UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Aplikace pro podporu činnosti zkušebny

Michal Bárnet

Bakalářská práce
2014

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2013/2014**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal Bárnet**
Osobní číslo:	**I11006**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace pro podporu činnosti zkušebny**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit aplikaci, která bude sloužit pro evidenci zkoušek různých vzorků prováděných vývojovým oddělením. Aplikace by měla evidovat následující informace: seznam testovacích zařízení, seznam testovaných vzorků, evidence zaměstnanců, záznamy s průběhy jednotlivých zkoušek (data, protokoly, audiovizuální záznamy, ...), plánovač (kalendář s rozvrhem všech uskutečněných a naplánovaných zkoušek), reporty (generované sumáře dle specifikovaných parametrů).

Samotná praktická část bude realizována za využití programovacího jazyka JAVA a využití vybraného relačního databázového systému (MySQL, Oracle, ...).

Textová část bakalářské práce se bude skládat z teoretické části a z popisu praktické části. V teoretické části bude provedena rešerše přístupových metod jazyka JAVA do relačního databázového systému (JDBC, ORM, ...). Dále se teoretická část bude zabývat problematikou návrhu databázových modelů. V popisu praktické části bude nejprve popsána analýza navrhovaného řešení. Následně bude popsána samotná implementace aplikace spolu s praktickými poznatky z provozu aplikace.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

SCHILDT, Herbert. Java 7: výukový kurz. 1. vyd. Brno: Computer Press, 2012, 664 s. ISBN 978-80-251-3748-2.
CONOLLY, Thomas, Carolyn E BEGG a Richard HOLOWCZAK. Mistrovství - databáze: profesionální průvodce tvorbou efektivních databází. Vyd. 1. Brno: Computer Press, 2009, 584 s. ISBN 978-80-251-2328-7.
GROFF, James R, Paul N WEINBERG a Richard HOLOWCZAK. SQL: kompletní průvodce. Vyd. 1. Brno: CP Books, 2005, 936 s. ISBN 80-251-0369-2.

Vedoucí bakalářské práce: **Ing. Tomáš Váňa**
Katedra informačních technologií

Datum zadání bakalářské práce: **20. prosince 2013**
Termín odevzdání bakalářské práce: **9. května 2014**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2014

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 2. 5. 2014 Michal Bárnet

Poděkování

Rád bych zde poděkoval vedoucímu panu Ing. Tomáši Váňovi za poskytnuté informace, rady a vstřícný přístup při zpracování této bakalářské práce.

Děkuji také své rodině a přátelům za ochotu a podporu při mém studiu.

Anotace

Práce se zabývá programovacím jazykem Java a jeho přístupem k databázím. Je v ní provedena rešerše přístupových metoda jazyku Java do relačních databází. Dále je probrána teorie návrhu tří základních databázových modelů. Na závěr je představena aplikace, na které jsou předešlé teoretické znalosti převedeny do praxe.

Cílem práce bylo vytvořit GUI aplikace, která bude využita oddělení zkušebna pro potřeby uložení, editace a tvorbu sumářů z měřených dat. Aplikace spolupracuje s databází Oracle.

Klíčová slova

Java, JDBC, ORM, relační databáze, návrh databáze, Oracle, Java GUI aplikace

Title

Applications that support testing departments.

Annotation

This work is about programming language Java and database access. Access method to Java relational databases are described here. Next it attends to present the theory of three basic database models. Finally, work transfer previous theoretical knowledge into practice.

The aim was to create a GUI application that will use a testing department for the purpose of storing, editing and creating summaries of the measured data. The application works with Oracle databases.

Keywords

Java, JDBC, ORM, relational databases, database design, Oracle, Java GUI applications

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

API	Application Programming Interface. Programové rozhraní aplikace.
CSF	Content Secure Format. Zabezpečený formát obrázku.
DDL	Data Manipulation Language. Příkazy pro vytvoření struktury databáze.
DNS	Domain Name Systém. Systém doménových jmen. Každé jméno má přiřazenou IP adresu.
EJB	Enterprise Java Beans. Serverové komponenty s cílem oddělit logiku aplikace od programového rozhraní.
GUI	Graphical User Interface. Grafické uživatelské rozhraní.
IDE	Integrated Development Environment. Vývojové prostředí pro tvorbu programů.
J2EE	Java Platform, Enterprise Edition. Součást platformy Java určená pro tvorbu podnikových aplikací.
J2SE	Java Platform, Standard Edition. Verze Javy využívaná pro vývoj konzolových nebo serverových aplikací.
JDBC	Java Database Connectivity. Rozhraní pro přístup k databázím.
JDO	Java Data Objects. Specifikuje standartní metody pro přístup k persistentním datům.
JPOX	Java Persistent Objects. Implementace JDO.
JPQL	Java Persistence Query Language. Dotazovací jazyk podobný SQL. Vykonává dotazy nad objekty databáze.
ODBC	Open Database Connectivity. Softwarové API pro přístup k systémům databáze.
ORM	Object-relational mapping. Programovací technika pro převod objektů do databáze.
POJO	Plain Old Java Object. Označení objektu Java.
SDK	Software development kit. Sada nástrojů pro vývoj softwaru.
SQL	Structured Query Language. Strukturovaný jazyk využívaný pro práci s daty relační databáze.

UML	Unified Modeling Language. Grafický jazyk pro zobrazení navrhovaného systému.
URL	Uniform Resource Locator. Definovaná struktura pro identifikaci zdroje.
XML	Extensible Markup Language. Značkovací jazyk, určený převážně pro publikování dokumentů.

Úvod

Moderní firmy zaměřené na výrobu elektronických zařízení disponují vlastním vývojem svých výrobků. Nedílnou součástí vysoké kvality výrobků je podrobná kontrola kvality. Tuto činnost vykonává akreditovaná zkušebna. Zkušební oddělení denně generují mnoho měřených dat a protokolů.

Dnešní programové systémy umožňují tuto práci zpřesnit, zjednodušit, archivovat a generovat přehledné sumáře vhodné pro management firmy. Struktura aplikací pro správu dat je standardně rozdělena na dvě základní úrovně. První částí je databázový systém, schopný uchovávat persistentní data. Dnešní moderní společnost je založená na shromažďování nejrůznějších dat. Pro ukládání dat se v dnešní době využívá především relačních databázových systémů. Jedním z hlavních důvodů je standardizovaný strukturovaný dotazovací jazyk (SQL). Druhou část těchto systémů pak tvoří samotná klientská aplikace, která komunikuje s uživatelem a plní jeho požadavky. Těchto aplikací je dnes velké množství. V našem případě potřebujeme konkrétní funkce pro editace objemných dat. Je proto vhodné vybrat vyšší programovací jazyk, který splňuje naše kritéria pro přístup k databázi.

Cílem této práce je přiblížit problematiku těchto částí a ukázat návrh systémů prakticky i teoreticky. Pro praktickou demonstraci práce představí aplikaci Zkušebna v jazyce Java. Aplikace eviduje seznamy testovacích zařízení, vzorků, zaměstnanců a protokoly. Data jsou ukládána na databázový server Oracle.

V teoretické části práce popíše základní přístupové metody jazyka Java k relačním systémům. Především pak základní metodu pomocí rozhraní JDBC. Dále budou představeny kroky ke správnému vytvoření databázového modelu.

Bakalářská práce má následující strukturu. První kapitola popisuje rešerši přístupových metod do relační databáze. Ve druhé kapitole je popsán návrh databázového modelu. Třetí kapitola probere praktické zpracování GUI aplikace s přístupem do relační databáze Oracle Database 11g.

1 Přístupové metody jazyka JAVA do relačního databázového systému

1.1 Základní přístupové metody

Následující kapitola reprezentuje základní přístupové metody jazyka Java do relační databáze. Základní metodou je univerzální rozhraní JDBC. Rozsáhlejší aplikace (např. bankovní systémy) plné datových objektů, využívají metody objektově relačního mapování. Základem této techniky je rozhraní Java Persistence. Nakonec bude představena komplexní technika JDO pro univerzální přístup do systémů databáze.

1.2 Rozhraní JDBC

JDBC (Java Database Connectivity) [1][2] je objektové univerzální nízko úrovňové aplikační rozhraní (API) pro přístup k relačním databázím. Je standartní součástí Java SE a J2SE. Poskytuje vývojářům v jazyce Java přístup k širokému spektru SQL databází (např. Oracle, MySQL, SQL Server atd.) nebo k jiným datovým zdrojům. Vytváří nezávislé standartní rozhraní pro řízení systému báze dat. Programátor nemusí znát specifické API databáze, postačí pouze znát jednotné rozhraní JDBC, které využije pro přístup do databáze. Důkazem univerzálnosti je možnost přístupu do jiných než relačních systémů (např. soubory CSF, soubory ve „sloupcové podobě“).

1.2.1 JDBC architektura

JDBC API [3] obsahuje dva modely, dvouúrovňový a tříúrovňový.

Prvním je **dvouúrovňový model.** V tomto případě JDBC ovladač přistupuje přímo ke zdroji databáze. Výstupní data databáze jsou odeslána na základě příkazů uživatele. Zdroj dat nemusí být přímo uživatelův počítač, lze využít například centrální server, ke kterému má uživatel přístup prostřednictvím sítě.

Oproti předchozímu modelu **tříúrovňový model** využívá „mezi vrstvu“, která posílá příkazy od uživatele databázi. Zavedením optimálního překladače pro kompilaci Java *bytecode* se rapidně zvýšil výkon konkrétních technologií např. Enterprise JavaBeans™. Výhodou je jednoduchá správa aktualizací a výkonnější přístup k databázi dat.

1.2.2 Ovladače JDBC

Každý výrobce [1][4] odlišně implementuje rozhraní JDBC pro přístup ke své databázi. Ovladač umožňuje aplikacím komunikovat s databázemi různých výrobců. Této vlastnosti

dosáhne na základě implementace standartní Javy interface pomocí protokolů a API konkrétního systému databáze. Ovladače jsou tvořeny implementací rozhraní `Driver`, balíčku `java.sql` a dalších pomocných tříd. Obvykle je nalezneme v podobě souboru archivu jar. Rozlišujeme základní čtyři typy ovladačů:

První typ **Brigde JDBC-ODBC**[1], využívá operační systém Windows. Tento ovladač se chová jako „most“ mezi JDBC a ODBC. Je součástí SDK. Nevýhodou je nutnost instalace ODBC knihovny na každý klientský počítač. Nejčastěji je tedy využíván v podnikové síti nebo na serverové aplikaci.

Druhým typem je **JDBC API**[2]. Vyžaduje přítomnost nativního ovladače k databázi na klientově počítači. Na tento typ JDBC překládá požadavky od klienta. Jeho nevýhody s instalací jsou stejné jako u předešlého typu. Mají ovšem tu výhodu, že jsou předem kompilovány na danou databázi a mají tudíž větší výkon.

Třetí verzí je implementace databázového ovladače **JDBC Network-Protocol Driver**[3], který využívá střední vrstvu. Převádí JDBC volání na protokol střední vrstvy (aplikační server), který je následně na straně serveru přeložen na síťový protokol konkrétního databázového systému. Díky střední vrstvě už nemusí klient instalovat konkrétní typ ovladače. Záleží pouze na typu databází, které střední vrstva podporuje. K databázi poté mohou přistupovat dva různí uživatelé s rozdílnými ovladači. Síťový protokol je platformě nezávislý, proto není nutné využití Javy. Podporuje také přístup firewallu.

Poslední implementací je **Database-Protocol Driver (Pure Java Driver)**[4]. Převádí JDBC volání přímo do protokolu specifikovaného výrobcem databáze. Ovladač je čistě programovaný v Javě. Není nutný převod požadavků do ODBC. Neobsahuje střední vrstvu, je přímo připojen k serveru databáze, ale není optimalizován pro jeho konkrétní OS.

1.2.3 Rozhraní JDBC

JDBC API [1] obsahuje tří základní vlastnosti:

- Služby pro spojení s databází.

[1] Dokumentace Brigde JDBC-ODBC: http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/bridge.html
[2] Dokumentace JDBC API: http://docs.oracle.com/cd/E19509-01/820-5069/ggzci/index.html
[3] JDBC Network-Protocol Driver: http://docs.oracle.com/cd/E19509-01/820-5069/ggzbn/index.html
[4] Dokumentace Pure Java Driver: http://docs.oracle.com/cd/E19509-01/820-5069/ggzbd/index.html

- Správa SQL příkazů.
- Zpracování výsledků.

1.2.4 Služby pro spojení s databází

Základem spojení [1][5] je registrace ovladače. JDBC umožňuje mnoho způsobů. Velice jednoduchým způsobem je využití třídy `Class` a její metody `forName`*(třída ovladače)*. Třídu ovladače určuje dodavatel databáze. Tento způsob statické inicializace nemusí fungovat na všech JDBC ovladačích. Univerzálním řešením je tedy v konstruktoru vytvořit instanci ovladače metodou `newInstance()`. Verze Javy 1.6 obsahuje mechanismus registrace JDBC ovladače. Je nutné správně nastavit metadata a vše se provede automaticky. Příklad připojení k databázi Oracle a mySQL:

```
try {
Class.forName("oracle.jdbc.driver.OracleDriver");
Class.forName("org.qjt.mm.mysql.Driver");
} catch(ClassNotFoundExceptione) {
System.err.println(" Driver error : "+e.getMessage());
}
```

Po úspěšném registrování ovladače je nutné vytvořit napojení na zdroj dat. Je nutné definovat URL databázového serveru. Parametry URL specifikuje dodavatel databáze. Obvykle je adresa složena z řetězce „jdbc:“, protokolu a DSN (*Data Source Name*). Příklad formátu URL při připojení do Oracle databáze:

```
jdbc:oracle:thin:@<host>:<port>:<SID>
```

Pro přístup ke zdroji dat (DBMS, souborový systém, nebo jiné odpovídající JDBC ovladači) je nutné navázat spojení. Třídy pracují mezi aplikační vrstvou a ovladačem JDBC. Aplikační programátor se tudíž nemusí starat o registraci ovladače, spojení je po zavolání metody `forName()` třídy `Class` navázáno a ovladač je registrován. Existují dvě třídy, které požadované spojení dokáží navázat:

- `DriverManager`

- DataSource

Třída **DriverManager** využívá ke spojení URL adresu, kterou vysvětlila předchozí podkapitola. Třída musí při navázání spojení registrovat nejenom nejnovější JDBC ovladač, ale i všechny předešlé verze.

Druhou možností je třída **DataSource,** která je preferovaná při potřebě objemné aplikace na internetu. Navázání nebo ukončení spojení vyžaduje masivní režii na systém databáze. Tato třída využívá systém „sdruženého připojení“, který umožňuje připojení k systému bez nutnosti vytvoření spojení.

Následující příklad představí jednoduché připojení k databázi pomocí třídy `DriverManger`. K navázání spojení je využita metoda `getConnection()` s parametry: URL databázového systému, vlastnosti uživatele (uživatelské jméno, heslo). Třída registruje vhodný ovladač a vrací rozhraní `Connection`. Příklad připojení k databázi:

```
String user ="Michal";
String password="heslo";
      Connectionconnection =
      DriverManager.getConnection(URL, user,password);
```

V případě, kdy třída nenalezne vhodný ovladač je vyvolána výjimka `SQL-Exception`. Když už není vytvořené spojení potřeba, je nutné jej uzavřít:

```
If(connection != null){
      Connection.close();
}
```

Další fází je správa SQL příkazů. Je využita třída `Statement`, která zpracuje SQL dotazy a vrátí odpovídající hodnoty. Vytvořením instance této třídy je získán přístup k metodě `executeQuery()` sloužící pro SQL příkaz `SELECT`. Modifikace databáze je docílena voláním metody `executeUpdate()`, která je využita při volání příkazů `UDATE`,

INSERT, *DELETE*. Obě tyto metody vracejí objekty třídy `ResultSet`. Třída `Statement` obsahuje velkou škálu metod, více informací lze zjistit z dokumentace. Následuje ukázka využití třídy `Statement`:

```
Statement statement = connection.createStatement();
String sqlSelect = "SELECT * FROM moje_tabulka";
ResultSet resultSetS = statement.executeQuery(sqlSelect);
String sqlUpdate = "DELETE FROM moje_tabulka";
statement.executeUpdate(sqlUpdate);
```

Objekt třídy `ResultSet` obsahuje selektovaná data z databáze ve formátu tabulkové reprezentace. Nejjednodušší způsob, jak tato data získat, je čtení dat z tabulky po řádcích. Třída udržuje kurzor označující aktuální řádek tabulky. Pohybu vpřed mezi řádky je docíleno metodou `next()`, na předešlý metodou `previous()`. Metodou `getXXX()`, kde XXX specifikuje typ proměnné v Javě, je získána hodnota záznamu z tabulky. Parametrem je index sloupce nebo jeho název. Na příkladu je ukázka základní práce s třídou `ResultSet`:

```
while(resultSet.next()) {
results.getString(1) ;
results.getString(„jmeno_sloupce“) ;
results.getInt(3);
}
```

1.3 Objektově relační mapování

Java je [6][7] objektově orientovaný programovací jazyk, využívaný dnes převážně k implementaci enterprise aplikací. Entita je tedy reprezentována jako objekt třídy. Oproti tomu entita v relační databázi je reprezentována množinou řádku v databázových tabulkách. Problém kombinace těchto technologii nazýváme tzv. „Impedance mismatch“[5]. Vznikla tedy technika převodu (mapování) z objektů jazyku Java do relační databáze.

[5] Podrobný popis problému: http://impedancemismatch.com/

Následující podkapitoly pojednávají o mapování objektů do relační databáze, vysvětlíme si základy vztahů mapování objektů na tabulky, jejich vztahy v databázi. Následně se budeme soustředit na základní ORM Framework Hibernate, Java API Persistence a JDO.

1.3.1 Úvod

Úlohou programátora [6][7] je vybrat vhodný ORM Framework. Tyto produkty obsahují Java API, které umožňují uložení objektů do databáze pomocí definovaných metod. Základem je vytvořit spojení mezi objekty a tabulkou databáze při zachování persistence (zachovává vždy předchozí verzi sama sebe, při změně je vytvořena nová struktura dat). Pokročilejší funkce ORM řeší převod dědičnosti objektů do relační databáze. Výhodou implementace systému ORM pro aplikačního programátora je volnost ve výběru systému databáze a to díky odstínění práce s SQL dotazy. Nevýhodou může být komplikovaná konverze převodu objektů do relační databáze. K objektově orientovanému programování bezpochyby patří využití dědičnosti. Je pro nutné využití složité techniky při převodu hierarchie tříd do relační databáze.

1.3.2 Mapování objektů do databáze

Tento pododstavec [7] představí základy matování objektů do databáze. Pro více informací je tu webová stránka[6] s podrobnějším popisem všech druhů mapování.

Základem ORM je převod a persistentní uchování entity (objektu) z Java aplikace do entity relační databáze, která je reprezentována řádkem v tabulce (tabulkách). Jednoduchý příklad entity uživatel:

```
Třída v Javě                        Tabulka relační databáze

Public class Uzivatele{             Table UZIVATELE(
     private int ID                 ID integer primary key,
     private String jmeno           jmeno varchar(150),
     private String prijmeni        prijmeni varchar(150)
     }                              )
```

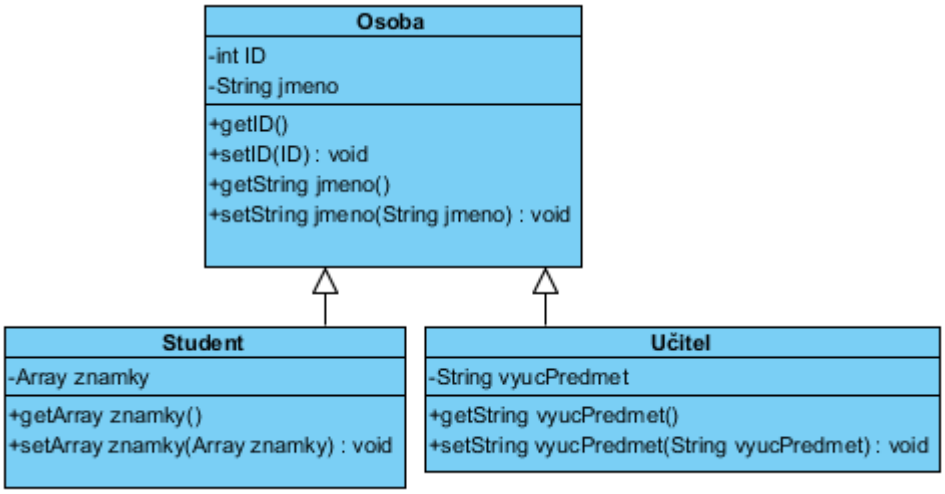
[6] http://www.agiledata.org/essays/mappingObjects.html

1.3.3 Strukturované třídy (dědičnost)

Relační databáze [7] nepodporují dědičnost. To by ale nemělo vývojáře odradit od používání dědičnosti při programování své aplikace. Existují tři techniky pro mapování dědičnosti do relační databáze:

- Celá hierarchie tříd do jedné tabulky.
- Každá celá třída do své tabulky.
- Každá třída do své tabulky.

Úvod odstavce představí jednoduchou strukturu třídy, na které bude postupně rozebráno mapování výše uvedených technik. Na obrázku 1 je znázorněna rodičovská třída `Osoba` obsahující dva základní parametry `ID` osoby typu `Integer` a `Jmeno` osoby typu `String`. Z třídy osoba dědí dvě třídy. Prvním potomkem je třída `Student` obsahující parametr známky typu `Array`. Druhý potomek třída `Učitel` obsahuje parametr vyučovaný předmět typu `String`.



Obrázek 1 - Třída Osoba a její potomci. Zdroj: autor.

První technika mapuje **celé hierarchie tříd do jedné tabulky**. Atributy jednotlivých tříd jsou vloženy do jediné tabulky, nejlépe s názvem rodičovské třídy pro lepší přehlednost. Dobré je do tabulky ukládat hodnotu, ze které je jednoduše určena třída uloženého objektu. Pro tuto potřebu je vytvořen atribut `TypOsoby`, do kterého je ukládáno počáteční písmeno třídy (např. pro třídu `Student`, písmeno S). Výhody techniky jsou více než zřejmé. Není nutné využití SQL příkazů `join`, při změně hierarchie není potřeba měnit strukturu

databáze. Nevýhodou je nevyužití většiny atributů, které nabývají hodnoty `NULL`, plýtváme tak místem. Není možné využít integritní omezení `NOT NULL`.

Při využití druhé techniky, **každá celá třída do své tabulky**, je pro každou tabulku vytvořena jedna tabulka v databázi a to i pro abstraktní třídy. Tabulka dané třídy obsahuje nejenom její vlastní atributy, ale i atributy rodičovské třídy. Výsledná tabulka `Student` obsahuje vlastní atribut `známky`, ale i atribut `ID` a `jmeno` z rodičovské tabulky `Osoba`. Přístup k objektu je jednoduchý, stačí přistupovat ke konkrétní tabulce. Není nutné měnit strukturu DB při „refaktoringu“. Problém nastává v případě přístupu k několika objektům rodičovské třídy, kde je nutné přistupovat k více tabulkám. Je doporučeno při generování umělých klíčů využívat sekvenci pro celé schéma tabulek, aby nedošlo ke shodě identifikačních atributů. V případě, kdy odkazujeme do cizí tabulky na objekt typu `Osoba`, musíme přidat atribut identifikující třídu potomka.

Poslední technika mapování, **každé třídy do své tabulky**, nejlépe odpovídá struktuře tříd z obrázku 1. Tabulka každé třídy obsahuje pouze své vlastní atributy. Zde je nutné dodržovat pravidlo, kdy je identifikační atribut potomka cizím a zároveň primárním klíčem rodiče. Pokud vytvoříme objekt typu `Student`, vytvoříme záznam v obou tabulkách (tabulce `Osoba` i `Student`). Výhodou je využití celé kapacity tabulky, neplýtvá se tak místem. Nevýhodou je změna struktury databáze při „refaktoringu“. Pro přístup k potomkům je využito spojení několika tabulek, které při velké hierarchii zpomaluje rychlost aplikace.

1.3.4 Vztahy entit

V jazyce Java vyjádříme vztah mezi objekty předáním reference. Tyto vztahy je nutné převést do naší relační databáze. Využitím cizích klíčů převedeme tyto vztahy do relační databáze.

1.3.5 ORM frameworky

Na jednoduchém příkladu třídy `Osoba` bude představena základní funkce ORM frameworků:

```
package org.hibernate.tutorial.domain;
public class Osoba{
    private Long id;
    private String celeJmeno;
      public Osoba () {}
      public Long getId() {return id;}
      private void setId(Long id) {  this.id = id; }
      public String getCeleJmeno () {  return celeJmeno;}
public void setTitle(String title) { this. celeJmeno =
celeJmeno;}
}
```

Třída `Osoba` obsahuje bezparametrický konstruktor, který je zároveň jedinou povinnou metodou. Dále jsou tu dva atributy, `Id` typu `Long` a `celeJmeno` typu `String`. Je doporučeno, aby třída poskytovala jeden atribut typu `Long` pro uchování hodnoty primárního klíče.

1.3.6 Java JPA

Java Persistence API [8][9] je standartní persistentní API vyvinutá jako součást a zjednodušení *EJB* entity beans[7], určená jak pro Java EE 5, tak i pro Java SE. Použití není omezeno pouze na EJB. Implementují jej populární frameworky (např. Hibernate, TopLink). Frameworky implementují tento standart spolu s vlastní upravenou verzí implementace JPA. Samotné standartní JPA popisuje rozhraní metod a chování knihoven, sama o sobě není schopná persistence. Pro mapování objektů lze využít dvou metod: pomocí anotací tříd nebo využití XML pro definici tříd.

Java Persistence je složena ze třech základních složek:

- Java Persistence API
- JPQL (Java Persistence Query Language)
- Objektově relační mapování metadat

1.3.7 Mapování objektů

[7] Více informací na: http://docs.jboss.org/ejb3/docs/tutorial/1.0.7/html/EJB3_Entities.html

Základem JPA [9][12] jsou entity, které jsou mapovány do relační databáze. Třídy je nutné před mapování upravit sadou anotací. Pro příklad je upravena třída `Osoba` z úvodu odstavce. Výsledkem mapovaní je následující třída (byla použita metoda anotace třídy):

```
@Entity(name = "Osoba") //Název entity
Public class Osoba{
@Id //definice identifikátoru
@Column(name = "ID", nullable = false)
@GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
@Column(name = "Cele_Jmeno", nullable = false,length = 250)
    private String celeJmeno;
       // následuje definice metod get a set
}
```

Takto upravené třídy popisují způsob mapování pro technologie splňující standart JPA (např Hibernate). Sloupce tabulky lze spojit s atributy (framework využívá Java Reflection) nebo skrze metody `get()`, `set()` (JavaBeans přístup). Nejčastěji využívané anotace:

- `@Entity`: Objekt, který je mapován.
- `@Id`: Označení primárního klíče.
- `@Transient`: Vytvořený objekt není spojen s databází.
- `@Emeddable`: Pro objekty třídy se nevytváří nová tabulka, připojí se k existující tabulce.

Objekty takto upravené třídy je možné persistentně uložit do relační databáze. Pro tuto činnost je nutné úspěšné vytvoření spojení s databází. Následující příklad demonstruje vytvoření spojení a blok transakce. Uvnitř bloku jsou provedeny veškeré operace s objekty:

```
try {
 EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory(PROPERTIES);
    EntityManager em =
entityManagerFactory.createEntityManager();
    EntityTransaction userTransaction =
em.getTransaction();
    userTransaction.begin(); // nová transakce
     // operace s daty
    userTransaction.commit();
    em.close(); // ukončení Entity manageru
    entityManagerFactory.close(); // konec spojení
} catch (Exception e) {
  utx.rollback();     // došlo-li k chybě, vrať se k
poslední úspěšné transakci
}
```

Základem je třída `EntityManagerFactory`, jejíž instance je vytvořena pomocí třídy `Persistence`, která díky vlastnostem (*PROPERTIES*) serveru vytvoří spojení. Tato instance ovládá všechny objekty třídy `EntityManager`, které jsou spojeny s konkrétními entitami. Nová transakce je vytvořena pomocí metod třídy `EntityTransaction`.

Mezi metody pro vytvoření a uložení transakce je vložen kód pro editaci objektů. **Editace objektu** do databáze je provedena následující příkazem:

```
em.persist(Nova_osoba1);// uložení nového objektu, objekt
spojen s databází
em.merge(Nova_osoba2);     // uložení nového objektu,
objekt není spojen s databází
em.remove(Nova_osoba3);    // smazání objektu z databáze
Osoba osoba4 = em.find(Osoba.class, ID); // načtě objekt
z databáze, podle zadeného ID a identifikátoru třídy
```

V případě **úpravy atributu objektu**, pokud jsme využili metodu *`persist()`*, nemusíme po úpravě volat žádnou metodu, změna bude uložena po úspěšném ukončení transakce. Pokud uložíme objekt metodou *`merge()`*, po úpravě atributu objektu je nutné volat tuto metodu znovu.

1.3.8 Závěrečné hodnocení JPA

Tuto API dnes implementuje mnoho ORM Frameworků. Hlavním důvodem je zjednodušení práce s persistencí. Dalším důvodem může být podpora a další vývoj této API. Nejnovější verze JPA 2.1 byla představena 22. května 2013. Tuto novou technologii implementuje i Framework Hibernate, kterému bude věnována pozornost v následující podkapitole.

1.3.9 Hibernate

Flexibilní, výkonný [8][10][15] a možná nejrozšířenější ORM Framework, který je k dispozici zdarma spolu s přehlednou dokumentací[8]. Obsahuje mapovací soubory, které zajistí správné převedení objektu Java aplikace do tabulek relační databáze. Obsahuje speciální jazyk HQL podobný jazyku SQL. Hibernate poskytuje transparentní persistenci, není nutné explicitně mapovat objekty do databáze pomocí příkazů SQL.

O samotné mapování mezi aplikací a databází se starají XML soubory, od verze Javy 1.5 je podporováno mapování pomocí anotace tříd představené v podkapitole JPA. Využívá také vlastní cache, takže v případě kdy uživatel zadává stejné požadavky na server databáze, hibernate načítá data z cache, což ušetří mnoho času a celý systém je tak efektivnější. V případech, kdy dva uživatelé získají z databáze stejné objekty, a jeden z uživatelů tento objekt modifikuje a následně uloží do databáze, druhému uživateli je možnost uložení jeho objektu zakázána (nemá aktuální objekt), aby nedošlo k nekonzistenci dat.

1.3.10 Mapování objektů

Na začátku [10][12][15] mapovacího souboru XML je nutné nastavit načítání a ukládání dat do databáze. Většina vývojových IDE umožňuje tuto hlavičku s nastavením generovat automaticky. Příklad mapování třídy (Obrázek 1) `Osoba`:

```
<hibernate-mapping package="org.hibernate">
    <class name="Osoba" table="OSOBA">
        <id name="id" column="OSOBA_ID">
            <generator class="native"/>
        </id>
        <property name="celeJmeno" type="String"
column="CELE_JMENO"/>
    </class>
</hibernate-mapping>
```

[8] na stránkách http://hibernate.org .

Všechny parametry jsou zadávány do příslušných tagů podle XML syntaxe. První tagem je vždy *hibernate-maping*. Dalším je tag *class* s parametry: název třídy objektu a název tabulky v databázi. Mezi tento tag je uveden atributy třídy. Na začátek je nutné uvést důležitý tag id, který slouží jako primární klíč objektu, s parametry: název atributu objektu a název sloupce tabulky. Je možné zvolit způsob automatického generování Id v tagu *generator*. Dále pak tag *property* nastaví zbylé atributy. Hibernate automaticky z parametru name jednotlivých tagů určí použití správných metod *get()* a *set()* jednotlivých atributů.

„Hibernate potřebuje pro správnou funkci několik nastavení, jako je JDBC driver, adresa databáze (connection string), generátor SQL dotazů (dialect) nebo cesty k mapovacím souborům. Tento konfigurační soubor se běžně pojmenovává hibernate.cfg.xml“[13].

1.3.11 Vytvoření spojení

Před vytvořením [8][10][15] spojení odstavec popíše, v jakých stavech se může objekt nacházet, pokud je připojen do databáze. Pro vytvoření spojení s relační databází je implementována třída Session. Hibernate nabízí tří druhy objektů, které ukládá a mapuje do databáze.

- Transient: Vytvořený objekt není spojen s databází, ani s vytvořenou „Session“.
- Persistent: Vytvořený objekt je spojen s databází i se „Session“. Při změně hodnoty objektu v aplikaci bude tento stav uložen i v databázi.
- Detached: Tento objekt se nachází v databázi, ale není spojen se „Session“. Jeho případná změna v aplikaci nebude uložena do databáze.

Navázání spojení s databází zajistí objekt třídy *org.hibernate.SessionFactory*, který řídí veškerou komunikaci. Třída vyžaduje informace z konfiguračního souboru *hibernate.cfg.xml*, probraný v předešlé podkapitole. K vytvoření „Session“ je využita třída *Configuration* a její metoda *configure()*. Příklad ukáže vytvoření a následné uzavření spojení:

```
try{
 // vytvoření spojení
 SessionFactory factory = new
Configuration().configure().buildSessionFactory();
// metoda pro ukončení spojení
session.close();
 }catch (Throwable ex) {
 System.err.println("Failed to create sessionFactory
object." + ex);
 }
```

Ve stavu nově navázeného spojení je možné začít objekty ukládat, upravovat a mazat z databáze. Příkladem v tomto odstavci je třída **Osoba** z podkapitoly 3.4.1 mapování objektů. Základní struktura metody:

```
Session session = factory.openSession();
// otevření spojení
 Transaction tx = null;
 try{
      tx = session.beginTransaction();
// začátek transakce, ulož vytvořené objekty
//tělo operací (ulož, uprav, smaž)
tx.commit(); // ulož transakci
 }catch (HibernateException e) {
}finally {
 session.close(); }
 // v případě nečekané chyby zavři spojení
```

Uložení objektu do databáze:

```
Osoba michal = new Osoba();
Osoba.setCeleJmeno("Michal_Barnet");
Int osobaID = (Integer) session.save(michal);
```

K uložení instance třídy `Osoba` je volána metoda `save()`. Parametrem metody je nově vytvořený objekt `michal`. Metoda vrací automaticky vygenerovaný primární klíč objektu v tabulce databáze.

Změna atributu objektu:

```
Osoba michal = (Osoba)session.get(Osoba.class, IDOsoba);
Osoba.setCeleJmeno("Pepa_Novák");
      session.update(michal);
```

Metoda `get()`, s parametry: třída objektu, primární klíč objektu, vrací instanci třídy objektu z databáze. Pomocí zadaných parametrů metoda vybere řádek tabulky z databáze. Atributy instance `michal` jsou následně upraveny a metoda `update()` uloží upravený objekt do databáze.

Výpis tabulky:

```
List osoby = session.createQuery("FROM OSOBA").list();
 for (Iterator iterator =
 osoby.iterator(); iterator.hasNext();){
 Osoba osoba = (Osoba) iterator.next();
 System.out.print("Jméno osoby : " +
osoba.getCeleJmeno());
}
```

Načtení všech objektů z konkrétní tabulky, zprostředkovává metoda `createQuery()`. Tato metoda umožňuje výpis objektů do různých kontejnerů (v ukázkovém příkladu se jedná o kontejner typu `List`).

Mazání objektů:

```
Osoba michal = (Osoba)session.get(Osoba.class, IDOsoba);
session.delete(michal);
```

Pro mazání objektu je nutné nejprve daný objekt z databáze načíst. Poté jej předáme jako parametr metodě `delete()`.

1.3.12 Závěrečné hodnocení

Hibernate je bezpochyby velice účinný, jednoduchý ORM Framework pro práci s objekty. Programátor nemusí znát specifické příkazy pro konkrétní databázi, ty přenechá perzistentní vrstvě. Stačí pouze využívat jednoduché API pro přístup a modifikaci objektů v databázi.

1.4 JDO

Java Data Objekt (JDO) [11] specifikuje standartní metody pro přístup k persistentním datům v databázi pomocí POJO[9]. Na rozdíl od předchozích probraných ORM, které striktně mapovaly objekty do relační databáze, JDO umožňuje volnější výběr zdroje dat (např. objektová databáze, soubory). Ovšem relační databáze jako zdroj dat nadále převládá, proto následující podkapitola probere pouze převod na tento systém databáze. JDO poskytuje základní rozhraní pro práci s databází, především pak se základními příkazy (načti, ulož, smaž, edituj). Aby technologie JDO dosáhla persistence dat, využívá technologii XML. Systém je vyvinut firmou Sun Microsystem. Programátor je tu stejně jako u systému hibernate oproštěn od znalosti relační databáze, vystačí si pouze se znalostmi technologie JDO.

1.4.1 Mapování objektů

JDO [11] využívá k mapování třídy objektů XML soubory. Každá třída obvykle má jednu tabulku nesoucí jméno třídy. Příklad mapování pomocí jedné z implementací JDO a to technologii JPOX[10]:

[9] POJO třídy se vyznačuje jednoduchostí, neimplementují žádné rozhraní. Obecné požadavky na POJO třídy jsou následující: bezparametrický konstruktor, metody *get()* a *set ()*. Více informací na stránce: http://msdn.microsoft.com/en-us/library/cc681329.aspx

[10] JPOX je implementací produktu JDO. Má za úkol zautomatizovat monotónní metody pro persistenci dat, které využívá JDO. Více informací na stránce: http://www.ibm.com/developerworks/data/library/techarticle/dm-0506bhogal/

```
    <class name="osoba" identity-type="datastore"
table="OSOBA" >
        <extension vendor-name="jpox"
                   key="table-name" value="Michal"/>
        <field name="id" primary-key="true" value-
strategy="identity"/>
        <field name="jmeno">
            <extension vendor-name="jpox" key="column-
name" value="NAME"/>
            <extension vendor-name="jpox" key="length"
value="max 50"/>
        </field>
    </class>
```

Specifikace jména třídy, název tabulky a typ třídy je uveden v tagu `class`. Tag `extension` určí implementaci JDO, v našem případě JPOX. Je možmé poté například specifikovat název sloupce tabulky, maximální velikost hodnoty atd. Pokud jde o vnořený objekt třídy, který je vkládán do stejné tabulky, je nutné do tagu `field` uvést atribut `embedded` nastavený na hodnotu `true`. JDO implicitně rozpozná základní datové typy Java aplikace. Ostatní typy jsou do tabulky převedeny jako pole bajtů. Relační databáze disponuje typem `BLOB`, který je schopen uložit dané pole bajtů.

1.4.2 Vytvoření spojení

Spojení [11] zprostředkovává třída `PersistenceManagerFactory`. Instanci této třídy je vytvořena pomocí metody `getPersistenceManagerFactory()` třídy `JDOHelper`, která vytváří spojení s relační databází. Dále je nutné specifikovat vlastnosti databáze (adresa, heslo, uživatelské jméno atd.). Příklad spojení s databází:

```
PersistenceManagerFactory pmfactory =
    JDOHelper.getPersistenceManagerFactory("properties");
PersistenceManager pm =
pmfactory.getPersistenceManager();
```

V metodě `getPersistenceManagerFactory()` jsou nastaveny vlastnosti databázového severu (heslo, uživatelské jméno, URL atd.). Ukázka souboru properties:

```
javax.jdo.option.ConnectionDriverName=org.h2.Driver
javax.jdo.option.ConnectionURL=jdbc:loaclhost:tpc
javax.jdo.option.ConnectionUserName=Michal
javax.jdo.option.ConnectionPassword= heslo
```

Instance třídy `PersistenceManager` se stará o persistenci objektů. Manipulace s objekty obstará „Transakce“. Jedná se o posloupnost příkazů splňující tyto vlastnosti:

- Atomicita: Atomické operace nelze přerušit. Pokud je vyvolána výjimka, je transakce vrácena do počátečního stavu tzn.: žádné z provedených změn nebudou uloženy. Například při převodu peněz z banky, kdy jsou peníze odečteny z účtu odesílatele a následně uloženy na účet příjemce, musí proběhnout obě operace ve správné posloupnosti.
- Konzistence: Při a po provedení transakce není porušeno žádné integritní omezení.
- Trvalost: Po úspěšné transakci jsou všechna data uložena v databázi, nemůže dojít k jejich ztrátě.

Ukázka základního použití transakcí na následujícím přikladu, který navazuje na vytvořené spojení v předešlé ukázce:

```
// pm je objekt třídy PersistenceManager
Transaction tx = pm.currentTransaction();
try {
    tx.begin();
    // operace s objekty
    tx.commit();
}
finally
{
    if (tx.isActive())
    {
        tx.rollback();  // V případě chyby zajistíme
konzistenci dat.
    }
}
```

1.4.3 Stavy objektů

Před operacemi [19] s objekty je důležité objasnit stavy, ve kterých se může objekt nacházet v průběhu transakce. Změny stavu vyvolává třída `PersistenceManager`,

přesněji její metody. JDO oproti hibernate obsahuje rozsáhlejší škálu stavů persistentních objektů. Základní trojicí stavů:

- Tranzistent: V tomto stavu se nachází každý nově vytvořený objekt. Není spojen s databází (není uložen jako persistentní). Pro JDO je to nedůležitý objekt, existuje pouze v paměti Java aplikace.
- Persistent: Objekt je uložen jako persistentní, jakékoliv změny atributů objektu v Java aplikaci se projeví do tabulky databáze. Stavy `persistent` se dále dělí na dalších pět podstavů podle vztahu objektu s databází.
- Hollow: Objekt změní stav na `Hollow`, po úspěšném uložení do databáze. Každý právě načtený objekt z databáze je tohoto typu. Se změnou atributů je objekt převeden zpět na typ `persistent`.

Dělení stavu persistent:

- New: Nově vytvořený objekt, který je po úspěšné transakci uložen do databáze.
- Clean: Při načtení objektu typu `hollow` zpět do aplikace. Jedná se o čistý objekt beze změn jeho atributů.
- Dirty: Hodnoty atributů v aplikaci a databázi se nerovnají. Tento stav nastává v případě jakékoliv změny objektu. Změna objektu se projeví až po úspěšné transakci.
- Deleted: Objekt je stále obsažen v databázi. Pouze ale do doby, kdy je úspěšné ukončena transakce, objekt je vymazán a převeden na typ `transient`.
- New Deleted: Objekt typu `new` je před ukončením transakce smazán. Nedojde tedy k jeho uložení do databáze.

1.4.4 Operace s objekty

Odstavec pojednává [19] o základních metodách pro editaci objektů. Příklady navazují na kód v odstavci *4.2 Vytvoření spojení*.

Příklad **uložení objektu** do databáze. Posloupnost stavů objektu při jeho uložení je následující: `Transient`, `Persistent new`, `Hollow`. Ukázka příkazu:

```
pm.makePersistent(new Osoba("Michal"));
tx.commit();
```

Pro možnost editace nebo smazání, je nutné nejprve provést načtení objektu z databáze. Existují dvě základní možnosti:

- Přes **indetifikátor**: podmínkou je znát identifikátor objektu
- Přes **extent**: V případě kdy není známí identifikátor objektu, je nutné načíst všechny objekty z tabulky.

V případech, kdy programátor generuje vlastní identifikátory objektů, je vhodné využít první možnost, metoda `getObjectById` třídy `PersistenceManager`. Metoda obsahuje dva parametry, `ID` objektu a `validate` typu `boolean`. Nastavením druhého parametru na hodnotu `true`, bude kontrolována přítomnost objektu v databázi.

Možností `extent` jsou načteny všechny objekty z tabulky. Následnou iterací příslušné kolekce jsou provedeny jednotlivé operace s objekty. Objekty je možné načíst metodou `getExtent()` třídy `PersistenceManager`. Prvním parametrem je objekt typu `Class`, druhým pak typ `boolean`. Pokud je vyžadováno načtení odvozených objektů dané třídy, je nutné nastavit druhý parametr na hodnotu `true`.

Po úspěšném načtení, lze objekt editovat. Před první úpravou atributů je volána metoda `begin()` pro vytvoření nové transakce. Uložené změny atributu jsou potvrzeny příkazem `commit()` třídy `Transaction`. Stejné pravidlo platí i pro operaci mazání objektů metodou `deletePersistent()` třídy `PersistenceManager`. Metodě je předán mazaný objekt v parametru.

1.4.5 Závěrečné hodnocení

JDO je všestranný, komplexní, univerzální nástroj pro mapování objektů nejenom do relační databáze, ale i do souborů nebo objektových systémů. Nutné je pouze využít příslušnou implementaci od konkrétního výrobce databázi. S využitím JDBC konektorů umožňuje persistenci objektů do všech hlavních datových systémů. Firma Sun vydala implementaci JDO volně na svůj web, existuje proto dnes mnoho komerčních i volně

dostupných implementací. Mezi ty nejzajímavější produkty, hlavně díky jejich open-source implementací, patří: Triactive JDO, JPOX (kvalitní, aktivní vývoj) a Speedo.

2 Problematika návrhu databázových modelů

Návrh databázových modelů lze rozdělit na tři základní fáze:

- Konceptuální: Identifikace podstatných entit (objektů).
- Logický: Převod objektů do tabulek.
- Fyzický: Implementace databázového systému.

2.1 Konceptuální model

V této [13][16][18] fázi návrhu není důležité zvolená fyzická implementace databáze nebo technologické omezení, úkolem je nalézt v modelovaném systému entity (s atributy) a vztahy mezi nimi. Na základě analýzy dat je vytvořen tzv. konceptuální model, který mapuje analyzované objekty a jejich vzájemný vztah. Cílem je obvykle vytvořit E-R diagram (konceptuální model vysoké úrovně), reprezentující prezentaci datových požadavků. Model by měl obsahovat požadavky uživatelů a vytvořit podklad pro pozdější fyzickou tvorbu modelu. Podkapitola představí základní pojmy, které budou hojně využívané v této kapitole: Entita (Objekt), Atribut (vlastnost), Vztah, primární klíč a integritní omezení.

Entita je objekt z reálného světa, musí být schopná nezávislé existence a rozlišitelná od ostatních objektů. Pro příklad je uveden reálný objekt: *Adresa*. Každý objekt obsahuje atributy (vlastnosti objektu), v případě *Adresy* např. Město, ulice, číslo popisné atd. *Adresa* může být obsažena v entitě osoba, existuje tedy mezi nimi vztah (každá osoba má např. adresu bydliště).

Z návrhu uživatele je někdy těžké určit správnou podobu entit. Dobrým vodítkem je držet se pravidla, kdy podstatná jména volíme jako entity a slovesa jako vzájemný vztah. Toto pravidlo neplátí vždy, zvláště v případech, kdy je podstatným jménem atribut (např. jméno, příjmení, barva, telefonní číslo). Hlavním rozdílem je, že atribut není schopen samostatné existence.

Integritní omezení popisuje pravidla a omezení pro atributy, entity a vazby, tak aby model odpovídal co nejvíce skutečnosti. Lze je vyjádřit pomocí grafického zpracování (ERD model) nebo textovou formou. Pro lepší pochopení je uvedeno několik příkladů. Zákazník videopůjčovny si může vypůjčit více filmů, ale konkrétní film lze vypůjčit pouze jednou

jedné osobě. Primární klíč je atribut, který jednoznačně identifikuje konkrétní entitu. Entita může obsahovat několik kandidátních klíčů, ale pouze jeden primární klíč.

Jednotlivé úkoly vytvořeného E-R modelu jsou následující:

- **Identifikace entit**: Správné analyzování a identifikace hlavních entit daného zadání.
- **Identifikace relací**: Analyzovat a charakterizovat spojitost mezi entitami z předešlého kroku. Je vyznačena kardinalita vztahu. Dále je nutné prověřit redundanci vztahů.
- **Identifikace kandidátních klíčů**: Identifikace atributů (jednoduché nebo spojené) jednoznačně určí entitu. Tyto atributy jsou označeny jako kandidátní klíče. Z této skupiny je zvolen primární klíč, který nejlépe odpovídá identifikaci entity. V některých entitách je primárné klíč složen z cizího klíče jiné entity.
- **Identifikace atributů spojené s entitami**: V této fázi je doplněna entita zbývajícími atributy, které jí charakterizují.
- **Kontrola redundance v modelu**: Kontrola E-R modelu, zejména je třeba zkontrolovat relace typu 1:1 a případně odstranit redundantní relace. Redundance nastává v případech, kdy model obsahuje dvě entity, které mají sice jiný název, ale jsou typově stejné. Například notebook a laptop. Nejjednodušším způsobem opravy je spojení nalezených entit do jedné.
- **Kontrola, zda model podporuje uživatelské transakce**: Po provedení předchozích úkolů je k dispozici úplný E-R model. Zbývá provést test, zda model podporuje všechny potřebné operace, obsahuje důležité entity a atributy. Existují dva typy ověření správnosti: Metoda popisu transakcí (kontrola zda pro konkrétní transakci jsou k dispozici všechny entity a atributy) a metoda sledování cest transakcí (pro konkrétní operaci sledujeme vztahy entit, přesněji dostupnost všech entit potřebných pro operaci).

Konceptuální model lze vyjádřit více způsoby: Lineární zápis a E-R model. Lze využít možnosti UML Class Diagramu, v případech použití objektově orientované databáze. Výhodou je implementace mnoha vývojových nástrojů včetně automatického generování. Dnešní doba využívá převážně relační databázi tudíž je tato možnost brána jako okrajová.

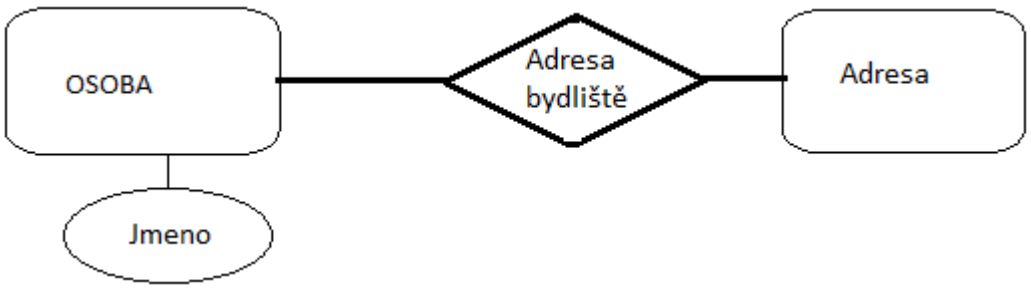
2.1.1 Lineární zápis

Používá textový zápis popisující entity a vztahy.

```
VZTAH (ENTITA_1, ENTITA_2, vztahovy_atribut_1,vztahovy_atribut_2)
```

2.1.2 E-R model

Druhá varianta je častěji využívaná, hlavně pro relační databáze. Znázorňuje entity, atributy a vztahy přehlednější grafickou formou. Dále obsahuje primární klíč (identifikátor entity) a integritní omezení. Nenaznačuje operace s objekty, pouze množinu entit a jejich vzájemné vztahy. Typický návrhem modelu je tzv. „návrh od shora dolů“(začneme postupně od nejdůležitější entity směrem k méně důležitým). Jak už bylo uvedeno v předchozím odstavci, výstup modelu by měl být jednoduchý, srozumitelný. Bohužel jeho notace nejsou jednotné, existuje mnoho variant zápisu. Jako příklad je zde uveden základní model na obrázku 2 obsahující tří základní grafické tvary: obdélník pro entitu, ovál pro atributy a kosočtverec určující vztah entit.



Obrázek 2 - E-R model. Zdroj: autor.

2.2 Logický model

Druhá fáze [13][16][18] návrhu je tzv. mezičlánek, který zpracovává data z E-R modelu a připravuje převod na fyzický model. Vše je popisováno v obecné rovině (bez znalosti konkrétního systému databáze), podobně jako v první fázi. Tento mezičlánek je důležitý v případech, kdy výsledná aplikace bude vytvářena pro různé databázové platformy. Není tak nutné

v případě změny upravovat modely všech verzí, ale pouze jeden. Funkce převodu bude dále vysvětlena na relačních databázích.

Logický návrh se dělí na tvorbu tabulek a kontrolu. Relační databáze je tvořena z tabulek obsahující sloupce, řádky jsou identifikovatelné pomocí primárního klíče.

2.2.1 Tvorba tabulek

Hlavní funkcí [13][16][18] této fáze je vytvoření tabulek relační databáze z konceptuálního modelu. Pomocí jednoduchých pravidel lze proces téměř zautomatizovat. Entita E-R modelu se stává tabulkou, atributy jsou převedeny na sloupce dané tabulky. Je také možné vytvářet vlastní tabulky, které E-R model neobsahoval. Relace jsou převedeny na primární nebo cizí klíč, podle identifikace (využití kardinality) rodičovské nebo dceřiné tabulky, kde dceřiná tabulka obsahuje primární klíč své rodičovské tabulky, tzv. cizí klíč. Pravidla relací:

Jde-li o vztah **1:n**, je přiřazen cizí klíč do tabulky na stranu n.

Vztah **1:1** je komplikovaný. V případě parciálního vztahu na obou stranách jsou tabulky spojeny, primárním klíčem se stává vhodnější kandidát z obou tabulek. Pokud je naopak vztah neparciální na obou stranách, je určena z dostupných informací rodičovská a dceřiná tabulka. Poslední možností je částečná parcialita, kde na nepovinné straně je tabulka určena jako dceřiná.

Vztah **M:N** není o nic složitější. Vztah je dekomponován na dva typy **1:N**. Na straně **N** je vytvořena nová tabulka, která bude obsahovat pouze atributy daného vztahu. Primární klíč této tabulky vznikne spojením cizích klíčů tabulek, které byly ve vztahu **M:N.**

2.2.2 Kontrola transakcí

Postup je shodný s postupem v konceptuálním modelu, entity jsou pouze zaměněny za tabulky.

2.3 Fyzický model

V předchozích [13][16][18] dvou fázích návrhu nebylo nutné znát implementaci databázového systému, navrhoval se pouze univerzální model. Předchozí kapitola představila logický model, ze kterého fyzický modelu vychází. Tento obecný logický model je převeden na konkrétní implementaci databázového serveru. Je tedy nutné důkladně nastudovat funkce, které systém umožňuje. Příklad základních funkcí:

- Definovat primární a cizí klíče.
- Vytvářet integritní omezení.
- Možnost nastavení vlastní domény.

Pokud programátor nastudoval funkce našeho databázového a logického modelu, je možné začít s převodem na model fyzický. Fáze je rozdělena na dvě podfáze: tvorba fyzických tabulek a tvorba integritního omezení.

2.3.1 Tvorba tabulek

Pro vytvoření [14] tabulky je využita následující informace: jméno tabulky, názvy a definice domény (maximální délka, typ hodnoty) sloupců. Dnešní vývojové softwary pro modelování logického modelu umožňují automatické generovaní skriptů, ze kterých je následné vytvořena struktura tabulek. Tvorba tabulek také závisí na použité implementaci databáze. Pro praktickou ukázku je vybrána databáze Oracle. Vytvoření tabulky, pokud databázový systém podporuje normu ISO SQL:2006, je docíleno příkazem `CREATE TABLE`. V tomto bloku následuje vždy název sloupce a jeho zvolený typ. Používáme základní datové typy z tabulky 1:

Tabulka 1 - Datové typy Oracle. Zdroj [14].

CHAR(size)	Rozsah 1 až 8 000 znaků
VARCHAR2(size)	Znakový řetězec proměnlivé délky s maximální délkou bytů od 1 do 4000.
NUMBER(p, s)	Číslo celé či reálné - s přesností p a měřítkem s..
DATE	Datum a čas mezi 1.1.4712 př. n. l. a 31.12.4712 n. l.

Příklad vytvoření tabulky Student:

```
CREATE TABLE student
    (student _id    NUMBER(6)
    , jmeno          VARCHAR2(100)
        CONSTRAINT jmeno_student NOT NULL
    , prijat      DATE  DEFAULT SYSDATE
        CONSTRAINT prijat_ student NOT NUL
, identifikace_studenta VARCHAR2(20)
    , CONSTRAINT     identifikace_studenta_student
                     UNIQUE (identifikace_studenta)
    ) ;
```

2.3.2 Integritní omezení

Tato fáze slouží pro finální nastavení integritního omezení. Vše je nutné si dobře promyslet, změny omezení v už naplněné tabulce nejsou možné. Příklad nastavení je patrný z předešlého nastavení. Příkazem `Contraint` jsou následně určena integritní omezení. Předchozí příklad používá omezení `NOT NULL` (data ve sloupci nemohou být prázdná) a `UNIQUE` (žádný záznam ve sloupci nesmí být duplicitní).

3 Aplikace zkušebna

Praktická část bakalářské práce je zaměřena na aplikaci, která bude sloužit ke správě dat na oddělení zkušebna. Primárním cílem této aplikace je editace zkoušek, vzorků, měřících zařízení, protokolů vypracovaných na základě dané zkoušky a vzorku. Aplikace je vyvinuta v jazyce Java verze JDK 1.6. Data jsou ukládána na databázový server Oracle. Obsahem této aplikace je skript pro vytvoření modelu databáze na server Oracle. Výpis dat zprostředkovává GUI Java aplikace. Pro tyto základní položky aplikace generuje přehledné sumáře vhodné pro lídra daného oddělení. K dispozici je dále přehledný kalendář mapující naplánované, právě probíhající nebo ukončené zkoušky.

3.1 Funkční požadavky

Předchozí odstavec pojednával o obecných funkcionalitách, které tato podkapitola představí podrobněji. Popis je rozdělen na dvě základní části, databázový systém a Java aplikaci. Relační databázový model by měl být schopný implementace na systém Oracle, ale i na systém MySQL.

Hlavním cílem aplikace je editovat výsledky zkoušky spolu s příslušnými daty. Umožnit tyto zkoušky zadat zaměstnancům. Dále editovat typy zkoušek, testovací vzorky, testovací zařízení. Administrátor by pomocí kalendáře měl mít přehled o průběžném plnění zkoušek, dále také o historii naplánovaných zkoušek. Pro přehlednost by měly být vytvořeny tabulky zobrazující generované sumáře. Pro zjednodušení, zaměstnancům by mělo být umožněno nahrát pouze cestu k protokolu a příslušným datům. Informace z těchto protokolů by měla aplikace sama zpracovat a nahrát na příslušný databázový server.

3.2 Soupis použitých technologii

Aplikace byla rozvržena na dvě základní části. Databázi firmy *Oracle* a GUI Java aplikaci. Popis začíná technologií databázového systému. Model databáze byl vytvořen v programu *Data Modeler 4.0.1* společnosti *Sun Microsystems*. Byl zde vytvořen konceptuální a relační model databáze. Tento program byl vybrán z mnoha důvodů. Umožňuje automatické generování DDL skriptů při převodu z logického na konkrétní fyzický model *Oracle* databáze. Dále díky jeho přehlednosti, možnostem a jednoduchostí je vývojáři využíván, existuje proto spousty přehledných návodů a dokumentací. Firma *Sun* doporučuje tento

program pro vývoj relačních modelů. K aplikování vygenerovaného DDL skriptu a správu databáze byl využit program *Oracle SQL Developer 4.0*. Poskytuje základní operace nad tabulkami databáze, úpravu PL/SQL skriptů, připojení k *Oracle* databázi nebo zobrazení metadat.

Druhá část byla programována v programu *Netbeans IDE 7.4* firmy *Sun Microsystems*. Jedná se o vývojové prostředí napsané v jazyce Java. Primárně je tedy postavené pro vývoj Java aplikací, podporuje ovšem i další programovací jazyky (např. C++, C# , PHP, HTML 5). Netbeans podporující všechny hlavní platformy: J2SE, J2EE. Aplikace byla odladěna na verzi Java 1.6.0. Pro připojení k Oracle databázi byl použit JDBC ovladač (verze 11.2.0). Zpracování protokolů umožňuje knihovna poi-ooxml a poi-3.9 [20].

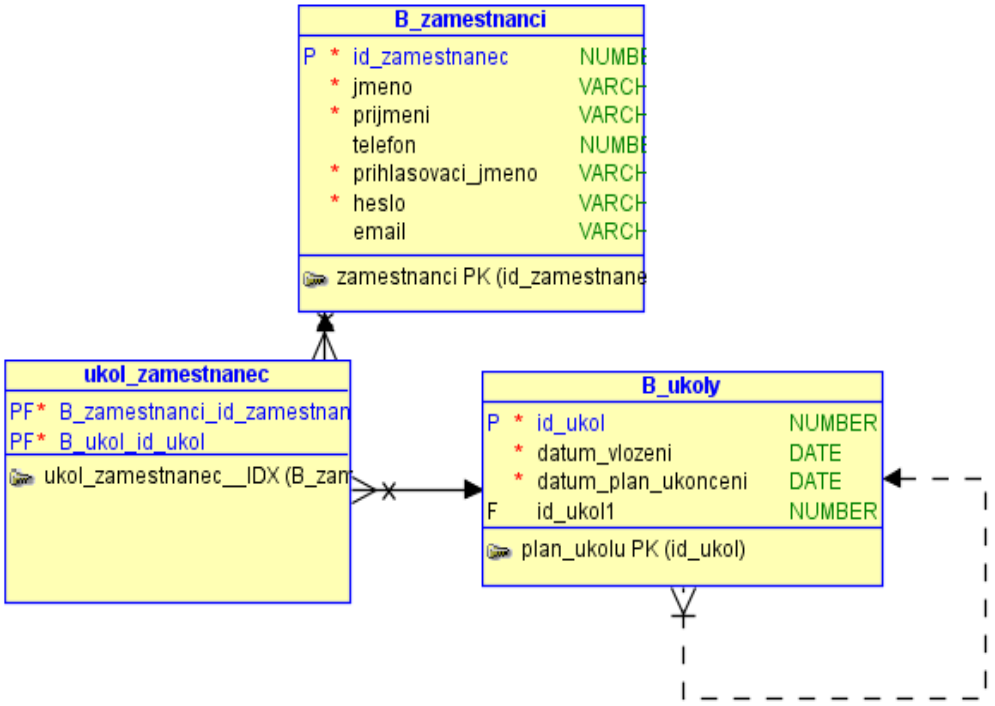
Důvodem výběru těchto vývojových prostředí je jejich vzájemná kompatibilita, pohodlné, profesionální zpracování a *open source* licence.

3.3 Popis databázového modelu

Teoretické základy byly představeny v předchozí kapitole 2: Problematika návrhu databázových modelů. Následuje problematika návrhu modelu v praxi. Postupně je představena podoba tabulek v relačním modelu spolu s jejich vzájemnými vztahy. Relační model obsahuje 19 tabulek. Podrobně je probrána pouze hlavní kostra návrhu, která čítá deset tabulek. Podoba relačního modelu v **příloze A**.

První tabulka `B_Zamestnanci`, sloužící pro uchování uživatelů, kteří mají právo přístupu (ovládání) k aplikaci. Primárním klíčem tabulky je `id_zamestnanec`. Tento umělý klíč je automaticky pomocí triggeru a seqvence generován do tabulky. Integritní omezení jedinečnosti je nastaveno na dva atributy: `prihlasovaci_jmeno` a `email`. Typy atributů nejsou nijak překvapivé.

Předchozí tabulka je relací M:N spojena s tabulkou `B_UKOLY`. Tabulka slouží pro uchování úkolů, které jsou pomocí aplikace zadány zaměstnanci. Umělý primární klíč `id_ukolu` je aplikací generovan pomocí příslušné sequence. Další atributy slouží pro nastavení vytvoření a plánovaného ukončení úkolu. Tabulka mimo jiné obsahuje unární relaci (relace je spojena sama se sebou) pro účely archivace úkolů v případě změny stavu nebo úpravě jejich atributů. Stavy úkolů (zadáno, hotovo, probíhá) jsou nastaveny pomocí cizího klíče tabulky *B_STAV*. Popsané tabulky přehledně na obrázku 3.



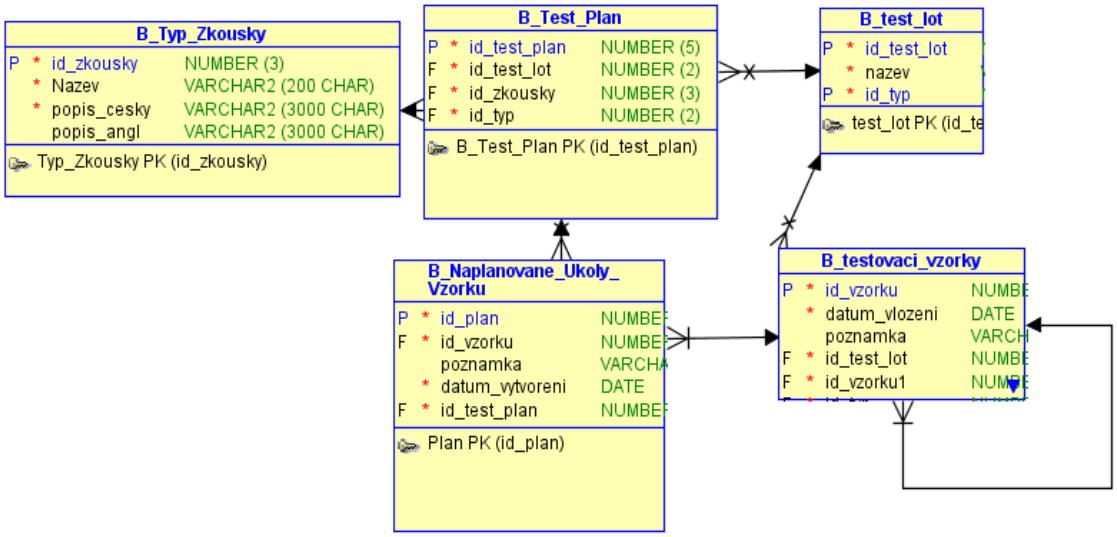
Obrázek 3 - Tabulky databáze Úkol a Zaměstnanci. Zdroj: autor.

Další tabulka `B_Typy_zkousky` slouží pro uchování veškerých typů zkoušek, které jsou na oddělení Zkušebna zkoušeny. Umělý primární klíč je generován automatiky pomocí příslušného triggeru a seqvence. Omezení `Unique` je nastaveno pro atribut označující název zkoušky.

Záznamy z předchozí tabulky jsou obsaženy, pomocí cizích klíčů, v následující tabulce `B_Test_Plan`. Na každé testovací verzi modelu vzorku je nutné provést určité zkoušky. Právě kvůli této vlastnosti aplikace slouží tato tabulka. Tabulka obsahuje dva cizí klíče, prvním je primární klíč tabulky `B_Typy_zkousky`, druhým je primární klíč tabulky `B_Test_Plan` (tabulka uchovává jednotlivé verze modelů). Generování primárního klíče zajistí příslušný trigger a seqvence.

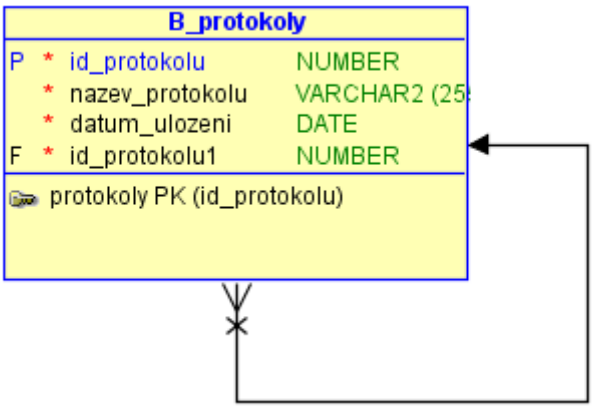
Pro testování jsou nutné vzorky. Záznamy jednotlivých vzorků jsou uloženy v tabulce `B_Testovaci_Vzorky`. Tabulka obsahuje tři cizí klíče tabulek obsahujících možnosti statusů, aktivitu vzorku a verzi vzorku. Dále obsahuje atributy název vzorku, datum uložení do databáze. Pro archivaci vzorku v případě změny jeho atributů, tabulka obsahuje unární relaci.

Hlavní funkci aplikace, testování vzorků podle plánu zkoušek, zajišťuje tabulka `B_Naplanovane_Ukoly_Vzorku`. Do tabulky ukládáme, který vzorek, na jaké zkoušce a s jakým příslušenstvím bude úkol testován. Atributy tabulky tedy tvoří cizí klíče tabulek zmíněných výše, datum zadání a poznámka. Primární klíč je generován pomocí triggeru a sequence. Popsané tabulky přehledně na obrázku 4.



Obrázek 4 - Tabulky databáze naplánované úkoly. Zdroj: autor.

Výsledky testů (protokoly) je nutné zaznamenávat. K tomuto účelu je vytvořena tabulka `B_Protokoly`. Atributy tabulky jsou složeny převážně z cizích klíčů tabulek, ve kterých jsou uloženy materiály využité ke zkoušce (obrázky, video, data). Tabulka pro archivaci změněných záznamů disponuje unární relací. Atributy tabulky přehledně na obrázku 5.



Obrázek 5 - Tabulka Protokoly. Zdroj: autor.

3.4 Připojení k DBMS

Pro připojení k Oracle databázi byl použit JDBC ovladač (verze 11.2.0). Princip připojení pomocí ovladače JDBC byl ukázán v první kapitole. Tento postup využívá i naše aplikace. Vytvořené spojení poskytuje třída `OracleConnection` z balíčku `oracleCon`. Tato třída obsahuje několik základních metod pro správu připojení. Metoda `autorizeConnection()` ověřuje, je-li příslušný ovladač k dispozici:

```
public static boolean autorizeConnection() throws
ClassNotFoundException, SQLException {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            return true;
        } catch (ClassNotFoundException e) {
            throw new ClassNotFoundException("OracleDriver není
dostupný ");
        }
    }
```

Třída dále obsahuje metody `getConnection()` (vrací aktuální spojení), `closeConnection()` (uzavírá aktuální spojení). Nejdůležitější metoda `setConnection()` nastavuje a vytváří spojení s databází. Struktura této metody je následující:

```
public static void setConnection(int port, String sid, String
serverN, String userName, String password)
            throws SQLException, ClassNotFoundException {
      try {
        if (autorizeConnection()) {
            Properties props = new Properties();
            props.put("user", userName);
            props.put("password", password);

            OracleConnection.url = "jdbc:oracle:thin:@"
                    + OracleConnection.serverName
                    + ":" + OracleConnection.port
                    + ":" + OracleConnection.sid;
            OracleConnection.connect =
DriverManager.getConnection(OracleConnection.url, props);
        }
```

Tato metoda je využita při startu aplikace v metodě `main()`. Při úspěšném navázání spojení je otevřeno dialogové okno sloužící pro přihlášení (Obrázek 6) uživatele.

3.5 Základní uživatelské rozhraní

Tato část podkapitoly postupně probere hlavní třídy, metody a diagramy (Use case diagram, class diagram).

3.5.1 Přihlášení uživatele

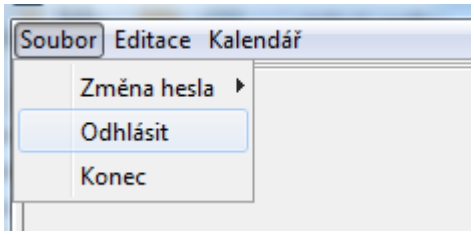
K ověření uživatele slouží třídy z balíčku `administrace`. Balíček obsahuje třídu `DMLAutorizace` určenou pro základní DML operace nad tabulkou uživatelů, třídu `Autorizace` ověřující heslo, které uživatel zadá pomocí dialogového formuláře (Obrázek 6).

Obrázek 6 - Dialogové okno pro přihlášení uživatele. Zdroj: autor.

Pokud uživatel správně zadá ověřující parametry, je otevřeno potvrzující dialogové okno a poté i hlavní formulářové okno, které zastřešuje třída `AplikaceZkusebna`.

3.5.2 Ovládací formulářové okno

Formulářové okno nabízí tři základní tzv. „menu položky“(Obrázek 7).



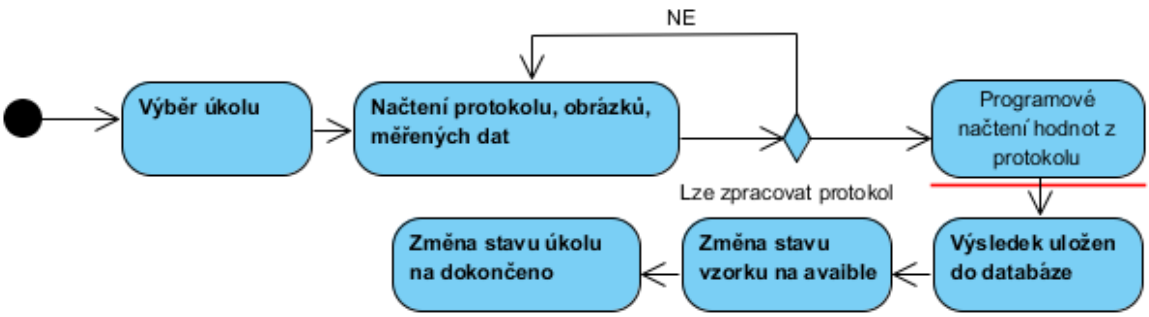
Obrázek 7 - Ovládací položky aplikace. Zdroj: autor.

Odstavec probere podrobný popis ovládacích položek aplikace a nastínění základních funkce aplikace.

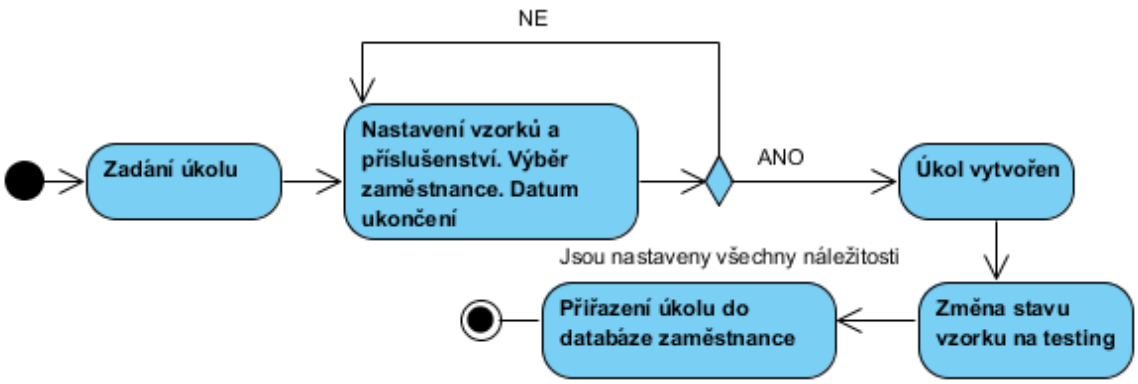
Položka **Soubor** obsahuje základní operace, které jsou přítomny v každé větší aplikaci. Aplikaci lze ukončit položkou **Konec**. Administrátor při vytvoření nového uživatele nastaví defaultní heslo, proto je nutné umožnit uživateli **změnu hesla**, popřípadě bezpečné **odhlášení** z aplikace.

3.6 Funkcionality aplikace

Pro lepší pochopení všech funkcí, které tato aplikace umožňuje, následuje obrázek 8 a 9.



Obrázek 8 - Activiti diagram zaměstnance. Zdroj: autor.



Obrázek 9 - Activity diagram administrátora. Zdroj: autor.

Odstavec vysvětlí, jak probíhá práce na oddělení zkušebna. Oddělení přímo spolupracuje s oddělení vývoje, pro které testuje vzorky podle předem vytvořeného plánu zkoušek. Například vývoj v automobilce. Tato firma vyvíjí několik modelu automobilů zároveň.

Každý model musí úspěšně absolvovat sérii zkoušek, které nařizuje Evropská unie a firemní politika. Tato série zkoušek je rozdělena do menších částí tzv. „Test loty“. Pro příklad, prvním „test lotem“ bude testování podvozků. Vývoj svůj prototyp podvozku nechá otestovat zkušebnou. Pokud mají testy kladné výsledky, může vývoj zahájit druhou fázi (test lot) například vývoj motoru. Tyto fáze pokračují do doby, než je vyvinut finální prototyp, který je posléze možné sériově vyrábět a prodávat. Aplikace pro záznam výsledků zkoušek v jednotlivých *test lotech* modelu disponuje přehlednou tabulkou (Test plán).

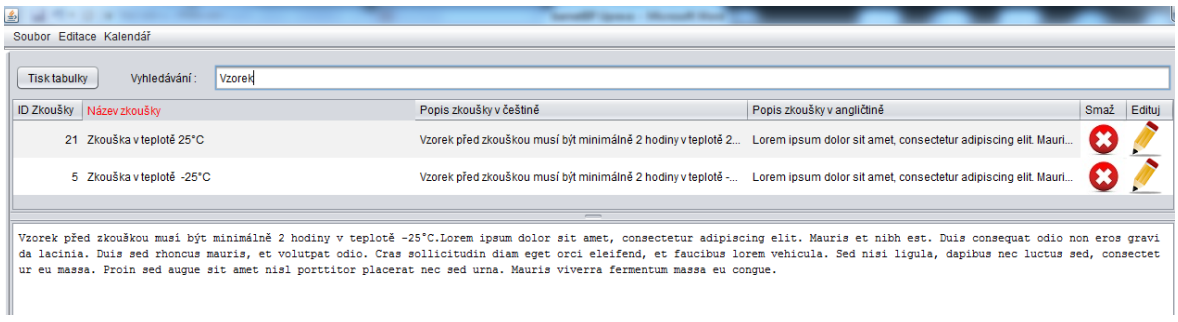
Obrázek 10 - Use case diagram. Zdroj: autor.

K hlavním funkcím aplikace se lze dostat přes menu **Editace** (Obrázek 7) v hlavním formulářovém okně. Přístup k funkcím se liší podle role uživatele. Přehled funkcí je zobrazen pomocí **use case diagramu** (Obrázek 10).

Před ukázkou tabulky **Test plán**, podkapitola probere základní prvky vyskytujícími se v této tabulce.

Zobrazení tabulek v aplikaci je vytvořeno pomocí třídy `JPanelEditaceObecne`, která dědí třídu `Jpanel`. Tento panel obsahuje tlačítko pro tisk tabulky a `JTextbox` určený pro vyhledávání záznamů v tabulce. Speciálně pro tento `JTextbox` byla napsána třída `NapovedaJProTextFiled`, díky které je možné zobrazit nápovědu před vyhledáváním. Dále panel obsahuje komponentu `JSplitPanel` (komponenta rozdělí panel na dvě podokna s posuvníkem, který mění poměr velikost oken). V jedné části komponenty je zobrazena tabulka, v druhé komponenta `JTextArea` zobrazující celý text buňky tabulky. Sloupce tabulky disponují automatickým řazením a filtrem řádků. Obrázek 9 nastiňuje funkce filtrování spojené s vyhledáváním. Nastavení filtru na sloupce docílíme dvojklikem na hlavičku tabulky. Následně je zobrazen formulář, do kterého zadáme požadovaný text. Takto nastavený sloupec je vyznačen červenou barvou hlavičky.

Jednotlivé třídy, které tuto tabulku plní daty z databáze, implementují rozhraní `INastaveniEditace`. V nabídce **Editace** v hlavním okně aplikace se jednoduše dostaneme k jednotlivým funkcím aplikace, které si v následujících odstavcích představíme.



Obrázek 11 - JpanelEditaceObecne. Zdroj: autor.

3.6.1 Editace zkoušek

Zobrazení dat z databáze zprostředkovává třída `EditaceTypuZkousek`. Třída nastavuje komponenty panelu `JPanelEditaceObecne` (Obrázek 11). Tabulka

zobrazuje následující parametry zkoušky: "ID Zkoušky", "Název zkoušky", "Popis zkoušky v češtině", "Popis zkoušky v angličtině", "Smaž", "Edituj". Zkoušky lze jednoduše přidat do evidence pomocí tlačítka v horní části panelu. Následně je otevřen dialogový formulář obsahující tři komponenty `JTextArea`, které slouží pro vyplnění hodnot: název zkoušky, popis v češtině a angličtině. Každý řádek obsahuje dvě ikony. Červený kříž pro smazání zkoušky z databáze a tzv. „tužku“ pro editaci zkoušky. Formulářové okno pro editaci je shodné s oknem pro přidání zkoušky, komponenty jsou pouze předvyplněné stávajícími hodnotami. Ikony jsou v tabulce zobrazeny díky třídě `JTableRender`. Tato vlastní třída umožňuje do tabulky vložit komponentu `JLabel`, díky které je možné požadovaný obrázek zobrazit. Pomocí správně nastaveného `listeneru` tabulky, lze jednoduše zjistit, kterou ikonu uživatel stiskl.

3.6.2 Editace uživatelů

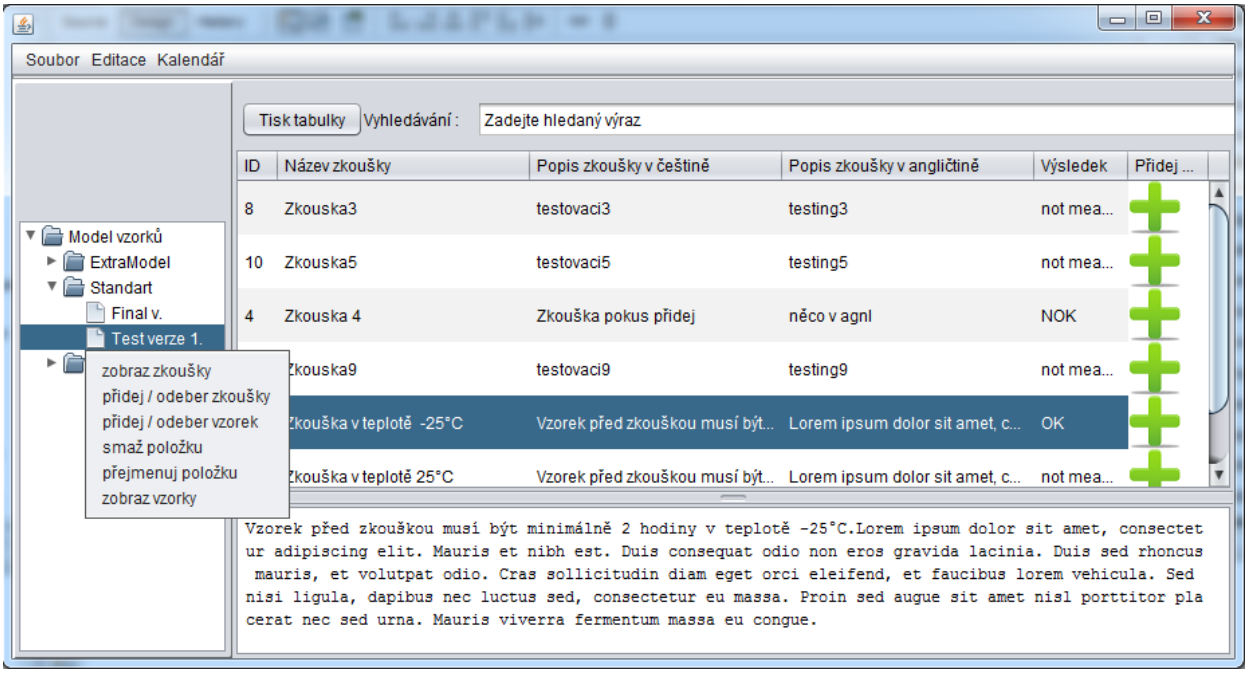
Hlavní část oddělení zkušebna tvoří zaměstnanci. Administrátor disponuje možností editace jednotlivých zaměstnanců. Pro tuto potřebu existuje třída `JPanelEditaceUzivatele` odvozena ze třídy `JPanel`. Třída střídavě zobrazuje dva panely. Prvním je ovládací panel s tlačítky pro přidání, editaci a mazání zaměstnance. Dále je tu komponenta `JList` zobrazující existující zaměstnance. Po stisku tlačítek přidej nebo edituj je zobrazen panel druhý, obsahující komponenty pro vyplnění základních atributů (jméno, příjmení, přihlašovací jméno, heslo, email atd.) zaměstnance. Hesla jsou šifrována pomocí SQL funkce:

```
sys.dbms_crypto.hash(utl_raw.cast_to_raw(„Heslo123“),1))
```

3.6.3 Test plán

Hlavní funkcí celé aplikace je **Test plán.** Z obrázku 12 je patrné rozložení na dva základní panely. Levý panel plní řídící funkce, pravý tyto požadavky zobrazuje. Levá strana je tvořena datovou strukturou strom. Jsou zde zobrazeny jednotlivé modely spolu s jejich test loty. Na pravé straně se po označení test lotu zobrazí přiřazené zkoušky. Levá strana je řízena třídou `JPanelEditaceJTreeModelZkousek`. Pravým stiskem tlačítka myši na položku stromu je zobrazena nabídka funkcí. Modely a jejich test loty lze touto nabídkou dynamicky editovat (přidání, smazání, přejmenování). Dynamický strom [21] využívá vlastní třídu `JTreeModel` .Nabídka je rozšířena o funkce při označení test lotu. Zkoušky lze přidat či odebírat pomocí dialogového formuláře. Jde o panel s tabulkou obsahující všechny zkoušky. Pomocí `checkboxů` v každém řádku lze jednoduše označit,

které zkoušky chce přidat či odebrat. Na pravé straně je mimo informací o zkoušce také v každém řádku zelené tlačítko sloužící pro přiřazení úkolu (testu zkoušky na vzorku). Při stisku na zelenou ikonu je otevřeno editační okno. Administrátor má možnost přiřadit, jaké chce otestovat vzorky na dané zkoušky, který ze zaměstnanců bude úkol vykonávat a s jakým příslušenstvím bude test vykonáván.



Obrázek 12 - Test plán. Zdroj: autor.

Dále má možnost připsat poznámku k testu pro informovanost zaměstnance a zadá datum, do kdy je nutné zkoušku vyhotovit. Pokud zaměstnanec vyhotoví úkol a nahraje do aplikace protokol ze zkoušky, v pravé části test plánu (obrázek 12) je zobrazen výsledek z testu. Výsledky mají tři stavy: `not mesure`, `OK`, `NOK`. Dvojklikem na stav protokolu následuje jeho otevření. Pokud se administrátor dožaduje více informací, kliknutím pravého tlačítka je zobrazena nabídka funkcí:

- Zobrazit aktuální protokol – aplikace nalezne defaultní program pro zobrazení protokolu, který následně otevře.
- Zobrazit historii protokolů – je zobrazena tabulka všech protokolů, které byly vyhotoveny na danou zkoušku v test lotu.

Administrátor není omezen počtem úkolů na jednu zkoušku. Do tabulky je vždy zobrazen nejnovější výsledek odevzdaného protokolu. Tato funkce generování sumářů je demonstrována na následujícím SQL příkazu, který využívá analytické funkce Rank:

```
nastVypisZkousek.naplnTabulkuZakladniData
(DMLOperace.select( new String[]{"*"},
      "( select id_test_plan, id_zkousky,"
     + "nazev_typ_zkousky,popis_cesky,popis_angl,"
     + "nazev,cesta_protokolu, id_protokolu,"
     + "rank () over (PARTITION by id_test_plan "
     + "order by datum_ulozeni desc) as rank "
     + "from  b_naplanovane_ukoly_vzorku "
     + "join b_protokol using (id_plan) "
     + "join b_vysledek_protokolu using "
     + "(ID_VYSLEDEK_PROTOKOLU) "
     + "right join b_test_plan using (id_test_plan) "
     + "join b_test_lot using (id_test_lot)  "
     + "join b_typ_zkousky using (id_zkousky) "
     + "where id_test_lot = "
     + child.getId() // id test lotu
     + " ) where rank = 1 "
       ));
```

Každý test lot obsahuje také evidenci vzorků. Stiskem pravého tlačítka na test lot je zobrazena nabídka s funkcí **zobraz vzorky**. Tato funkce zobrazí panel s evidencí vzorků (zobrazení i funkce jsou stejné jako v případě editace zkoušek). Další funkcí **přidat vzorek** umožníme přidání vzorku do evidence pomocí jednoduchého formuláře, který obsahuje komponenty pro vyplnění základních parametru vzorku. Pokud administrátor přiřadí vzorek k testu, jeho aktivita je změněna z `available` na `testing`. Vzorek se statusem `testing` není možné přiřadit k úkolu. Poté co zaměstnanec odevzdá protokol, stav vzorku je znovu změněn. Pokud byl vzorek během testování poškozen a není nadále vhodný pro další testování, zaměstnanec jednoduše změní jeho stav na `broken` a vzorek je vyřazen. Aplikace tento vzorek i nadále eviduje pro účely zálohy.

3.6.4 Protokoly

Každý zaměstnanec ke svému úkolu zpracuje protokol, který následně nahraje do aplikace. Samotné nahrávání bude probráno později, nyní si ukážeme zobrazení samotných protokolů spolu s jejich daty (obrázky, data z měření). Zobrazení protokolů je rozděleno na dvě tabulky. První tabulka zobrazuje všechny vytvořené protokoly řazené podle data odevzdání sestupně. Zobrazeny jsou všechny detaily o protokolu: cesta k protokolu, název,

výsledek, datum uložení. Dvojklikem na příslušný protokol docílíme jeho zobrazení. Vše řídí třída `NastaveniVypisProtokol`. Tabulky jsou zobrazeny pomocí standartní třídy aplikace `JPanelObecne`. Lze tedy jednoduše záznamy filtrovat podle zaměstnance nebo měsíce odevzdání.

Zaměstnanec při odevzdávání pouze pomocí `JFileChooser` vybere cesty k protokolu, obrázkům a pomocným datům (měřené hodnoty, video, atd.). Protokol je vyplněn do předem vytvořené šablony v programu MS Excel. Z předem určených buněk umí aplikace data vyseparovat (výsledek, poznámka, důvody v případě neúspěšného testu). Zpracování protokolu řídí třída `ZpracujExcel`. Pro přístup k datům programu MS Excel je využita knihovna `poi-ooxml` verze 3.9. Metody této třídy umožňují přístup k jednotlivým buňkám souboru. Ukázka přístupu k listu souboru:

```
private Sheet nactiWorkbook(String path) throws
FileNotFoundException, IOException,
InvalidFormatException {
        try{
           FileInputStream inp = new FileInputStream(new
File(path));
        Workbook wb = WorkbookFactory.create(inp);
        inp.close();
        return wb.getSheetAt(0);
           // blok catch
    }
```

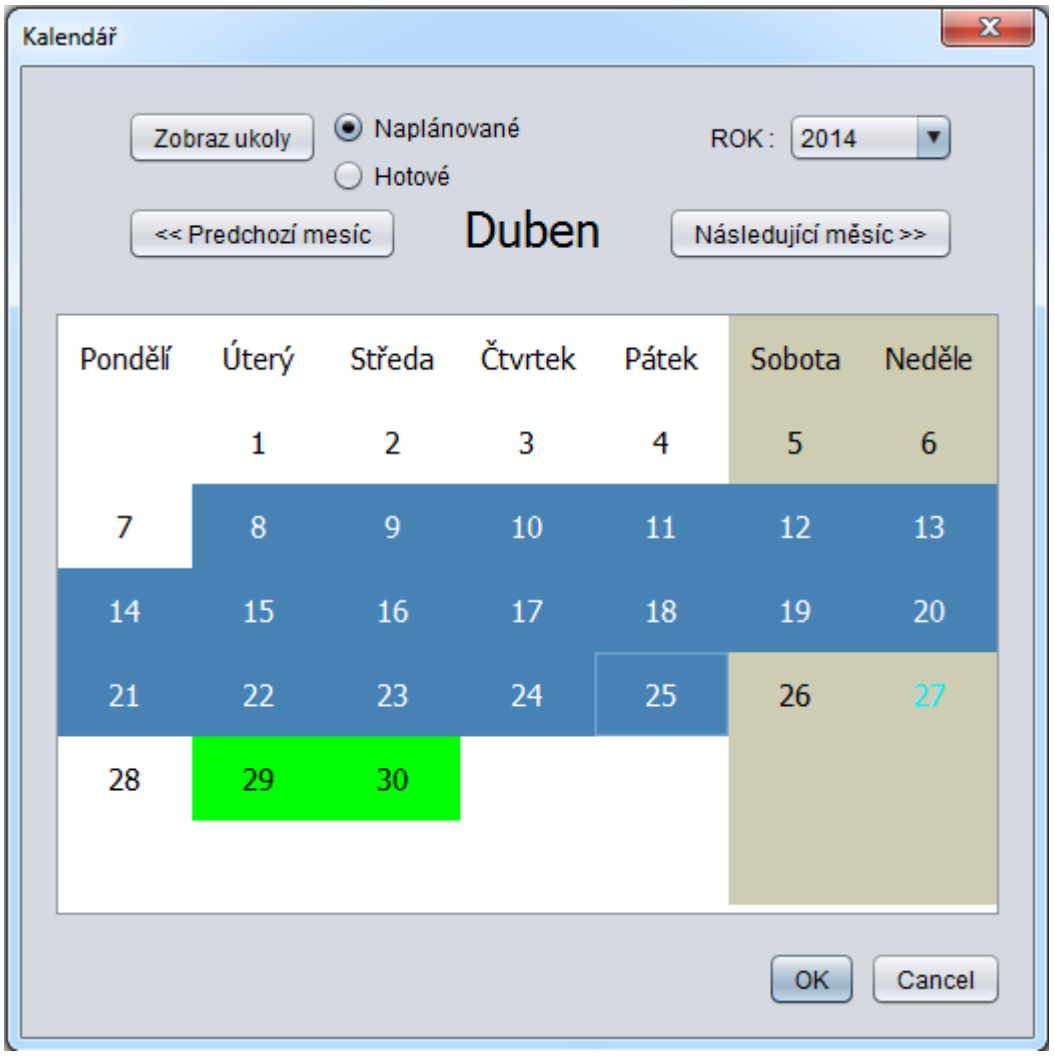
3.6.5 Úkoly

Administrátor pomocí test plánu zadává úkoly jednotlivým uživatelům. Tyto úkoly si uživatele jednoduše zobrazí pomocí menu **Editace** a položky **úkoly zaměstnance**. Pro výpis je využit základní panel s tabulkou `JPanelObecne`. Administrátor má možnost prohlížení všech zadaných úkolů v aplikaci, zaměstnanci mají přístup pouze k aktivním úkolům, které jim byly přiřazeny. Výpis pro zaměstnance obsahuje také v každém řádku tabulky zelenou ikonu pro přiřazení protokolu k úkolu.

3.6.6 Kalendář

Pro lepší přehlednost naplánovaných a hotových zkoušek aplikace disponuje kalendářem (Obrázek 13). Jedná se o klasický kalendář zobrazující dny v daném měsíci. Pomocí uživatelských tlačítek (předchozí, následující měsíc) je možné měnit měsíc i rok zobrazení. Světle modrá barva označuje aktuální den. Kalendář pomocí komponenty `checkbox`

umožňuje přepínání mezi stavy úkolů. Jednotlivé zkoušky jsou barevně odlišeny. Pro naplánované (neuložené) zkoušky, které mají termín měření před sebou, jsou označeny zeleně. Zkoušky po termínu jsou vyznačeny červeně. Protokoly uložené do aplikace jsou označeny žlutou barvou. Detail úkolu lze jednoduše vypsat pomocí tlačítka **Zobraz ukoly**. Označené dny mají modrou barvu. Existuje možnost výpisu jednoho nebo více dnů. Pro možnost více dnů při označování je důležité stisknout klávesu *Shift*. Není nutné se omezovat pouze na dny, lze vypisovat detaily z více měsíců či roků, princip je stejný. Zobrazení kalendáře zprostředkovává třída `KalendarUkolu`.



Obrázek 13 - Kalendář. Zdroj: autor.

4 Závěr

Cílem této práce bylo vytvořit aplikaci, která bude sloužit pro evidenci dat měřených na oddělení Zkušebna. Aplikace je konstruována pomocí jazyku Java (verze Javy 1.6.0). Pro ukládání dat aplikace byl zvolen databázový systém společnosti Oracle, konkrétně Oracle Database 11g. Pří návrhu aplikace bylo nutné seznámit se přístupovými metodami jazyka Java do relační databáze. Kapitola postupně představila základní unifikované rozhraní a možnosti uložení persistentních dat. Po provedené rešerši byl pro naši aplikaci vybrán způsob s využitím základního rozhraní JDBC (verze 11.2.0). Aplikace disponuje potenciálem pro nasazení techniky ORM. Dalším úkolem bylo vypracovat vhodný databázový model. V praktické části byl model vypracován přesně podle teorie popsané ve třetí kapitole této práce.

Myslím, že se mi podařilo vytvořit univerzální logický model, který je možné převést nejenom na databázový server Oracle ale i na databázi MySQL nebo SQL Server. Oproti tomu podoba aplikace je implementována pouze na systém Oracle a vytvořena pro požadavky jednoho konkrétního oddělení zkušebny.

Aplikace postupně implementuje všechny funkcionality ze zadání práce. Aplikace zvládá editaci měřených dat, umožňuje evidovat seznamy testovacích zařízení, testovaných vzorků a evidenci zaměstnanců. Z těchto dat je možné generovat příslušné sumáře vhodně pro management. Pro další usnadnění obsahuje aplikace kalendář zobrazující plánované a uskutečněné zkoušky. Uživatelům je umožněna funkce automatického zpracování protokolů typu Excel souboru. Nejtěžší fází návrhu se překvapivě ukázala úprava standartních Java swing komponent (`JTable` a `JTree`). Díky velkému množství materiálů bylo připojení a samotná komunikace se serverem Oracle jednoduchá. Těžší fází návrhu se ukázalo graficky přívětivé zpracování aplikace. Myslím, že standartní platforma J2SE nebyla příliš vhodná pro aplikace s grafickým rozhraním.

Aplikace by v budoucnu mohla evidovat pracovní cesty zaměstnanců, manuály nebo interní dokumenty. Další úprava by měla rozšířit uživatelské role pro zobrazení hlavních výsledků veřejnosti. Aplikace má potenciál implementace i na jiné oddělení firmy.

Literatura

[1] ŠEDA, Jan. Interval.cz. *Úvod do JDBC* [online]. 04.03.2003. [cit. 2014-05-02]. Dostupné z: http://interval.cz/clanky/uvod-do-jdbc/. ISSN 1212-8651

[2] ORACLE CORPORATION. *JDBC Overview* [online]. [cit. 2014-05-02]. Dostupné z: http://www.oracle.com/technetwork/java/overview-141217.html

[3] ORACLE CORPORATION. *Lesson: JDBC Introduction* [online]. 18.3.2014 [cit. 2014-05-02]. Dostupné z: http://docs.oracle.com/javase/tutorial/jdbc/overview/index.html

[4] JDBC driver. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001, 24.3.2014 [cit. 2014-05-02]. Dostupné z: http://en.wikipedia.org/wiki/JDBC_driver

[5] ORACLE CORPORATION. *Lesson: JDBC Basic* [online]. 18.3.2014 [cit. 2014-05-02]. Dostupné z: http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

[6] FOWLER, Martin. *Patterns of Enterprise Application Architecture*. 2002. vyd. Addison Wesley, 2002. ISBN 0-321-12742-0.

[7] AMBER, Scott. *Mapping Objects to Relational Databases: O/R Mapping In Detail.* AGILE ALLIANCE. [online]. 2006 [cit. 2014-05-02]. Dostupné z: http://www.agiledata.org/essays/mappingObjects.html

[8] BERNARD, Emamnuel, Steve EBERSOLE a Gavin KING. *Hibernate EntityManager* [online]. 15.9.2010. 2010 [cit. 2014-05-02]. Dostupné z: http://docs.jboss.org/hibernate/entitymanager/3.5/reference/en/html_single/#d0e46

[9] JENDROCK, Erick, Ricardo CERVERA-NAVARRO, Ian EVANS, Devika GOLLAPUDI, Kim HAASE, William MARKITO a Chinmayee SRIVATHSA.*The Java EE 6 Tutorial* [online]. 1.2013. 2013 [cit. 2014-05-02]. Dostupné z: http://docs.oracle.com/javaee/6/tutorial/doc/docinfo.html

[10] HIBERNATE. *Hibernate. Everything data.* [online]. 8.4.2014 [cit. 2014-05-02]. Dostupné z: http://hibernate.org/orm/documentation/

[11] BURGET, Radek. *JDO: Java Data Objects*. Interval.cz [online]. 6.1.2005. 2005 [cit. 2014-05-02]. Dostupné z: http://interval.cz/clanky/jdo-java-data-objects/. ISSN 1212-8651

[12] BERNARD, Emamnuel, Steve EBERSOLE a Gavin KING. *Hibernate EntityManager* [online]. 15.9.2010. 2010 [cit. 2014-05-02]. Dostupné z: http://docs.jboss.org/hibernate/orm/3.6/reference/en-US/html/mapping.html

[13] CONOLLY, Thomas, Carolyn BEGG a Richard HOLOWCZAK. Mistrovství - databáze: profesionální průvodce tvorbou efektivních databází. Brno: Computer Press, 2009. ISBN 978-802-5123-287.

[14] TROCH, Josef. ORACLE: *Základní datové typy a základní operace s tabulkami.* [online]. 4.3.2002. 7.3.2002 [cit. 2014-05-02]. Dostupné z: http://jt.wz.cz/vytvory/oracle/oracle.htm

[15] BAUER, Christian a Gavin KING. Hibernate in action. Greenwich: Manning Publications, 2005, xxiii, 408 s. ISBN 19-323-9415-X

[16] GROFF, James R, Paul N WEINBERG a Richard HOLOWCZAK. SQL: kompletní průvodce. Vyd. 1. Brno: CP Books, 2005, 936 s. ISBN 80-251-0369-2.

[17] SCHILDT, Herbert. Java 7: výukový kurz. 1. vyd. Brno: Computer Press, 2012, 664 s. ISBN 978-80-251-3748-2.

[18] Silberschatz, A., Korth H.F, Sudarshan, S.:Database System Concepts. 6. vyd. McGraw-Hill: Education, 2010, 1376 s. ISBN 0-07-352332-1

[19] BURGET, Radek. *JDO: Práce s persistentními objekty*. Interval.cz [online]. 3.2.2005. 2005 [cit. 2014-05-02]. Dostupné z: http://interval.cz/clanky/jdo-prace-s-persistentnimi-objekty/. ISSN 1212-8651. ISSN 1212-8651

[20] OLIVER, Andrew C., Glen STAMPOULTZIS, Avik SENGUPTA, Rainer KLUTE a David FISHER. *Apache POI.* APACHE POI. [online]. 2002-20014 [cit. 2014-05-06]. Dostupné z: http://poi.apache.org/

[21] SUN MICROSYSTEMS, Inc. *Dynamic Tree* [online]. 1995 - 2008 [cit. 2014-05-06]. Dostupné z: http://www.java2s.com/Tutorial/Java/0240__Swing/DynamicTree.htm

Příloha A – Relační návrh databáze