODA AUTO a.s. Součástí této aplikace je databáze Vyroba, která byla vytvořena podle pravidel návrhu a tvorby relačních databází. Součástí této aplikace byl také vytvořen objektově orientovaný program na práci s těmito daty, který byl napsán v jazyce C#.

Takto zpracovaná aplikace je připravená na provoz v reálném prostředí. Aplikace je odzkoušená, ale nemohu vyloučit, že zde nezůstala nějaká neošetřená chyba. To už je otázka testování v reálném prostředí.

Během tvorby této aplikace jsem narazil na problém s tiskem jednotlivých oken. Jednotlivé ovládací prvky, které je potřeba vytisknout se vykreslí do jednoho obrázku, který se poté postupně tiskne, ale kvalita tisku není plně dostačující. Pro zlepšení kvality tisku by bylo třeba tuto část programu vylepšit.

Nevýhodou této aplikace je stálé nastavení pro připojení programu do databáze. Pokud by byla databáze přenesena na jiný server nebo pokud by vzdálený server změnil svoji IP adresu, musela by se udělat úprava kódu programu. Tato nevýhoda by šla odstranit možností nastavení připojení při spuštění aplikace.

Tvorba této aplikace mě obohatila o nové zkušenosti v návrhu relačních databází. V neposlední řadě jsem získal znalosti i o tvorbě rozsáhlejší aplikace, napsané dle pravidel objektově orientovaného programování.

Literatura

[1] STEPHENS, Ryan K, Ronald R PLEW a Arie JONES. *Naučte se SQL za 28 dní*. Vyd. 1. Překlad Lukáš Krejčí. Brno: Computer Press, 2010, 728 s. ISBN 978-80-251-2700-1.

[2] HANÁK, Ján. *C# 3.0: Programování na platformě .NET 3.5*. Vyd. 1. Brno: Zoner Press, 2009, 282 s. ISBN 978-80-7413-046-5.

[3] BISHOPOVÁ, Judith. *C#: návrhové vzory*. Vyd. 1. Brno: Zoner Press, 2010, 323 s. ISBN 978-80-7413-076-2.

[4] MICROSOFT. *MSDN* [online]. 2013 [cit. 2013-08-15]. Dostupné z: http://msdn.microsoft.com/cs-cz/ms348103.aspx

[5] SQL Server 2012 connection strings. *Connection strings* [online]. 2012 [cit. 2013-08-15]. Dostupné z: http://www.connectionstrings.com/sql-server-2008

Obsah CD

- Bakalářská práce ve formátu PDF.
- Projekt Visual Studia pro aplikaci Vyroba.
- Kód pro vytvoření databáze v databaze.txt.