

# Webová aplikace využívající XML funkce databáze Oracle

Web application using XML functions of Oracle database

Bc. Lukáš Svoboda

---

Diplomová práce  
2012



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky  
akademický rok: 2011/2012

## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Lukáš SVOBODA**  
Osobní číslo: **A10333**  
Studijní program: **N 3902 Inženýrská informatika**  
Studijní obor: **Počítačové a komunikační systémy**

Téma práce: **Webová aplikace využívající XML funkce databáze Oracle**

Zásady pro vypracování:

1. Nastudujte možnosti použití XML v databázi Oracle.
2. Nastudujte možnosti využití databáze Oracle ve webové aplikaci a proveďte literární rešerši.
3. Vyberte vhodné řešení pro webovou aplikaci využívající XML v databázi Oracle a svůj výběr zdůvodněte.
4. Naprogramujte webovou aplikaci, která bude využívat zpracování XML v databázi Oracle.
5. Aplikaci zprovozněte na školním serveru.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. LACKO, Luboslav. ORACLE: Správa, programování a použití databázového systému. Vyd. 1. Brno: Computer Press, 2003, 464 s. ISBN 80-722-6699-3.
2. LECKY-THOMPSON, Ed a Steven D NOWICKI. PHP 6: programujeme profesionálně. Vyd. 1. Překlad Ondřej Gibl. Brno: Computer Press, 2010, 718 s. Programujeme profesionálně. ISBN 978-802-5131-275.
3. KOSEK, Jiří. PHP - tvorba interaktivních internetových aplikací: podrobný průvodce. Vyd. 1. Praha: Grada, 1999, 490 s. Průvodce (Grada). ISBN 80-716-9373-1.
4. MLÝNKOVÁ, Irena a Jaroslav POKORNÝ. XML technologie: principy a aplikace v praxi. 1. vyd. Praha: Grada, 2008, 267 s. Průvodce (Grada). ISBN 978-802-4727-257.
5. URMAN, Scott, Ron HARDMAN a Michael MCLAUGHLIN. Oracle: programování v PL/SQL. Vyd. 1. Překlad Jiří Fadrný. Brno: Computer Press, 2007, 720 s. ISBN 978-802-5118-702.

Vedoucí diplomové práce: **Ing. Kateřina Ježková**

Datum zadání diplomové práce: **24. února 2012**

Termín odevzdání diplomové práce: **28. května 2012**

Ve Zlíně dne 24. února 2012

  
prof. Ing. Vladimír Vašek, CSc.  
*děkan*



L.S.

  
prof. Ing. Karel Vlček, CSc.  
*ředitel ústavu*

## ABSTRAKT

Táto práca sa zaoberá vytváraním webovej aplikácie využívajúcej XML funkcionality v databáze Oracle, za pomoci programovacieho jazyka PHP.

V teoretickej časti sa práca zaoberá vysvetlením pojmov ako databázové systémy, v krátkosti popisuje Oracle, jeho štruktúru ako aj popis objektového modelu Oracle. V stručnosti je vysvetlená skupina príkazov jazyka SQL. V ďalšej časti sa práca zaoberá XML funkcionality ako aj jej využitím v praxi.

Druhým hlavným celkom práce je praktická časť. Obsahuje návrh riešení, či už z databázového hľadiska, užívateľského alebo aplikačného. Vysvetlené sú postupy a princípy akým spôsobom bolo v práci postupované a uvedené dôvody prečo to tak bolo.

Kľúčová slova: Oracle, XML, PHP, XPath, SQL, webová aplikácia

## ABSTRACT

In this thesis I deal with creating a web application which is using XML functions in the Oracle database. PHP programming language is also used to develop this application. This thesis is divided into several parts.

The theoretical part of this thesis is dealing with the explanation concept of the database system, briefly describes the Oracle system, its structure and the objected model of Oracle. The group of commands from the SQL language is also briefly explained. The next part of this thesis is dealing with XML functions and its practical use.

The second main goal of my thesis is the practical part. It consists of the design of the solution, the database, user and application aspects. It also explains the procedures and principles of how I process during work, as well as mentions the reasons why it was like that.

Keywords: Oracle, XML, PHP, XPath, SQL, webová aplikácia

Chcel by som sa poďakovať mojej konzultantke Ing. Kateřině Ježkovéj, ktorá má viedla, usmerňovala, pri tvorbe mojej diplomovej práce. Ďalej by som chcel poďakovať mojim rodičom za trpezlivosť a podporu, ktorú mi preukazovali. V neposlednom rade chcem poďakovať Petre Maliarikovej za pomoc pri vytváraní grafiky k aplikácii.

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....  
podpis diplomanta

**OBSAH**

<b>ÚVOD</b> .....	<b>10</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>11</b>
<b>1 DATABÁZOVÉ SYSTÉMY</b> .....	<b>12</b>
<b>2 ORACLE</b> .....	<b>14</b>
2.1 LOGICKÁ ŠTRUKTÚRA .....	14
2.2 FYZICKÁ ŠTRUKTÚRA .....	15
2.3 SCHÉMA VS. USER.....	15
2.4 OBJEKTOVÝ MODEL ORACLE.....	15
2.5 ROZDELENIE PRÍKAZOV JAZYKA SQL .....	16
2.5.1 DDL (Data Definition Language) .....	16
2.5.2 DML (Data Manipulation Language) .....	16
2.5.3 DCL (Data Control Language).....	16
2.5.4 TCC (Transaction Control Commands).....	16
2.6 TYPY PRE UKLADANIE OBJEMNÝCH DÁT.....	17
<b>3 XML</b> .....	<b>18</b>
3.1 XML DOKUMENT .....	18
3.2 XML V ORACLE.....	18
3.2.1 XML schéma .....	19
3.3 XPATH .....	21
3.4 VYUŽITIE XML.....	21
<b>4 PHP</b> .....	<b>22</b>
4.1 OOP V PHP .....	22
4.2 PHP A ORACLE .....	22
4.2.1 OCI Prepared statements.....	24
4.3 VYUŽITIE XML V ORACLE A PHP .....	24
<b>II PRAKTICKÁ ČÁST</b> .....	<b>26</b>
<b>5 NÁVRH APLIKÁCIE</b> .....	<b>27</b>
5.1 DATABÁZOVÝ NÁVRH .....	27
5.1.1 Tabuľka USERS.....	28
5.1.2 Tabuľka USER_DATA.....	28
5.1.3 Tabuľka TRACK_LIST .....	29
5.1.4 Tabuľka STATEMENTS .....	30
5.1.5 Tabuľka SESSIONS.....	30
5.1.6 Tabuľka PLAYLISTS .....	30
5.1.7 Tabuľka NADPISY .....	31
5.1.8 Tabuľka GENRES.....	31
5.2 NÁVRH APLIKÁCIE Z PROGRAMÁTORSKÉHO HĽADISKA.....	32
5.2.1 Triedy .....	32
5.2.1.1 ClassConnectDB.....	32
5.2.1.2 ClassController .....	33
5.2.1.3 ClassAuth.....	33
5.2.1.4 ClassSecurity .....	33

5.2.1.5	ClassSessions .....	34
5.2.1.6	ClassStatements .....	35
5.2.1.7	ClassTokenArray .....	36
5.2.1.8	ClassOracle .....	37
5.2.2	Registrácia .....	37
5.2.3	Login .....	39
5.2.4	Logout (Odhlásenie).....	40
5.2.5	Užívateľské rozhranie .....	40
5.2.6	Vyberanie XML údajov z Oracle .....	42
5.2.7	Prehľad a nastavenia účtu.....	44
5.2.8	Pridávanie skladieb .....	46
5.2.9	Úprava skladieb.....	48
5.2.10	Vytvorenie a uloženie profilu.....	48
5.2.11	Vyhľadávanie .....	51
5.2.12	Chybové oznamy a nadpisy .....	52
5.3	UŽIVATELSKÝ NÁVRH APLIKÁCIE.....	53
5.3.1	Využitie aplikácie.....	54
<b>6</b>	<b>MOŽNÉ NÁVRHY NA VYLEPŠENIE .....</b>	<b>55</b>
6.1	VYHLADÁVANIE.....	55
6.2	ZÍSKAVANIE VZDIALENÉHO OBSAHU .....	55
6.3	SÚKROMNÉ VS. VEREJNÉ ZOBRAZOVANIE.....	56
<b>7</b>	<b>SPOJAZDZENIE NA SERVERI.....</b>	<b>57</b>
7.1	VYTVORENIE SCHÉMY .....	57
7.2	ADRESÁROVÁ ŠTRUKTÚRA .....	58
	<b>ZÁVĚR .....</b>	<b>60</b>
	<b>ZÁVĚR V ANGLIČTINĚ.....</b>	<b>61</b>
	<b>SEZNAM POUŽITÉ LITERATURY.....</b>	<b>61</b>
	<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....</b>	<b>64</b>
	<b>SEZNAM OBRÁZKŮ .....</b>	<b>65</b>
	<b>SEZNAM TABULEK.....</b>	<b>66</b>
	<b>SEZNAM PŘÍLOH.....</b>	<b>67</b>

## ÚVOD

V súčasnej modernej dobe sme obklopení rôznymi druhmi webových aplikácií. Nikomu netreba snáď predstavovať youtube, ebay, amazon a asi najviac známy facebook. Všetky tieto webové aplikácie majú veľa spoločných črtov. Všetky využívajú obrovské databázy na ukladanie údajov v rôznych formátoch. Hardwareové vybavenie je tiež veľmi dôležité. Taktiež nemôžem nespomenúť rôzne programovacie jazyky či frameworky, ktoré sú alebo boli využívané pri tvorbe hore uvedených aplikácií. Okrem týchto, nazvem to voľne, technických záležitostí však tieto webové aplikácie majú spoločnú ešte jednu vec, ktorá ma ďaleko od databáz, programovania. A to je prínos pre ľudí, ktorí chcú byť v kontakte so svojou rodinou, počúvať hudbu bez nutnosti sťahovania alebo nakupovať v pohodlí domova. Spoločným menovateľom prínosu je uľahčenie života ľuďom v ich každodennom živote.

Moja diplomová práca nemá za úlohu vytvoriť obrovskú aplikáciu nadnárodných rozmerov. Skôr ide o aplikáciu, ktorá by určitým ľuďom mala uľahčiť život. Čo sa týka technického zamerania tejto práce, budem sa snažiť čo najlepšie rozobrať problematiku spojenú s databázovým systémom Oracle, v ňom XML a všetko prepojené pomocou skriptovacieho jazyka PHP. Všetko bude na školskom serveri, na ktorom beží Windows server 2008 64bit. Vzhľadom k pomerne voľnejšiemu zadaniu, v ktorom sú spomenuté hlavné body, som si musel vymyslieť zameranie celej aplikácie. Pre nápad som nemusel ďaleko chodiť. Do značnej miery mi pomohla moja bakalárska práca, ktorá sa venovala vytváraniu playlistov pre indoor cycling. Tento nápad však je do značnej miery až druhoradý, primárnym zámerom bolo vytvorenie hudobnej databázy, ktorá by slúžila inštruktorm indoor cyclingu. Preto sa dobre hodila kombinácia Oracle s XML a PHP. Uvedomujem si, že spojenie komerčnej databázy Oracle s veľmi rozšíreným jazykom PHP je taká neštandardná kombinácia, ale v konečnom dôsledku si myslím, že výsledok je dostatočný. V tejto práci sa budem snažiť priblížiť databázový systém Oracle, XML a spomeniem základné veci z objektového PHP. V praktickej časti sa budem venovať jednotlivým častiam webovej aplikácie. Na záver spomeniem možné vylepšenia aplikácie ako aj jej využitie v praxi.

## **I. TEORETICKÁ ČÁST**

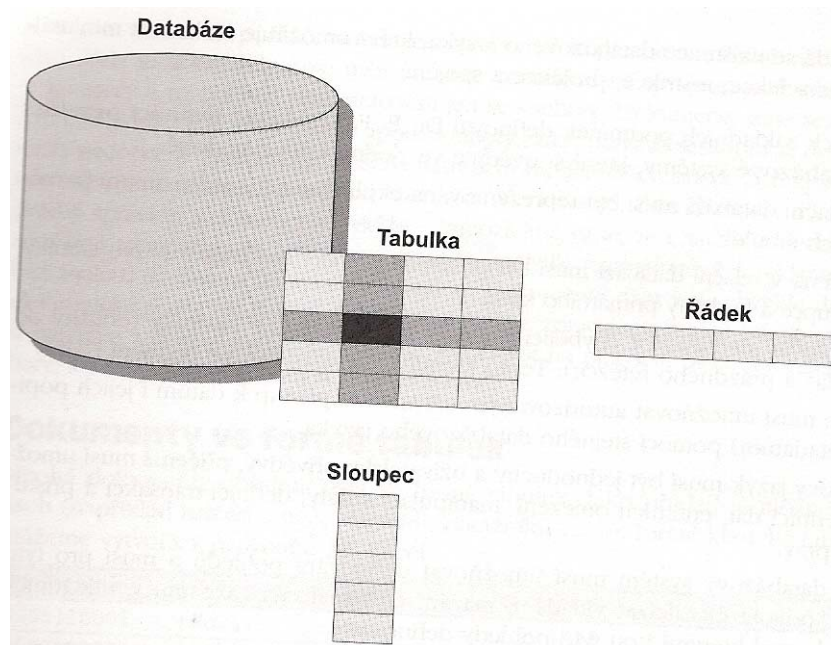
## 1 DATABÁZOVÉ SYSTÉMY

Je možné, že som práve nezvolil najlepší názov kapitoly, ale aspoň mi to dáva priestor lepšie túto problematiku priblížiť. Po správnosti by som to mal skôr nazvať relačné databázy. Pre laika môže tento pojem vzbudzovať rešpekt u niekoho dokonca odpor ale, pochopenie relačných databázových systémov ako celku nie je až také zložité aj keď sa to tak môže zdať. Dost' voľná definícia relačných databáz by sa dala opísať ako sada nástrojov pre efektívne a spoľahlivé ukladanie dát a manipuláciu s nimi [1]. Ak sa posuniem ďalej, tak menšou časťou spadajúcou pod komplexný celok databázového systému je pojem databáza. Preferujem anglický pojem DBMS (Database Management System) a aby som bol aj jazykovo korektný, tak voľný preklad by bol SRBD (Systém Riadenia Báže Dát), ktorý pojem databáza vystihuje asi najlepšie. Aby som lepšie vysvetlil čo to znamená, tak v podstate ide o spojenie dát a nástrojov, ktoré zaisťujú ukladanie a manipuláciu s dátami. V minulosti by sa dala za takú databázu považovať aj obyčajná kartotéka. Nástrojom, ktorý ukladal a pracoval s dátami bola úradníčka a nie sofistikovaný databázový systém. Mám rád názorné príklady, preto spomeniem, že takým spojením s databázami by mohol byť, veľkoryso nazvem, dochádzkový systém tzv. „cvikačky“. Viem, že to nie je úplne dobrý príklad ale aspoň mi to pomôže prejsť na ďalšiu časť, ktorou je štruktúra údajov v databázy. Tak ako spomínané „cvikačky“ tak aj dáta v databázy majú určitú štruktúru ako sú uložené. V oblasti databázových systémov nie je pre ostrieľaných borcov pojem tabuľka žiadny neznámy pojem. Je to základná štruktúra pre uchovávanie dát. Tá sa ďalej delí na stĺpce a riadky. Stĺpce sú dátovo závislé, tzn. že v stĺpci musia byť dáta toho istého typu. Ďalšou časťou tabuľky sú riadky, tiež nazývané ako záznamy. Tie nesú informácie, ktoré sú uložené v tabuľke.

Existujú rôzne druhy databáz. Najrozšírenejšími sú však relačné databázy. Dr. E. F. Codd definoval minimálne tri podmienky, ktoré musia byť splnené, aby sa databáza mohla považovať za relačnú[1]:

- 1) Všetky dáta v databáze sú uložené v tabuľkách
- 2) Fyzická štruktúra dát a ich uloženie sú nezávislé a od užívateľa odtienené (teda neexistujú pre užívateľa viditeľné prístupové cesty – vrátane indexov)
- 3) Predpokladá sa existencia databázového jazyka, ktorý umožňuje realizovať minimálne operácie selekcie, reštrikcie, projekcie a spojenia.

Vyššie spomínaný Dr. Codd, navrhol ďalších dvanásť pravidiel relačných databáz, ale myslím si, že bohato postačia aj tri základné. Radšej ešte spomeniem funkciu primárneho a cudzieho kľúča. V prípade, že tabuľka je dobre navrhnutá, každý riadok by mal byť definovaný primárnym kľúčom. Už zo samotného názvu sa dá tušiť, že tento kľúč spĺňa funkciu jedinečného identifikátora každého riadku v tabuľke. Jedinečnosť v tomto prípade značí, že v stĺpci, ktorý je označený ako primárny kľúč, sa nemôže vyskytovať rovnaká hodnota prípadne hodnota NULL. Ďalším faktom je skutočnosť, že bez primárneho kľúča nie je možná relácia na ďalšiu tabuľku. Okrem primárneho kľúča, existuje aj kľúč cudzí. Jeho úloha spočíva v tom, že spája stĺpec s cudzím kľúčom so stĺpcom s primárnym kľúčom inej tabuľky. Na Obrázku 1 je dobre vidieť štruktúru dát v databáze.



Obrázok 1: Štruktúra dát v databáze[1]

## 2 ORACLE

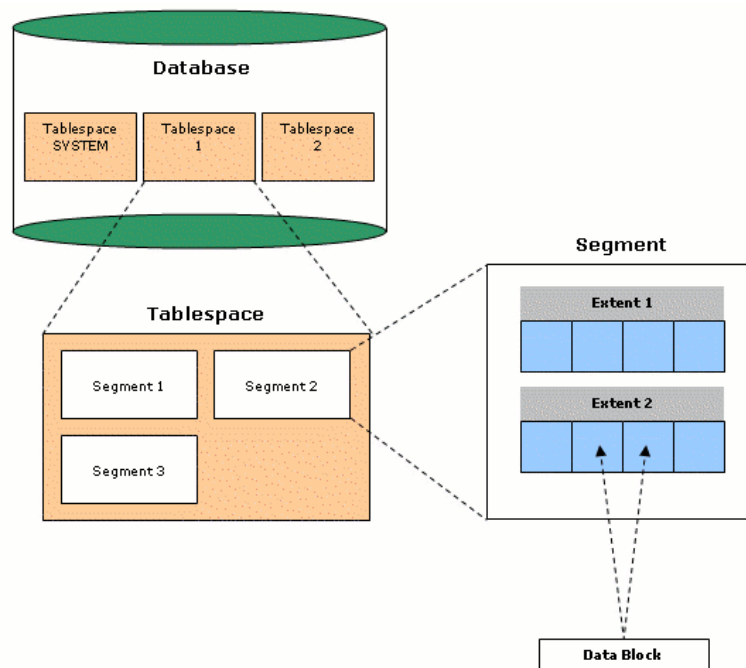
V tejto kapitole sa budem venovať databáze Oracle. Len veľmi stručne, pretože popísať celý systém by vystačilo na samotnú diplomovú prácu.

Čo je vlastne Oracle databáza. Z vlastnej skúsenosti môžem povedať, že to je obrovský systém, ktorý je veľmi dobre premyslený a prepracovaný. Počas tvorby tejto diplomovej práce, som sa stretával s rôznymi problémami a pri ich riešení som musel našťudovať aj veci, ktoré priamo nesúviseli s touto prácou, ale nevyriešenie mi bránilo pokračovať ďalej. Ak to dám všetko dokopy, tak som si uvedomil, aké možnosti ponúka Oracle.

Oracle databáza sa skladá z dvoch hlavných komponentov. Inštancie a databázy samotnej. Inštancia pozostáva z pamäťovej štruktúry a procesov bežiacich na pozadí, kým databáza sa odkazuje na diskové zdroje[8]. Čo sa týka databázovej inštancie, je to komplexná vec, ktorá pozostáva z väčšieho množstva pamäťových štruktúr a procesov. Databázová časť je rozdelená do dvoch hlavných štruktúr. Logická a fyzická štruktúra.

### 2.1 Logická štruktúra

Predstavuje rozdelenie do menších logických jednotiek, ktoré majú za úlohu riadiť, uchovávať a efektívne získať dáta. Jednotlivé logické jednotky sú nasledovné: tablespace, segment, extent a data block.



Obrázok 2: Oracle logické štruktúry

Z obrázka je vidieť, že databáza musí obsahovať minimálne jeden tablespace. Ten je ďalej delený do segmentov. Ten uchováva objekty rovnakých dátových typov. Každý segment je ešte rozdelený do extent, ktorý obsahuje jeden alebo viac blokov dát. A relatívne najmenšia časť je data block určená pre uchovávanie dát v Oracle. Logická štruktúra preto, ak beriem tablespace ako najväčšiu časť, lebo nie je viditeľná v súborovom systéme databázového servera.

## 2.2 Fyzická štruktúra

Pod touto štruktúrou si môžeme predstaviť štruktúry, ktoré nie sú priamo upravované používateľom. Fyzická štruktúra pozostáva z datafiles, redo log files a control files. Prvé menované sa zhoduje s tablespace. Tzn. jeden datafile môže byť použitý jedným tablespace, ale tablespace môže mať viacej datafiles. Redo Log Files sú určené pri obnovovaní databázy. A control files sa používajú na ukladanie informácií ohľadne fyzickej štruktúry databázy ako veľkosť a umiestnenie datafiles, redo log files atď.

## 2.3 Schéma vs. User

Schema a tzv. user majú k sebe veľmi blízko. Schéma predstavuje kolekciu logicky štruktúrovaných dát alebo objekty schémy. Schému vlastní používateľ (user) a zároveň má rovnaké meno ako používateľ. Z toho vyplýva, že každý používateľ má vlastnú schému. Objekty schémy môžu nadobúdať rôzne typy ako clustre, trigre, indexy, externé procedúry, sekvencie, pohľady, uložené procedúry a ďalšie. V praktickej časti sa tejto téme budem venovať viac.

## 2.4 Objektový model Oracle

Na začiatku som spomínal pojem relačná databáza. Oracle spĺňa podmienky pre relačnú databázu, avšak čo som ešte nespomenul je skutočnosť, že implementuje objektový model ako rozšírenie relačného modelu. Pričom si zachováva základnú funkcionality spojenú s relačnými databázami ako sú dotazy, commit príkazy, zálohovanie, obnovovanie atď.

Pod pojmom objektové typy by sme si mohli predstaviť súbor takých istých dátových typov ako napríklad varchar prípadne number. V prípade, ak sa jedná o objektový dátový typ, tak hovoríme o inštancii daného typu. Inštancia objektu sa taktiež nazýva aj objekt. Výhody tohto objektového prístupu sú podobné, ako pri objektovom prístupe

v programovacích jazykoch ako je napr. Java alebo C++. Medzi tieto výhody patrí zapúzdrenie dát, objekty sú efektívne atď.

## 2.5 Rozdelenie príkazov jazyka SQL

Vyššie som spomínal Oracle ako databázu. Aby sme dokázali efektívne pracovať v tomto prostredí, je treba sa oboznámiť aspoň so základnými príkazmi jazyka SQL. Delia sa do štyroch skupín:

- DDL
- DML
- DCL
- TCC

### 2.5.1 DDL (Data Definition Language)

Pomocou príkazov z tejto skupiny môžeme definovať, vytvárať, meniť a rušiť rôzne databázové štruktúry, ako tabuľky, indexy, triggery, uložené procedúry a pod[6]. Medzi tieto príkazy patria, CREATE TABLE, CREATE DATABASE, ALTER TABLE, DROP TABLE, CREATE VIEW atď.

### 2.5.2 DML (Data Manipulation Language)

Ako vyplýva z názvu, do tejto skupiny patria príkazy, ktoré sú určené pre manipuláciu s dátami[1]. Najznámejším je určite SELECT, potom INSERT, UPDATE a DELETE.

### 2.5.3 DCL (Data Control Language)

Táto skupina zahŕňa špeciálne príkazy pre riadenie prevádzky a údržbu databázy. Patria sem: GRANT, REVOKE CREATE USER, ALTER USER, DROP USER, GRANT, REVOKE[1].

### 2.5.4 TCC (Transaction Control Commands)

Prvé tri skupiny patrili medzi základ jazyka SQL. Avšak môžem pridať ešte aj skupinu príkazov pre riadenie transakcií. Medzi ne patria napríklad príkazy: SET TRANSACTION, COMMIT, ROLLBACK, SAVEPOINT[1].

## 2.6 Typy pre ukladanie objemných dát

Túto kapitolu som zaradil sem kvôli tomu, že pri ukladaní XML do tabuliek, je výhodné ukladať pomocou dátových typov pre objemné dáta. V Oracle existujú takéto dátové typy označované ako LOB. Do tejto skupiny patria ešte BLOB, CLOB a NCLOB. Tieto typy sú vhodné pre rozsiahle multimedialne dáta (zvuk, video, a iné), ktoré sa ukladajú priamo do databázy. Pre externé ukladanie týchto dát slúži dátový typ BFILE. V Oracle 10g majú všetky tieto dátové typy obmedzenie veľkosti na maximálne 4GB veľkosť[1]. Rozdiel medzi BLOB a CLOB je nasledovný. BLOB je skratka pre (Binary Large Object). Využíva sa na ukladanie dát ako obrázky, video atď. CLOB je skratka pre (Character Large Object). Je určený na ukladanie textových súborov, ako napríklad XML, CSV, TXT a iné. Ďalším rozdielom je fakt, že CLOB je závislý od kódovania, keďže pracuje s textom a BLOB nie.

### 3 XML

V skratke by som chcel vysvetliť pojem XML a pre aké účely sa tento značkovací jazyk používa. Skratka XML značí eXtensible Markup Language.

Jiří Bráza vo svojej knihe XML, praktické příklady definuje XML nasledovne: XML je štandard definujúci, ako majú vypadáť dokumenty XML. Určuje, akým spôsobom sú označované elementy, ako môžu byť elementy vnorované, akým spôsobom zapisovať atribúty, komentáre a ďalšie konštrukcie. Napriek tomu tento štandard neurčuje, aké tzv. tagy je možné v dokumentoch XML používať – užívateľ si ich vytvára podľa svojich požiadaviek[15]. Tento prístup je odlišný oproti HTML, kde sú pevne dané tagy, ktoré sa smú používať. Preto XML je skôr metajazyk. Pod pojmom metajazyk si treba predstaviť súbor pravidiel, ktoré sú určené pre vytváranie dokumentov. Ako bolo vyššie spomenuté, XML štandard nie je obmedzený na tvrdo definované tagy. Užívateľ má voľné pole pôsobnosti, čo sa týka vytvárania tagov, atribútov, vytváranie vnorovaní, nastavovanie obmedzení a iné.

#### 3.1 XML dokument

Dokument XML obvykle nazývame súbor alebo skupinou súborov, ktoré obsahujú akési značkovanie. Toto značkovanie musí odpovedať pravidlám určeným štandardom XML a obvykle taktiež pravidlám daných štruktúrou. Buď DTD alebo schémou. V prípade, ak nie je určená schéma alebo DTD, dokument XML musí vyhovovať iba základným požiadavkám na vnorovanie elementov a použitie XML konštrukcií[9]. Základné pravidlá sú veľmi oklieštené a musia spĺňať minimálne správne vnorovanie a konvenciu pri používaní ostrých zátvoriek, úvodzovky či apostrofy pri deklarovaní atribútov v elementoch. Ako bolo vyššie spomenuté, aby sme zabezpečili lepšiu kontrolu nad obsahom dokumentu, môžeme použiť buď DTD (Definition Type Document) alebo schému XML. Pri použití buď jedného alebo druhého, môžeme kontrolovať správnu štruktúru (wellformdness) a taktiež validitu. To nám hovorí, že každý dokument, ktorý je odkazovaný na DTD alebo schému, musí ich štruktúre zodpovedať. V prípade opaku validačný parser zahlási chybu.

#### 3.2 XML v Oracle

Databáza Oracle, okrem ostatných objektových vlastností a typov, umožňuje pracovať aj s XML. V praxi to prináša možnosť vytvárať stĺpce, ktoré budú obsahovať dáta vo formáte

XML. Aby som stále nerozprával o tomto formáte všeobecne, tak je tu malá ukážka ako by mohol vyzerat' veľmi krátky XML dokument.

```
<?xml version='1.0' encoding='utf-8'>
  <users>
    <user idUser = '1'>
      <name>Lukas</user>
      <surname>Svoboda</surname>
      <age>24</age>
    </user>
  </users>
```

Práve takýto súbor, by kludne mohol byť v Oracle tabuľke, v stĺpci umožňujúcom uchovávať a vyberať XML dáta. Pre takéto dáta sa využíva objektový typ SYS.XMLType, ktorý umožňuje ukladať do databázových tabuliek priamo vo formáte XML. Skutočnosť, že to je objektový typ znamená, že tento dátový typ môže mať členské funkcie[6]. Pod týmto pojmom sa skrývajú funkcie, ktoré ďalej rozširujú funkcionality samotného objektového typu. Medzi najpoužívanejšie členské funkcie môžem zaradiť XMLType.getStringVal(), XMLType.extract a iné.

### 3.2.1 XML schéma

Jedná sa o definíciu štruktúry. XML Schema je štandard, ktorý sa snažil odstrániť nedostatky DTD. Medzi výhody XML Schema patria:

- Podpora tzv. namespace
- Definícia štruktúry je taktiež vo formáte XML, teda pre návrh schém, je možné použiť bežné XML nástroje.
- Schémy umožňujú definovať dátové typy a veľmi často presne špecifikovať obmedzenia možných hodnôt.
- V prípade násobnosti elementov, je možné kontrolovať možný počet opakovaní.

Samozrejme, že XML Schema má aj svoje nevýhody, medzi ktoré patrí zložitejší mechanizmus, menšia podpora v aplikáciách, nie je možné určenie koreňového elementu a iné[15].

Základom schémy XML je hierarchická štruktúra, odpovedajúca požadovanému dokumentu, tvorená prevažne elementami *element*. Koreňový element je vždy element *schema*, ktorý celú schému zastrešuje[15]. Okrem iného, tento element obsahuje odkaz na menný priestor, ktorý definuje elementy schémy. Na nasledujúcom obrázku je ukážka jednoduchej schémy.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xdb="http://xmlns.oracle.com/xdb" version="1.0">
  <xs:element name="vodici" xdb:defaultTable="PRJ10SVO_vodici">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vodic" maxOccurs="100">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="meno" type="xs:string"/>
              <xs:element name="priezvisko" type="xs:string"/>
              <xs:element name="vek" type="xs:integer"/>
              <xs:element name="zamestnany_od" type="xs:date"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:integer" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Obrázok 3: XML schéma – príklad

Podpora pre XML Schema je v Oracli zaistená. Každá schéma, ktorú používame, musí byť zaregistrovaná v Oracle. Registrácia schémy by mohla vypadat' nasledovne:

```
BEGIN
```

```
  DBMS_XMLSCHEMA.registerSchema(
    SCHEMAURL=>
    'http://localhost/dp/data/xml/schema_interprets.xsd',
    SCHEMADOC=>
    bfilename('MEDIADIR', 'schema_interprets.xsd'),
    CSID => nls_charset_id('AL32UTF8'));
END;
```

Pričom, musíme taktiež vytvoriť tzv. MEDIADIR, v ktorom sa daná schéma fyzicky nachádza.

### 3.3 XPath

Vzhľadom k tomu, že štruktúra XML je stromového charakteru, je treba vedieť správne a dobre v tejto štruktúre vyhľadávať. Na tieto účely existujú rôzne jazyky a jedným z nich je práve XPath (XML Path Language). Patrí medzi dotazovacie jazyky, ktorý sa využíva pri vyhľadávaní cesty, tak aby sa dostal k vyhľadávanému prvku v XML štruktúre. Podľa zadanej cesty, prehľadáva jednotlivé uzly (nodes). V závislosti od použitej funkcie je možné vyhľadávať hodnoty, ako aj vyhľadávať existenciu jednotlivých uzlov. Na dole uvedených príkladoch bude najlepšie vidieť ako XPath funguje v databázy Oracle.

```
select extract(user, '//user') from users;
```

Tento jednoduchý select zobrazí hodnoty v uzle user. V prípade, že chceme zobrazit' uzol username, XPath bude vyzerat' nasledovne:

```
select extract(user, '//user/username') from users;
```

Tieto vyššie uvedené skripty hodnoty zobrazovali, pretože sme použili funkciu extract. Aby sme vedeli zistiť, či sa daný uzol v štruktúre nachádza, poslúži nám funkcia existsNode().

```
select existsNode(user, '//user/usernames') from users;
```

Návratová hodnota je typu boolean. Tzn. že dostaneme buď true alebo false. Teda existenciu uzla alebo nie. Každopádne využitie XPath je široké a umožňuje ľahké vyhľadavanie aj v zložitých XML dokumentoch, uložených v Oracle.

### 3.4 Využitie XML

V dobe smart telefónov a iných im podobných zariadením sa stretávame s XML bez toho aby sme o tom vedeli. Tak napríklad používatelia operačného systému Android, využívajú nevedomky tzv. Androidmanifest.xml. Tento súbor reprezentuje základné informácie o aplikácii v Android systéme. Tieto informácie je nutné aby poznal pred spustením kódu danej aplikácie. Ďalšie využitie XML je v tzv. RSS feeds. Operačný systém Mac OS, využíva XML na ukladanie dát v aplikáciách. Súborové formáty v Microsoft Office sú založené na XML. A mohol by som menovať ďalej. Využitie XML je širokospektrálne. Je to zapríčinené hlavne tým, že tento štandard je flexibilný a pomerne ľahko implementovateľný.

## 4 PHP

Písať niečo rozsiahle a predstavovať tento programovací jazyk je ako nosiť drevo do lesa. Všetci aspoň trochu zbehlí v programovacích jazykoch vedia, že skratka PHP neznamená Penový Hasiaci Prístroj, ale znamená PHP Hypertext preprocessor. Toto nič nehovoriace vysvetlenie rozvinem do ľudskejšej reči a vysvetlím, že ide o populárny serverový skriptovací jazyk. Využíva sa hlavne na vytváranie dynamických webových aplikácií. Je platformovo nezávislý, môže bežať na väčšine operačných systémoch ako je Windows , UNIX, Linux, Mac OS a iné. Nevyhnutnou vlastnosťou je konektivita na rôzne druhy databáz. Či už ide o pravdepodobne najviac rozšírenú MySQL, MSSQL, PostgreSQL, Oracle alebo SQLite. Veľkou výhodou je objektový prístup, ktorý je možný od PHP 5. PHP sa dá považovať ako alternatíva k ASP.NET.

### 4.1 OOP v PHP

Ako bolo vyššie spomenuté, podpora pre Objektovo-orientované programovanie prišla s verziou PHP 5. Tak ako v ostatných programovacích jazykoch, aj v PHP sú triedy, metódy, využíva sa dedičnosť, rôzne druhy viditeľnosti členských premenných atď. Porovnávať s nejakým iným jazykom je dosť komplikované. Z môjho pohľadu by som OOP v PHP prirovnal k C++.

Objektovo-orientované programovanie vyžaduje rozdielny spôsob uvažovania nad tým, ako vytvárať aplikácie. Objekty lepšie umožňujú pomocou kódu modelovať úlohy skutočného sveta, procesy a ďalšie myšlienky. Jednou z hlavných výhod OOP je jednoduchosť. Ďalšou výhodou je možnosť opätovného použitia kódu[9]. A ďalšie výhody.

### 4.2 PHP a Oracle

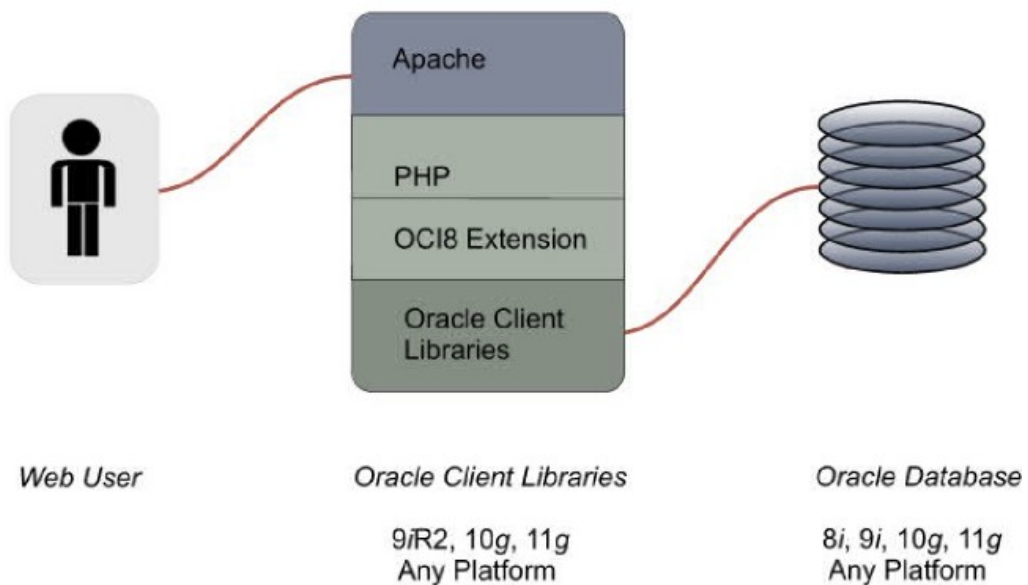
PHP má niekoľko rozšírení, ktoré dovoľujú pracovať s Oracle. Existujú rôzne abstraktné databázové knižnice písané v PHP prípadne v jazyku C. Medzi také najznámejšie PHP rozšírenia pre Oracle patria[9]:

- Oracle – zastaraná verzia, ktorá sa už nepoužíva
- OCI8 – najpoužívanejšie rozšírenie
- PDO – vychádza z PHP Data Objects. Kedže je to všeobecná knižnica pre prácu s databázami, tak každá databáza má svoje vlastné drivery, ktoré dopĺňajú jednotlivú funkcionality tej ktorej databázy.

- ODBC

Keďže v diplomovej práci využívam OCI8 rozšírenie, tak v nasledujúcich riadkoch to viacej rozpišem.

Ako je možné vidieť na obrázku 4, PHP využíva na spojenie s Oracle, Oracle Client Libraries. Tie poskytujú základnú konektivitu s databázou, ktorá môže byť jak lokálna, tak vzdialená[9].



Obrázok 4: Spojenie PHP s Oracle pomocou Oracle Client Libraries

Oracle má kompatibilitu medzi jednotlivými verziami. Tzn. ak PHP OCI8, ktoré je spojené s Oracle Database 10g pomocou klientských knižníc, potom PHP sa môže pripojiť na Oracle Database 8i, 9i, 10g alebo 11g.

Pre spojenie sa s Oracle slúži funkcia `oci_connect()`:

```
$connectDB = oci_connect($user, $pass, $dbname);

if (!$connectDB) {
    $e = oci_error();
    trigger_error(htmlentities($e['message'],
ENT_QUOTES), E_USER_ERROR);
}
```

Tak ako spojenie s databázou nadviážeme, je nutné ho aj korektne ukončiť. V tomto prípade sa využíva ďalšia funkciu ktorou je:

```
oci_close($conn);
```

#### 4.2.1 OCI Prepared statements

V tejto malej kapitolke sa chcem zamerať na využitie tzv. prepared statements. Voľne preložené ako predpripravené príkazy. O čo vlastne ide. Sú to parametrizované príkazy, ktoré môžu použiť opakovane rovnaký statement s vysokou efektivitou. Samotný proces pozostáva z dvoch krokov. Príprava dotazu a následné jeho vykonanie. V prvej fáze je dotaz spracovaný a poslaný na databázový server. Ten vykoná kontrolu syntax a inicializuje vnútorné zdroje servera pre neskoršie použitie. Príklad je uvedený nižšie.

```
$query = "select email from users where email=:mail";  
$stmt = oci_parse(Controller::connectToDB(), $query);
```

V druhej fáze je exekúcia. Počas nej, klient tzv. binduje hodnoty parametrov a posielajú ich serveru. Server vytvorí statement a naviaže hodnoty na jeho vykonanie, pričom používa vytvorené zdroje z predchádzajúceho kroku[12].

```
oci_bind_by_name($stmt, ":mail", $Email);  
oci_execute($stmt);
```

Ako je možné vidieť v prvom príklade, výhodou prepared statements je nahradzovanie bind parametra premennou, ktorá je volaná funkciou *oci\_bind\_by\_name*. Okrem nej, je treba zástupný znak (placeholder), ktorý v mojom príklade je : (dvojbodka).

Na záver je potrebné uvoľniť zdroje, ktoré boli priradené danému statement-u.

```
oci_free_statement($stmt);
```

Medzi výhody patrí väčšia bezpečnosť, majú lepší výkon ako aj práca s nimi je prijateľnejšia.

### 4.3 Využitie XML v Oracle a PHP

Tak Oracle ako PHP majú výborné XML možnosti, umožňujúce značné množstvo pre spracovávanie informácií. Všetky edície Oracle obsahujú XML DB, teda XML možnosti databázy Oracle[9]. Ako som už vyššie spomínal, XML môže byť uchovávané ako CLOB.

Jednou z dobrých vlastností XML DB je existencia relačných SQL tabuliek, ktoré môžu byť vykreslené v XML formáte. Hodnoty, ktoré dostaneme sú XML fragmenty, avšak nie sú plne formátované ako XML[9]. Na vyberanie dát v PHP slúži SimpleXML rozšírenie,

ktoré konvertuje vybrané dáta z databázy do PHP objektu. V nasledujúcom príklade je práve vytiahnutie a uloženie dát do PHP objektu.

```
$sxml_one[] = simplexml_load_string(iconv("windows-1250",  
"utf-8",$data_oneUser[0]));
```

Samotné vytiahnutie dát z tabuľky bolo vysvetlené vyššie. Možnosť, ako zobrazit' dáta je na ďalších riadkoch.

```
for($l_one=0; $l_one<=count($inc_oneUser-1); $l_one++) {  
    echo "sxml_one[$l_one]->artist";  
}
```

Pre zobrazenie konkrétnych uzlov z XML a ich hodnôt, je treba postupovať v cykle for, prípadne foreach, pričom k jednotlivým položkám sa pristupuje pomocou názvu uzlov.

## **II. PRAKTICKÁ ČÁST**

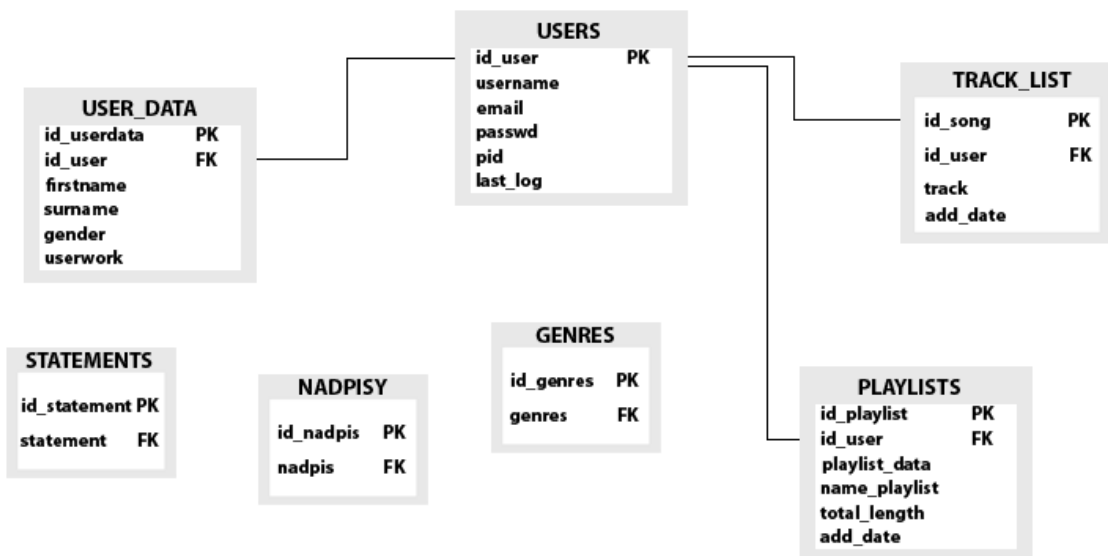
## 5 NÁVRH APLIKÁCIE

Návrh pozostával z databázovej a aplikačnej časti. V databázovej časti sa zameriam na celkový databázový návrh, vysvetlenie tabuliek. V druhej časti – aplikačný návrh, sa budem snažiť zamerať na vysvetlenie tried, ukladania skladieb, atď. V prípade, že je nutné, text je doplnený aj o obrázkovú časť. Okrem týchto základných vecí, sa budem snažiť vysvetliť dôvody práve môjho riešenia, prípadne navrhнем iný spôsob, akým by sa daná časť mohla riešiť.

Na začiatku, kým sa rozpíšem, chcem upozorniť na jednu skutočnosť. V texte budem občas používať niektoré programátorské slová z angličtiny, ktoré je docela ťažké preložiť do slovenčiny. Ak by som to aj spravil, výsledok a hlavne význam by nemusel presne korešpondovať s originálnym slovom. Môžem spomenúť prepared statements, bind parameter/hodnota a iné. V niektorých prípadoch sa môže vyskytnúť, hlavne pri databázových názvoch dotazov ako update-ovať a pod.

### 5.1 Databázový návrh

Databázový návrh pozostával zo správneho vytvorenia tabuliek. V mojom prípade ich je osem. Niektoré sú navzájom prepojené pomocou cudzích kľúčov. Niektoré sú využívané len samostatne, bez potreby viazania sa na ostatné tabuľky. Tieto tabuľky využívam spoločne s XML funkcionalitou na rôzne zobrazovania chybových hlášok, nadpisov a podobne. V nasledujúcom obrázku, sú zobrazené všetky tabuľky a prepojenia medzi nimi.



Obrázok 5: Databázové tabuľky - návrh

### 5.1.1 Tabuľka USERS

Táto tabuľka je určená hlavne pre účely prihlasovania do aplikácie. Pozostáva zo základných stĺpcov ako id\_user, username, atď. V priloženom skripte je celá štruktúra danej tabuľky.

```
create table users (
    id_user number not null,
    username varchar2(150),
    email varchar2(150),
    passwd varchar2(512),
    pid number,
    last_log date,
    constraint users_PK primary key (id_user)
);
```

### 5.1.2 Tabuľka USER\_DATA

Táto tabuľka rozširuje pôvodnú tabuľku USERS. Jedná sa viac menej o doplnkovú tabuľku, ktorá slúži na uchovávanie ďalších údajov o užívateľovi.

```
create table user_data (  
    id_userdata number not null,  
    id_user number not null,  
    firstname varchar2(150),  
    surname varchar2(200),  
    gender varchar2(5),  
    userWork varchar(200),  
    constraint user_data_PK primary key (id_userdata),  
    constraint user_data_FK foreign key (id_user) references  
    users  
);
```

### 5.1.3 Tabuľka TRACK\_LIST

Určená pre ukladanie užívateľských záznamov, resp. Ukladá skladby uložené užívateľom. V tejto tabuľke využívam funkcionálnosť XML v Oracle – XMLType stĺpec.

```
create table track_list (  
    id_song number not null,  
    id_user number not null,  
    track XMLType,  
    add_date date,  
    constraint track_list_PK primary key (id_song),  
    constraint track_list_FK foreign key (id_user)  
    references users  
)  
XMLType track STORE AS CLOB;
```

Ďalšia vec, ktorá stojí za povšimnutie je typ ukladania XML stĺpca. Tým typom je CLOB. Je to z dôvodu toho, že XML môže mať veľký objem, preto sa doporučuje ukladanie v tomto datovom type. Z praktického pohľadu však poviem, že v neskorších štádiách vytvárania praktickej časti, sa mi tento typ ukladania hodil. Hlavne pri vyberaní XML dát z tabuliek.

#### 5.1.4 Tabuľka STATEMENTS

Jedna z tých menších tabuliek. Tak ako v predchádzajúcej, aj v tejto využívam XMLType. Samotný význam tabuľky je taký, že jednotlivé záznamy využívam ako text pri rôznych chybových, upozornovacích alebo iných programových hlásení.

```
create table statements (  
    id_statement number not null,  
    statement xmltype,  
    constraint statements_PK primary key (id_statement)  
) XMLType statement store as clob;
```

#### 5.1.5 Tabuľka SESSIONS

Táto tabuľka je používaná k uchovávaní užívateľských (SESSION) údajov počas prihlásenia. Výhoda tejto tabuľky je v tom, že užívateľské prihlasovacie údaje sa ukladajú do databázy namiesto súborového systému servera. V prípade zdieľaného servera nie je táto možnosť ukladania sessions vhodná.

```
create table sessions (  
    id_session number not null,  
    session_data varchar2(512) not null,  
    date_expire date,  
    constraint session_id_PK primary key (id_session)  
);
```

Viacej o využití tejto tabuľky bude v ďalšej časti práce.

#### 5.1.6 Tabuľka PLAYLISTS

Keďže moja aplikácia umožňuje ukladanie playlistov, táto tabuľka je určená práve na to. Opäť som v nej využil XMLType.

```
create table playlists (  
    id_playlist number not null,  
    id_user number not null,
```

```
playlist_data XMLType,  
name_playlist varchar2(256),  
total_length varchar2(20),  
add_date date,  
constraint playlist_PK primary key (id_playlist),  
constraint playlist_FK foreign key (id_user) references  
users  
  
) XMLType playlist_data store as clob;
```

### 5.1.7 Tabuľka NADPISY

Dosť podobná s tabuľkou STATEMENTS. Avšak namiesto chybových či iných hlásení, túto tabuľku využívam na popisy názvu stránok v tagu title.

```
create table nadpisy (  
    id_nadpis number not null,  
    nadpis xmltype,  
    constraint nadpisy_PK primary key (id_nadpis)  
  
) XMLType nadpis store as clob;
```

### 5.1.8 Tabuľka GENRES

Posledná z tabuliek je určená pre žánre, ktoré sú zobrazované pri pridávaná jednotlivých skladieb. Túto tabuľku som zvolil preto, aby boli zjednotené žánre a aby užívatelia mali zjednodušené vkladanie.

```
create table genres (  
    id_genre number not null,  
    genre varchar2(100),  
    constraint genres_PK primary key (id_genre)  
  
);
```

## 5.2 Návrh aplikácie z programátorského hľadiska

Tento návrh pozostával z navrhnutia správnych tried, návrhu štruktúry webovej aplikácie ako aj grafické spracovanie aplikácie.

### 5.2.1 Triedy

Vzhľadom k tomu, že v mojom návrhu používam viacej tried, vysvetlím len tie základné. Sú to napríklad triedy na pripojenia sa k databáze, trieda na prihlásenie užívateľa, trieda na prácu s tzv. sessions, trieda pre generovanie token-u pri prihlasovaní a iné. Pri vytváraní som sa snažil dodržať zásadu, aby som jednotlivé metódy využíval opakovane, teda určitým spôsobom znova využitie kódu.

#### 5.2.1.1 *ClassConnectDB*

Cieľom tejto triedy je vytvoriť spojenie na databázu Oracle. Táto trieda je pomerne dôležitá pretože, prístup na databázu je v mojej aplikácii pomerne častý. Metóda *connect()* je bez argumentov. Tie sú zadané v tele triedy.

```
class ConnectDB {  
  
    private $user = "dp";  
    private $pass = "nejakeHeslo";  
    private $dbname = "dp";  
    private $charset = "";  
  
    function connect() {  
  
        $conn = oci_connect($this->user, $this->pass, $this->dbname);  
  
        if (!$conn) {  
            $e = oci_error();  
            trigger_error(htmlentities($e['message'], ENT_QUOTES), E_USER_ERROR);  
        } else {  
            header('location: /'. $cfg['domain']. '/404.php');  
        }  
        return $conn;  
    }  
}
```

### 5.2.1.2 *ClassController*

Táto trieda by mala spĺňať do určitej miery takú hlavnú triedu. Implementuje rozhranie *connection*. V tomto rozhraní sa nachádzajú dve metódy: *connectToDB* a *security*. Obe metódy majú za úlohu vytvárať inštanciu triedy *Security* a *ConnectDB*. Následne v skriptoch tieto metódy volám pomocou špeciálneho mena *parent* a tzv. scope resolution operator (::).

### 5.2.1.3 *ClassAuth*

Ako názov napovedá, trieda sa venuje prihlasovaniu užívateľov do aplikácie. Verejná funkcia *getUserData(userEmail)* vyberá z databázy (tabuľka *USERS*) užívateľa podľa zadaného emailu. (Podrobnejší popis prihlasovania bude v kapitole 5.2.3) Metóda *getLogin(formPasswd)* vracia informáciu o porovnaní dvoch hashov. Toto porovnávanie využíva metódu z triedy *Security*. Posledná metóda *Authorization(UserDataArr)* spája dohromady viacero metód. Vyberá dáta z databázy, číta dáta z triedy *SessionManager*, deserializuje dáta, ktoré nakoniec porovná s dátami prichádzajúcimi z prihlasovacieho formulára.

### 5.2.1.4 *ClassSecurity*

Niekoľko môže si pomyslieť, že aký veľkorysý názov, ale na druhú stranu, táto trieda vytvára hash hodnoty, pracuje s nimi aj ich dokáže porovnávať. Pomocou konšuktora triedy preberám potrebné premenné. Privátna metóda *setHash()* je tzv. setter, ktorý využíva *mencrypt* modul. Generuje tzv. soľ (salt), ktorú následne pridáva k užívateľskému heslu, ktoré je následne PHP funkciou *mhash()* zahashované. Ako hashovací algoritmus využívam *MHASH\_SHA256*. Na druhú stranu pomocou getter *getHash()* pristupujem k privátnej metóde vyššie spomínanej. Posledná metóda v tejto triede je *validateHash(formPass, dbHash)*. Princíp je založený na porovnaní odtlačku hash reťazca z databázy s odtlačkom hash reťazca, ktorý je po kontrole a pridaní soleného reťazca hashovaný do podoby vhodnej na porovnanie. Lepšie ako tisíc slov je názorná ukážka.

```
public function validateHash($formPass, $rightHash) {  
  
    $saltValid = substr($rightHash, 0, 64);  
    $validHash = substr($rightHash, 64, 64);
```

```

        $testedHash = bin2hex(mhash(MHASH_SHA256,
    $saltValid.$formPass));

        return $testedHash === $validHash;
    }

```

Ako som už v predchádzajúcej podkapitole spomenul, túto metódu využívam v triede `ClassAuth`.

Na tomto mieste by som ešte sa vrátil k spomínanej metóde `setHash()`. Hlavným dôvodom, prečo treba hashovať heslá, je prostý. Tým je bezpečnosť. Ak sú heslá v tzv. plain texte, tak je jasné, čo nastane. Hashovanie hesiel do určitej miery toto odstraňuje. Avšak nie úplne. Na príklade vysvetlím. Povedzme, že budem mať heslo *pivo*. V prípade, že toto heslo zahashujem (napr. sha1), dostanem tento výsledný odtlačok: `e7f980e22bc5150a000c9dfaf3155af3e7e04487`. Ak druhý užívateľ obľubuje pivo tak ako ja, a napadne ho dať si rovnaké heslo, výsledok sa dá predpokladať. Budú mať rovnaké hash odtlačky! V takomto prípade sa doporučuje solenie hesiel alebo tzv. salting hash. Tento prístup využívam práve v spomínanej metóde `setHash()`.

```

private function setHash() {

    $td = mcrypt_module_open('rijndael-256', '', 'ofb', '');
    $salt =
bin2hex(mcrypt_create_iv(mcrypt_enc_get_iv_size($td),
MCRYPT_RAND));
    $hash = bin2hex(mhash(MHASH_SHA256, $salt.$this-
>user_passwd));
    $this->settedHash = $salt.$hash;

}

```

Na tento účel využívam `mcrypt_module_open`. Ten inicializuje daný modul. Soľ vytvorím pomocou `mcrypt_create_iv`, pričom využívam inicializáciu z predchádzajúceho kroku. Výhodou je generovanie náhodnej postupnosti, ktorá je pre každý raz iná. Na hash používam funkciu `mhash` s algoritmom sha256. Výsledný hash, je spojenie reťazca soľ a reťazca, ktorý vrátila funkcia `mhash()`.

### 5.2.1.5 *ClassSessions*

Na vytvorenie tejto triedy som sa nechal inšpirovať článkom `Storing PHP Sessions in a database` od Rich Smith[10]. Základným princípom je využitie PHP funkcie `session_set_save_handler`, ktorá nastavuje session funkcie na užívateľskej úrovni. Tie sú

používané pre ukládanie ako aj pre výber dát priradených k danej session. Je šesť metód: *open()*, *close()*, *read(id)*, *write(id, data)*, *destroy(id)* a *gc()*. Intuitívne názvy naznačujú, že je nepotrebné nejako zvlášť rozoberať jednotlivú funkčnosť. Avšak zastavím sa pri metóde *write*. Táto metóda by nebola ničím zvláštna keby bola implementovaná v MySQL. Lenže v Oracle prostredí mi príde veľa vecí zložitejších. To čo by sa napísalo na dva riadky, tu sa to píše na osem. Konkrétne mám na mysli príkaz, ľudovo nazvem „upsert“. Teda spojenie *update* a *insert*. V Oracle sa na tento účel využíva príkaz „merge“.

```
$query_sessionRead = "
    merge into sessions
        using dual
        on (sessions.id_session =:sessionId)
        when matched then
            update set sessions.session_data =:sessionData

        when not matched then
            insert (sessions.id_session,
                sessions.session_data, sessions.date_expire)

            values (:sessionId, :sessionData,
                :dateExpire)";
```

Napriek dlhšiemu kódu, je však prehľadný a pomerne zrozumiteľný. V tejto časti sa ešte zastavím pri malej metóde *gc()*. Je to taký garbage collector pre sessions (preto aj skratka *gc*). Stará sa o vymazanie prebytočných databázových záznamov, ktoré nespĺňajú podmienku, že dátum expirácie je menší ako *sysdate-1*.

### 5.2.1.6 ClassStatements

Veľmi jednoduchá trieda ktorá aplikuje statické triedy *message(messageType, messageVar)* a *titles(title)*. Statické z dôvodu jednoduchosti celej triedy. Nejde totiž o extra dôležitú. Preto som zvolil variantu so statickými metódami, ktoré nemusia vytvárať inštanciu triedy.

```
$stmt = new Statements;
```

Stačí keď sa využije už spomínaný scope resolution operator. Na krátkom príklade je opäť vidieť použitie na metóde *message*.

```
Statements::message('error', 'badLogin');
```

Keďže táto trieda pristupuje k databázovej tabuľke, musí dediť z triedy Controller. Využíva metódu *parent::connectToDB()*.

Uvedomujem si, že tak ako táto trieda, tak aj tabuľka STATEMENTS by sa dali nahradiť. Napadá má spôsob pomocou obyčajného XML uloženého na serveri a nie v databázy, prípadne cez konfiguračné súbory a pod. Avšak mojim cieľom bolo využitie možností Oracle čo najviac a ako mi to možnosti dovoľia.

### 5.2.1.7 *ClassTokenArray*

Keď som rozmýšľal, ako by som zabezpečil prihlasovací formulár, našiel som triedu TokenGrid od André Liechti[11]. Táto trieda generuje tzv. Grid kartu, ktorú poznáme pri bankových transakciách. Pôvodnú myšlienku som troška zmenšil a vytvoril triedu ktorej názov môžete vidieť vyššie.

Ide o vygenerovanie poľa rôznej dĺžky, v ktorom využívam tokenSalt a nastavujem aj dĺžku požadovaného reťazca. Všetky tieto tri premenné sú zadávané do konštruktora triedy *TokenArray(arraySize, tokenLength, tokenSalt)*. Výsledné reťazce, ktoré naplnia pole, sú poskladané z veľkých písmen abecedy a číslíc 0-9.

Metóda *getSingleToken(pos, arrID)* generuje jeden token. Tak napr. ak v konštruktoře mám hodnotu *tokenLength = 5*, výsledný token môže vyzerat' nasledovne: J1KHA. O naplnenie celého poľa o dĺžke *arraySize* sa stará metóda *getTokenArray()*. Nakoniec je ku každému tokenu priradený kľúč poľa (zvolil som si písmeno A). Teda A0 => J9L3S až A(*arraySize-1*) =>P1V0O. Následná implementácia v prihlasovacom formulári je nasledovná. Do skrytého poľa (tzv. hidden field) je umiestnená informácia o aktuálnom náhodne vybranom prvku z tokenArray (napr. A10). Tejto hodnote zodpovedá hodnota v tokenArray, ktorá je po prihlásení kontrolovaná metódou *validToken(randPos)*. V prípade, že sa s formulárom nevhodne manipulovalo a hodnoty sa nezhodujú, záhľási aplikácia chybu. Využitím tokenSalt sa miera náhodnosti zvyšuje. Výhodou tejto triedy je bezpečnejšia autentizácia užívateľov.

### 5.2.1.8 *ClassOracle*

Poslednú triedu, ktorú chcem spomenúť je trieda *ClassOracle*. Dôvod prečo som túto triedu vytvoril je nasledovný. Za prvé, potreboval som zisťovať a inkrementovať hodnoty stĺpcov v tabuľkách. Za druhé zisťovať, či sa daný užívateľ nachádza v databázy pri registrácii a ako posledný dôvod bolo vkladanie nového užívateľa.

Oracle má pomerne zložité vytváranie tzv. *auto\_increment* vlastnosti oproti napr. MySQL. Táto vlastnosť mi počas tvorby aplikácie dosť chýbala, preto som sa rozhodol, že obídem klasický prístup pomocou kurzora v Oracle a vytvoril som triedu s príznačným názvom *autoIncrement(column, table)*. Základom je jednoduchý *select*, ktorý vyberá maximálnu hodnotu zo stĺpca (argument *column*) z ktorejkoľvek tabuľky (argument *table*). K hodnote, ktorú dostanem, pripočítam jednotku a dostávam nasledovné číslo v poradí.

V metóde *checkRegisteredUser(email)* kontrolujem pri registrácii, či sa už daný užívateľ neregistroval. Funkcia vracia príznak, podľa ktorého viem, či email sa nachádza v databázy alebo nie.

*AddNewUser(autoInc, userName, email, passwd, pid, lastLog)* je metóda na vkladanie nového užívateľa. Preberá šesť parametrov, ktoré sú potrebné pri bindovaní hodnôt pri vkladaní.

Určite by som mohol pokračovať aj s ostatnými triedami, ale myslím si, že tie ktoré som menoval dostatočne oboznamujú čitateľa s návrhom aplikácie.

## 5.2.2 Registrácia

Preto, aby sa užívateľ mohol prihlásiť a využívať aplikáciu, je nutné sa najskôr registrovať. Samotný proces netrvá veľmi dlho. Na obrázku č. 6 je vidieť vyplnený registračný formulár. Hlavnou myšlienkou kontroly pri registrácii, je využitie JQuery doplnku, ktorý sa stará o kontrolovanie jednotlivých polí vo formulári. Podľa zadaných pravidiel je napr. skontrolovaný email, pomocou regulérneho výrazu. Taktiež sa porovnávajú hodnoty oboch hesiel. Aplikácia je nastavená tak, že v prípade slabého hesla neumožní registráciu. Užívateľ o vhodnosti, či krátkom hesle je automaticky informovaný na postrannom indikátore, doplnené o slovné vyjadrenie kvality hesla (Veľmi slabé, Slabé, Ujde to, Silné). Čo sa týka samotného procesu registrácie, o to sa stará PHP spoločne s Oracle. V prvom rade, je potrebné ošetriť všetky formulárové prvky pomocou funkcie *htmlentities()*. Tá

prevádza všetky znaky na tzv. html entity. V ďalšej podmienke testujem pomocou regulárneho výrazu emailovú adresu. (V prípade, že by Javascript na užívateľskom prehliadači bol vypnutý).

```
$valid_mail = (preg_match("[a-zA-Z0-9_.-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-]+.$", $email));
```

Obrázok 6: Registračný formulár – ukážka

Nutnou podmienkou v procese registrácie je kontrola, či sa zadaný email nachádza v databáze. Na tento účel slúži metóda `checkRegisteredUser($email)` z triedy *ClassOracle*. Podľa vráteného príznaku sa pokračuje buď na pridanie nového užívateľa, alebo sa zobrazí upozornenie o existencii zadaného emailu. V prvom prípade sa ďalej využije metóda `autoIncrement('id_user', 'users')` a metóda na pridanie nového užívateľa do databázy:

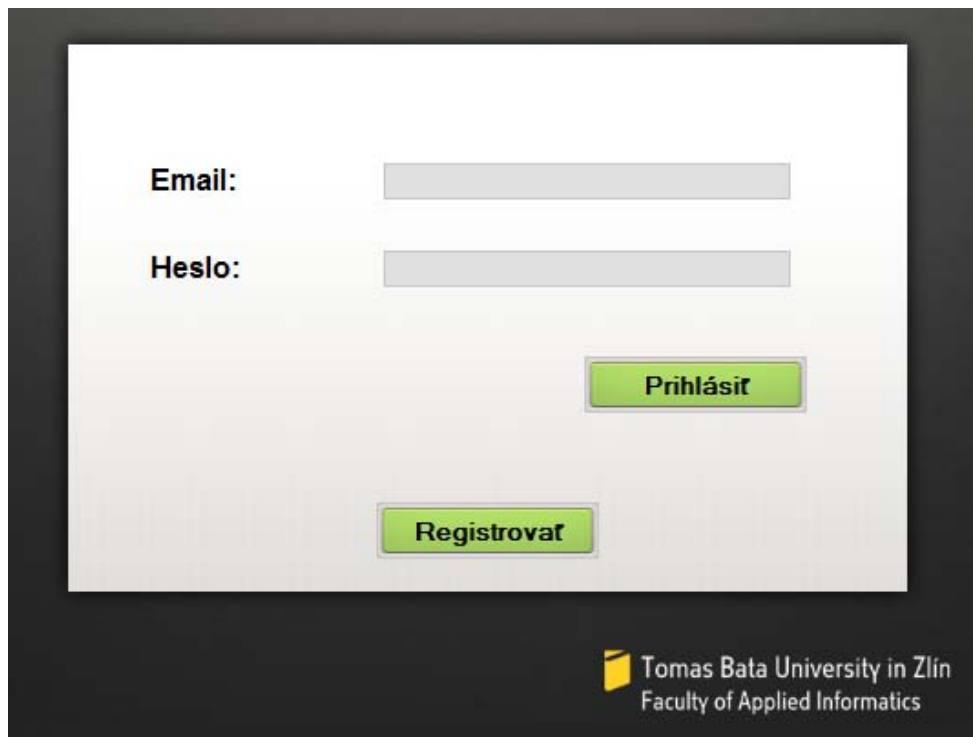
```
addNewUser($autoInc, $userName, $email, $getHash->getHash(),
    $pid, date("d.m.Y"));
```

Vysvetľovať jednotlivé argumenty je zbytočné, pretože som sa snažil názvy premenných menovať tak, aby boli čo najzrozumiteľnejšie. Zastavím sa však pri treťom argumente. Ten

volá metódu `getHash()` z triedy *ClassSecurity*. Princíp metódy je vysvetlený v kapitole 5.2.1.4

### 5.2.3 Login

S touto časťou sa užívateľ stretne ako prvé. Pre prihlásenie je nutné sa však najskôr zaregistrovať. Postup registrácie som popísal v predchádzajúcej kapitole.

The image shows a login form with a white background and a dark border. It contains two input fields: 'Email:' and 'Heslo:'. Below the 'Heslo:' field is a green button labeled 'Prihlásiť'. Below the 'Prihlásiť' button is another green button labeled 'Registrovať'. In the bottom right corner, there is a logo for 'Tomas Bata University in Zlín Faculty of Applied Informatics'.

Obrázok 7: Formulár pre prihlásenie

Po správnom zadaní emailu a užívateľského hesla sa presmeruje na hlavnú stránku – `home.php`. Predtým však musí prebehnúť proces prihlásenia. Ako prvé sa skontroluje token hodnota s hodnotu v prihlasovacom formulári. Aj v tomto kroku ošetrujem formulárové prvky už spomínanou funkciou *htmlentities()*. V prípade zlého token-u, je beh skriptu presmerovaný na prihlasovaciu stránku s vhodným upozornením. Ďalej testujem, či užívateľ zadal obe hodnoty do formulára. Pomocou *switch* riadim správanie sa aplikácie pri nezadaní potrebných prihlasovacích údajov.

```
switch ($_POST) {  
    case empty($username) && empty($password):  
        $_SESSION['logNamePass'] = 'logNamePass';  
        header('location: /'. $cfg['domain']. '/index.php');  
        break;
```

```

    case empty($username):
        $_SESSION['logName'] = 'logName';
        header('location: /'.$cfg['domain'].'/index.php');
        break;
    case empty($password):
        $_SESSION['logPass'] = 'logPass';
        header('location: /'.$cfg['domain'].'/index.php');
        break;
    default;
}

```

Ak je všetko zadané, prichádza na rad trieda *ClassAuth*. V nej sa metódou *getUserData* načítajú dáta z databázy a porovnávajú so zadanými hodnotami z formulára. Ak všetko prebehne v poriadku, vygeneruje sa dočasný, ja ho nazývam „login hash“, ktorý je zložený z užívateľských údajov plus pridaný náhodný hash reťazec. Tým sa docieli, že rovnaký užívateľ má zakaždým rozdielny „login hash“. To znamená určitú náhodnosť v prihlasovaní. V závere procesu využívam triedu *ClassSessionManager* a jej metódu *write()*, na zapísanie session hodnoty do databázovej tabuľky SESSIONS.

V ďalších podstránkach na kontrolu prihlásenia používam súbor *protected.inc.php*. Princíp je založený na porovnávaní údajov, ktoré sa porovnávajú s údajmi z databázy SESSIONS (login hash).

#### 5.2.4 Logout (Odhlásenie)

Opačným postupom oproti prihláseniu je odhlásenie (logout). Vykonávajú sa v ňom dve akcie. Prvou z nich je update dátum posledného prihlásenia, ktorý sa vloží do tabuľky USERS ku konkrétnemu užívateľovi. Druhou akciou je vymazanie session záznamu v tabuľke SESSIONS. Na tento účel volám metódu *destroy()*:

```
$sessDestroy->destroy($user_lastLog[0]);
```

a metódu *gc()*:

```
$sessDestroy->gc();
```

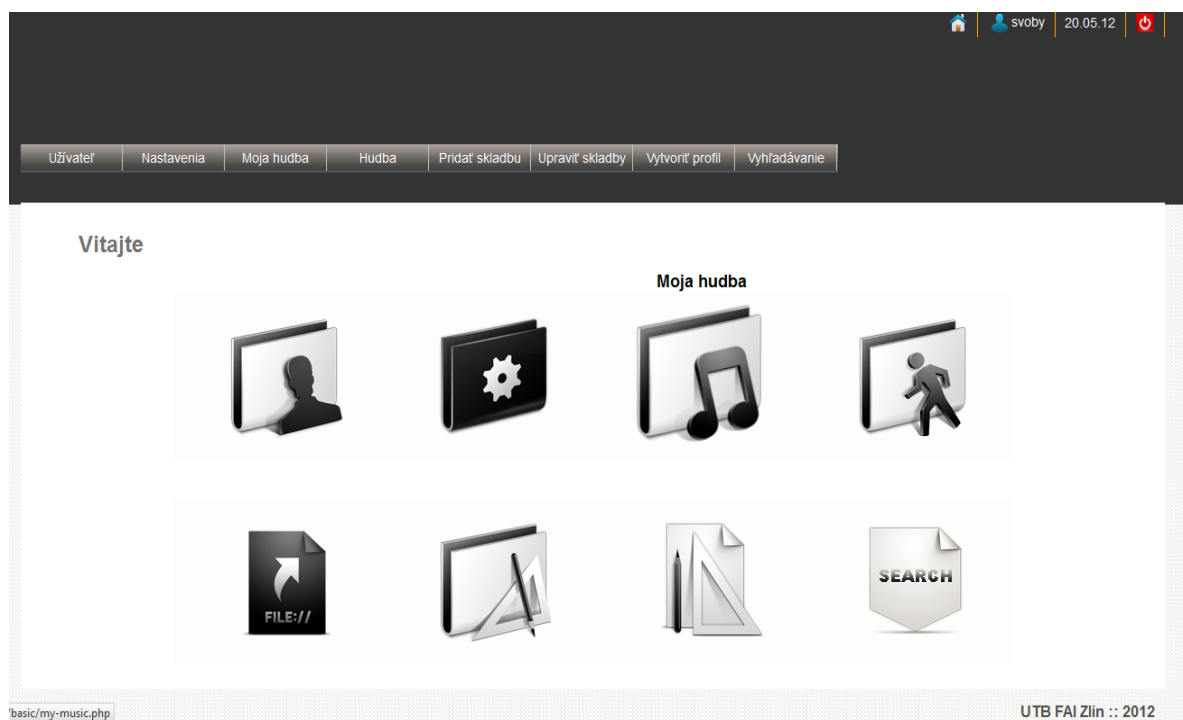
Tá ma za úlohu vymazať všetky nepotrebné záznamy.

#### 5.2.5 Užívateľské rozhranie

Na obrázku č. 8 je možné vidieť výrez po prihlásení. Užívateľ má možnosť si vybrať z ôsmich kategórií:

- Užívateľ

- Nastavenia
- Moja hudba
- Hudba
- Pridať skladbu
- Upraviť skladby
- Vytvoriť profil
- Vyhľadávanie



Obrázok 8: užívateľské rozhranie

V pravom hornom rohu je informačná oblasť, kde ako je možné vidieť prihláseného užívateľa, dátum posledného prihlásenia a možnosť odhlásenia. Horizontálne menu by sa mohlo zdať redundantné, ale v konečnom dôsledku v jednotlivých podstránkach umožňuje rýchlu navigáciu.

### 5.2.6 Vyberanie XML údajov z Oracle

Trúfnem si povedať, že je to jedna z najdôležitejších úloh, ktorú je treba popísať. Bez vhodného a správneho výberu by som nemohol zobrazovať viac menej žiadne údaje typu *XMLType*, ktoré sú uložené v databázy.

Vzhľadom k tomu, že využívam viacero XML tabuliek, princíp vysvetlím len na tabuľke *TRACK\_LIST*. Ďalšie využitie v iných XML tabuľkách je čistou analógiu tohto postupu.

Základom je správne zostavenie SQL dotazu. V teoretickej časti som sa venoval tzv. XPath, takže teraz budem mať už pripravenú pôdu. Teda pochopenie nebude vôbec náročné.

```
$query_oneUser = "select extract(track, '//track') from
track_list where id_user=:idUser";
```

V dotaze sa pokúšam vytiahnuť XML dáta zo stĺpca (*track*), pričom využívam cesty *//track*. Tá mi určuje, ktoré elementy budú vytiahnuté zo zdrojového záznamu. Potrebnou podmienkou je definovanie *id\_user*, pretože inak by sa extrahovali všetky hodnoty z danej tabuľky. (Vo vyfiltrovaní všetkých skladiel sa táto podmienka neudáva!).

```
$stmt_oneUser = oci_parse($connect, $query_oneUser);
oci_bind_by_name($stmt_oneUser, ":idUser", $idUser);
oci_execute($stmt_oneUser);
```

```
while($data_oneUser = oci_fetch_array($stmt_oneUser,
OCI_NUM)) {
    $sxml_one[] = simplexml_load_string(iconv("windows-
1250", "utf-8", $data_oneUser[0]));
    $inc_oneUser++;
}
```

Daný dotaz parsujem, naviažem potrebnú hodnotu na bind parameter (ak treba) a následne vykonám. Funkcia *oci\_fetch\_array()* uloží do danej premennej vrátený obsah z databázy. Aby sa lepšie pracovalo s danými hodnotami, v cykle *while* prechádzam vrátené hodnoty a ukladám do novej premennej typu pole. Zároveň využívam funkciu *simplexml\_load\_string*. Výsledkom je tzv. SimpleXMLObject. Čo som tým dosiahol? Dosiahol som tým rovnakú štruktúru XML aká je uložená v databázy. Výhoda tohto prístupu spočíva v tom, že mi umožňuje v ďalšom kroku elegantne pristupovať k jednotlivým elementom. Tie vykresľujem do tabuľky. Krátku ukážku je možné vidieť na obrázku č. 9

## Moja hudba

Č.	Interpret	Názov	Dĺžka	Žáner	Typ	Užívateľ	Dátum
1	Eva a Vašek	Orchidej	00:04:22	Alternative	cooldown	svoby	18.05.12
2	Florence and the machine	Drumming song	00:03:44	Pop	rovina	svoby	14.05.12
3	Survivor	Burning heart	00:03:52	Rock	kopec	svoby	23.05.12
4	AC/DC	Nieco	00:04:59	Rock	kopec	svoby	14.05.12
5	Faithless	Insomnia	00:04:39	Electronic	rovina	svoby	18.05.12

Obrázok 9: Tabuľka skladieb konkrétneho užívateľa

Ako som už písal, výhodou uloženia do poľa je tá, že jednotlivé položky môžem prechádzať rôznymi funkciami (for, foreach, while,...). Keďže si uchovávam hodnotu `$inc_oneUser++`; môžem využiť cyklus *for*. Tento prístup by sa mohol nahradiť aj vyššie menovanými cyklami a použitím funkcie *count(arr)*, avšak som si vybral tento štýl, pretože mi príde najvýhodnejší. Časť je vidieť na tomto kúsku kódu:

```
$pekne_cislo = 1;
for($l_one=0; $l_one<=$inc_oneUser-1; $l_one++) {
    echo "<tr>";
    echo "<td>" . $pekne_cislo . "</td>";
    echo "<td class='artist'>" . iconv("utf-8", "windows-
1250", $sxl_one[$l_one]->artist) . "</td>";
    echo "<td class='title'>" . iconv("utf-8", "windows-
1250", $sxl_one[$l_one]->titleTrack) . "</td>";
    echo "<td class='add_date'>" . $sxl_one[$l_one]-
>duration . "</td>";
    echo "<td class='add_date'>" . iconv("utf-8", "windows-
1250", $sxl_one[$l_one]->genre) . "</td>";
    echo "<td class='zaner'>" . iconv("utf-8", "windows-
1250", $sxl_one[$l_one]->type) . "</td>";
    echo "<td class='user'>" . iconv("utf-8", "windows-
1250", $sxl_one[$l_one]->username) . "</td>";
    echo "<td class='add_date'>" . $restData_one[$l_one][2]
. "</td>";    echo "</tr>";
$pekne_cislo++;
}
```

Tento prístup ako som už spomenul, využívam všade tam, kde vyťahujem XML dáta z databázy (hudba všetkých užívateľov, upravovanie skladieb, vytvorenie profilu, vyhľadávanie).

V závere je dobrým zvykom a nutnosťou uvoľniť zdroje využité pri týchto prepared statements.

```
oci_free_statement($stmt_getRestData_one);  
oci_free_statement($stmt_oneUser);
```

### 5.2.7 Prehľad a nastavenia účtu

Mojím cieľom, okrem iného, bola aj užívateľská prívetivosť a do značnej miery aj prispôsobivosť aplikácie. Snažil som sa to docieľiť základnými nastaveniami účtu. Zmena hesla považujem za samozrejmé, prídavné položky ako meno, priezvisko, pohlavie a zamestnanie považujem za dobrovoľné.

V týchto čiastkových skriptoch, sa naplno prejaví vlastnosť objektového prístupu. Tou je znovu využitie kódu. Tak napríklad v nastavení nového hesla. Užívateľ musí zadať najskôr staré heslo. To je pomocou metódy *validateHash()* z triedy *ClassSecurity*, porovnané s heslom z formulára. V prípade zhody, sa vykoná uloženie nového hesla do databázy, prípadne uloženie ostatných údajov.

Čo sa týka prehľadu užívateľského konta, za zmienku stoja dve veci. Dotaz spájajúci dve tabuľky (USERS a USER\_INFO a využitie agregáčnej funkcie COUNT() v oboch dotazoch.

```
select users.username, user_data.firstname,  
user_data.surname, user_data.gender, user_data.userwork,  
count(track_list.id_user) as track_nr  
  
from users join user_data  
on users.id_user = user_data.id_user  
  
join track_list  
on users.id_user = track_list.id_user  
  
where users.id_user =:id_user  
  
group by users.username, user_data.firstname,  
user_data.surname, user_data.gender, user_data.userwork'
```

Z vyššie uvedeného dotazu jasne vyplýva myšlienka využitia *JOIN* ako aj funkcie *COUNT()*. V tomto prípade chcem dostať celkový počet skladieb, ktoré sa viažu k danému užívateľovi. Druhým kratším dotazom je vrátenie počtu uložených playlistov.

```
select count(playlists.id_user) as cntPlaylist
  from playlists
    join users on playlists.id_user = users.id_user
  where users.id_user = :id_user
```

V prípade, keď vrátená hodnota je NULL (žiadny playlist!), tak je táto hodnota zobrazená pričom sa nezobrazí odkaz, na stránku s uloženými playlistami. Ten sa zobrazí až vtedy, keď vrátená hodnota je väčšia, prípadne rovná jednej. Bližšie o playlistoch budem hovoriť v kapitole 5.2.10 Vytvorenie a uloženie profilu

<b>Username</b>	svoby
<b>Meno</b>	Lukáš
<b>Priezvisko</b>	Svoboda
<b>Pohlavie</b>	muž
<b>Zamestnanie</b>	študent VŠ
<b>Počet skladieb</b>	5

Moje playlisty (0)

Obrázok 10: Súhrn užívateľských informácií

Username	svoby
Staré heslo	<input type="text"/>
Nové heslo	<input type="text"/>
Meno	Lukáš
Priezvisko	Svoboda
Pohlavie	muž ▾
Zamestnanie	študent VŠ

Obrázok 11: Formulár pre užívateľské úpravy

### 5.2.8 Pridávanie skladieb

Opäť jedna z dôležitých častí celej aplikácie. Základom je vytvorenie XML elementu `<track>`, ktorý pozostáva z ďalších tzv. *child* elementov. XML štruktúra jednej takejto skladby vyzerá nasledovne:

```
<track idTrack="1" idUser='1'>
  <artist>Emilia Torriny</artist>
  <titleTrack>Jungle drum</titleTrack>
  <duration>00:02:00</duration>
  <genre>alternative</genre>
  <type>rovina</type>
  <username>svoby</username>
</track>
```

Pre úplnosť ešte doplním rodičovským (root) element, ktorým je:

```
<songs>
.
.
.
</songs>
```

Z tohto návrhu vyplývajú viaceré závery. Užívateľ zadáva len názov interpreta, názov skladby, dĺžku, žáner a typ. Posledný element *username* je doplnený automaticky, v závislosti na prihlásenom užívateľovi. Miernou nevýhodou je podmienka, že všetky polia formulára, musia byť vyplnené. Je to z jednoduchého dôvodu. Pri nevyplnení všetkých polí, by vyhľadávanie v skladbách malo za následok nezobrazovanie skladieb, ktoré by inak zodpovedali zadaným kritériám. Na zjednotenie hudobných žánrov, z ktorých si užívatelia môžu vybrať (a zároveň vyhľadávať podľa žánru) je určená tabuľka GENRES, ktorú som vysvetľoval v kapitole: 5.1.8 Tabuľka GENRES. Na začiatku som plánoval širokospektrálny zoznam žánrov, ale nakoniec som tento zoznam skrátil na také žánrové minimum:

- Alternative
- Blues
- Classical
- Country
- Dance

- Electronic
- Hip – hop/rap
- Latino
- New Age
- Pop
- RnB
- Rock
- Ine

Dost' bolo planých rečí, poďme sa pozrieť na samotné vkladanie. To je docela rozsiahle. Prvým problémom, je správny formát vkladania dĺžky. Pomocou JQuery, som spravil malú nápovedu, ktorá vysvetľuje, aký formát majú užívatelia vložiť. Tak ako to bolo pri prihlásení, aj v tomto prípade musím kontrolovať zadanie všetkých polí. Robím to opäť pomocou programovej konštrukcie *switch*. Vzhľadom k tomu, že je tam veľa kombinácií, ktoré nemožem opomenúť, nebudem celý kód uvádzať.

Po kontrole vstupných údajov, nastáva fáza vytvorenia XML dokumentu. Ten vytváram pomocou triedy *DOMDocument*.

```
$newTag = new DOMDocument('1.0');  
$newTag->preserveWhiteSpace = false;  
$newTag->formatOutput = true;
```

Vytvorenie celej XML štruktúry je v Prílohe P I: Štruktúra XML. Základom je vytvorenie elementov a priradenie hodnôt, pochádzajúcich z polí formulára. Nakoniec je celá štruktúra uložená ako XML, využívajúc funkciu: *saveXML()*.

Keďže pracujem a musím vkladať do databázovej tabuľky okrem iných hodnôt aj XML hodnotu, musím k tomu pristupovať iným spôsobom, ako pri bežnom vkladaní, výbere alebo aktualizácii údajov. Tu naplno využijem pridaný parameter *XMLType track STORE AS CLOB*, definovaný pri vytváraní tabuliek. To však prináša mierne skomplikovanie. Musel som vytvoriť nový, prázdny LOB deskriptor, ktorý alokuje zdroje určené pre LOB.

```
$lob = oci_new_descriptor($cnt->connect(), OCI_D_LOB);
```

SQL dotaz musel byť taktiež upravený, resp. bind parameter sa odvolával na *XMLType(:clob)*. Tak isto aj bind funkcia musí mať upravený tvar, umožňujúci naviazať

hodnotu premennej z vytvoreného LOB deskriptora, ktorému som ešte nepriradil žiadnu hodnotu.

```
oci_bind_by_name($stmt_sessionRead, ':clob', $lob, -1,  
OCI_B_CLOB);  
$lob->writeTemporary($newTag->saveXML());
```

Vyššie je ukázaný spôsob dočasného naplnenia LOB deskriptora. Následne sa postupuje obdobne ako pri iných prepared statements. Pre správne ukončenie je ešte potrebné uzavrieť LOB deskriptor.

### 5.2.9 Úprava skladieb

V tejto možnosti je okrem úpravy, možné aj vymazanie skladieb. Princíp update-u skladby je rovnaký ako vyššie spomenuté vkladanie. Rozdiel je v použítom databázovom dotaze.

```
$stmt_editTrack = oci_parse($cntdb->connect(), "update  
track_list set track=XMLType(:clob), add_date=:addDate where  
id_song=:idt and id_user=:iduser");
```

Ako je možné vidieť, update-ovanie je závislé na dvoch podmienkach. Tými sú *id\_user* a *id\_song*. Tak sa zaistí jednota pri úprave ako aj pri vymazávaní. Pri odstraňovaní je to výhodné z dôvodu, aby nedochádzalo k vymazávaniu skladieb, ktoré daný užívateľ „nevlastní“. Takto môže zmazať maximálne len svoje skladby. Nasledujúci dotaz je práve na odstraňovanie zmienených skladieb.

```
$query_delTrack = "delete from track_list where id_song  
=:idTr and id_user=:id_user";
```

### 5.2.10 Vytvorenie a uloženie profilu

Keď som premýšľal, ako čo najlepšie vytvoriť túto sekciu, spomenul som si opäť na moju bakalársku prácu. V nej som využil jCart modul[13]. Ten istý modul, s určitými zmenami som sa rozhodol použiť aj teraz. O čo vlastne ide. jCart je bezplatný nákupný košík, založený na Ajax-e. Áno, čítate správne – nákupný košík. Môže to znieť akokoľvek smiešne, dokonca zarážajúco, avšak tento modul dokonale spĺňa to, čo potrebujem. Pridávanie skladieb bez nutnosti obnovenia stránky (o to sa stará práve Ajax). Aby tento „košík“ bol podľa mojich predstáv, musel som upraviť niektoré časti. V prvom rade, som všetky popisy premenoval v konfiguračnom súbore. Ďalším krokom bolo pridanie potrebných premenných:

```
$item['type'] = $this->itemtype[$tmp_item];  
$item['subtotal'] = $item['price'];
```

Tie boli nevyhnutné, pri ukladaní typu skladby a dĺžky playlist-a. Jedným z problémov, bola konverzia ceny (napr. 23.99) na formát času, resp. dĺžky (napr. 00:06:12) pri sčítavaní celkového trvania playlist-u. Vyriešil som to nasledovne. Na začiatku som definoval nulový čas a funkciou *explode()* rozdelil tento čas na pole o troch prvkoch (za delimiter znak som použil dvojbodku). Následne tento čas som funkciou *mktime()*, prekonvertoval na unix-ový čas. Tieto hodnoty som následne použil pri inicializovaní premennej na uchovávanie celkového času.

```
$this->total = date("H:i:s", strtotime("+". $pomTotal[0] . "  
hours +" . $pomTotal[1] . " minutes +" . $pomTotal[2] . "  
seconds", $tot));
```

Využil som tam klasickú funkciu na prácu s časom v PHP – *date()* a *strtotime()*. Keďže celkový čas je vo vnútri internej funkcie *\_update\_total()*, je treba celkový čas aktualizovať v cykle *foreach*, po každom pridaní alebo odobraní skladby. V tele cyklu sa využíva rovnakého princípu, kde sa k buď začiatočnému času (ak sa pridá prvá skladba) alebo k už existujúcemu nenulovému času pridá ďalšia skladba. Funkcia nakoniec vracia automaticky aktuálne spočítaný čas v požadovanom formáte. Ďalšie úpravy čisto grafického charakteru, prípadne malé úpravy v konfiguračnom súbore.

JCart je napísaný objektovo. Z toho vyplýva, že pri používaní je nutné vytvoriť najprv inštanciu triedy *jcart()*.

```
$cart =& $_SESSION['jcart'];  
if(!is_object($cart)) $cart = new jcart();  
$cart->display_cart($jcart);
```

Ak sa inštancia podarí, metóda s intuitívnym názvom *display\_cart()* bez prekvapenia zobrazí vytúžený „košík“. Zobrazenie jednotlivých skladieb je rovnaké, ako pri zobrazovaní všetkých skladieb od všetkých užívateľov. Jediným rozdielom je pridanie tlačidla a vloženie skrytého poľa (hidden field) pre účely jCart. Ešte je treba spomenúť neoddeliteľnú súčasť ktorou sú odkazy na jQuery, jCart a css súbory. Na obrázku č. 12 je vidieť „košík“ s pridanými skladbami.

## Vytvorenie profilu

Playlist (2 položiek)			
1	AC/DC	Nieco	kopec 00:04:59 <a href="#">Odstrániť</a>
2	Faithless	Insomnia	rovina 00:04:39 <a href="#">Odstrániť</a>
Celkový čas: 00:09:38			
<input type="button" value="Pokračovať"/> Názov playlistu: <input type="text"/>			

Č	Interpret	Názov	Dĺžka	Žáner	Typ	Užívateľ	Dátum	Pridať
1	Eva a Vašek	Orchidej	00:04:22	Alternative	cooldown	svoby	18.05.12	<input type="button" value="Pridať"/>
2	Florence and the machine	Drumming song	00:03:44	Pop	rovina	svoby	14.05.12	<input type="button" value="Pridať"/>
3	Survivor	Burning heart	00:03:52	Rock	kopec	svoby	23.05.12	<input type="button" value="Pridať"/>
4	AC/DC	Nieco	00:04:59	Rock	kopec	svoby	14.05.12	<input type="button" value="Pridať"/>
5	Faithless	Insomnia	00:04:39	Electronic	rovina	svoby	18.05.12	<input type="button" value="Pridať"/>

Obrázok 12: Ukážka vytvorenia playlistu

Užívateľ má možnosť svoj profil aj uložiť. Premýšľal som, podľa čoho bude najlepšie ukladanie a prípadné vyhľadávanie profilov. Jedným z nápadov bolo podľa dátumu, ale prišlo mi to nepraktické, predsa uloženie podľa mena resp. názvu profilu mi prišlo najprívetivejšie a také user-friendly. V procese ukladania bolo potrebné opäť zavolať triedu *jcart()* a metódu na získanie hodnôt z „košíka“ *get\_contents()*. Opäť bolo treba spraviť konverziu času tým istým spôsobom, ako bolo popísané vyššie. V cykle *for* spočítať celkovú dĺžku, vytvoriť štruktúru XML, ktorá bude vkladaná do databázy a záverečný proces ukladania. Ten prebieha takmer identicky ako pri ostatných vkladaniach.

Ako som spomínal v kapitole Prehľad a nastavenia účtu, ak má užívateľ aspoň jeden playlist, je možnosť pozrieť si svoje uložené profily. V tomto kroku je zobrazený len ich zoznam, ale po kliknutí na ikonu, je rozbalený ľubovoľný playlist daného užívateľa. Taktiež ako to bolo pri skladbách, je umožnené vymazať svoje profily.

### 5.2.11 Vyhľadávanie

Netreba si tu predstavovať úplne špičkový search engine. Ani žiadne Google like vyhľadávanie. Ide o pomerne jednoduché vyhľadávanie, ktorého funkciou je hľadanie podľa zadaných kritérií. Rozdelil som ich na hlavné a rozšírené a sú to tieto kritériá:

- Podľa názvu interpreta
- Podľa názvu skladby
- Podľa typu
- Podľa žánru

Možné sú rôzne kombinácie, pričom toto obmedzenie sa vzťahuje na hlavné a rozšírené hľadanie. Za zmienku ešte stojí spomenúť jednoduchý jQuery kód, potrebný na zobrazovanie rozšíreného hľadania po kliknutí na tlačidlo. Aj keď neobľubujem jQuery prípadne Javascript, tento kód je veľmi jednoduchý na pochopenie. Skript počká, kým bude *document* načítaný. Pomocou CSS vlastností kontajneru *div* s triedou *extend\_search* a funkcie *click()*, ktorá zobrazí tabuľku pre zmenu s id *extendSearch*. Zobrazenie je vykonávané tzv. *slidedown* efektom a definovanou rýchlosťou rolovania.

```
<script type="text/javascript">
    $(document).ready(function(){
        $(".extend_search").click(function(){
            $("#table#extendSearch").slideDown(1000);
        });
    });
</script>
```

Pri procese vyhľadávania, bolo treba nadefinovať vhodné XPath dotazy. Tých je celkovo šesť. Základom je vyhradené XPath slovo *extract* a pridanie podmienky na presvedčenie sa o existencii elementu *existsNode*. Opäť bude lepšie si ukázať to na príklade:

```
$existsNodePath = '//type[text()=''.$srch_type.'']';
$query_srchType =
    "select extract(track, '/songs/*') from track_list where
    existsNode(track, '$existsNodePath')=:cislo";
```

Hodnota, ktorá sa testuje v XPath je preberaná z užívateľského vstupu, ktorý je ošetrený. Bind parameter „cislo“ je nutný, pretože tento dotaz vracia povedzme boolean hodnotu.

Toto tvrdenie však presne nevystihuje hodnotu, ktorú v skutočnosti dostaneme. Áno, v negatívnom prípade nedostaneme žiadne výsledky, ale pri nájdení zhody dostaneme TRUE hodnotu, avšak nazval by som ju ako takou imaginárnou hodnotou, pretože v skutočnosti dostaneme výpis všetkého, čo sa zhoduje s hľadaným výrazom alebo reťazcom. V Prílohe P II uvádzam zvyšných päť dotazov pre vyhľadávanie v aplikácii.

Nevýhodou tohto hľadania je skutočnosť, že zadané kritériá musia byť rovnaké, aké sú v databáze. To znamená závislosť na veľkých a malých písmenách (case-sensitive), ako aj na diakritike pri vyhľadávaní slovenských, prípadne českých skladieb. Uvediem príklad. Ak sa v databáze nachádza skladba „Jede jede mašinka“, vyhľadávanie nevráti výsledky po zadaní „jede jede masinka“ prípadne „jede jede mašinka“. Uvedomujem si, že to nie je najideálnejšie. Vyhľadávanie by sa však dalo z optimalizovať rôznymi spôsobmi. Ako prvý ma napadá konverzia hľadaného reťazca a všetkých prehľadávaných reťazcov na malé písmená bez diakritiky, druhým spôsobom vyhľadávanie pomocou podreťazcov (substring search). Tzn. z hľadaného reťazca zobrať len prvé tri alebo 4 znaky. Ďalším spôsobom by mohlo byť využitie tzv. autocomplete. Pre tých, ktorí nevedia o čo ide vysvetlím. Ide o jQuery UI, ktoré ponúka užívateľovi návrhy pri hľadaní. Prečo chodiť okolo horúcej kaše, keď krásnym príkladom je samotný Google. Takže možnosti a priestor na optimalizáciu existujú.

### 5.2.12 Chybové oznamy a nadpisy

Tie riešim pomocou triedy *ClassStatements*. Využíva sa buď metóda *message(messageType)* alebo metóda *title(titles)*, v závislosti na tom, čo potrebujeme.

V prípade nadpisov je situácia o niečo jednoduchšia, pretože nepotrebujem získavať údaje zo SESSION, tak ako to je pri správach. Stačí zavolať statickú metódu ako je vidieť na príklade:

```
echo Statements::titles('home');
```

Podľa hodnoty v argumente, sa v tabuľke NADPISY, vyhľadá daný XML záznam. Štruktúra tohto záznamu, ako aj ukážka vkladania do tabuľky je zobrazená v prílohe č.IV.

Pri zobrazovaní chybových, upozorňovacích alebo úspešných správ je postup nasledovný. Najskôr v danom skripte musím inicializovať a naplniť premennú `$_SESSION` hodnotou. Mohlo by to vyzeráť nasledovne:

```
$_SESSION['createPlaylistERR'] = 'createPlaylistERR';
```

Na stránke, kde chceme danú správu zobrazit' potrebujeme testovať hodnotu premennej v SESSION. Môže to vyzerat' nasledovne:

```
if ($_SESSION['createPlaylistERR'] == 'createPlaylistERR') {  
    echo Statements::message('error', 'createPlaylistERR');  
    $_SESSION['createPlaylistERR'] = null;  
}
```

V závere je treba priradiť SESSION s daným menom hodnotu NULL, aby sa správa nezobrazovala. V prílohe č. V uvádzam štruktúru XML ako aj príklad vloženia záznamu do tabuľky STATEMENTS. Uvedomujem si, že by sa to dalo spraviť iným spôsobom. Napríklad preberať parameter z URL riadka a pomocou \$\_GET premennej zobrazovať správy. Tak isto by sa mohli ukladať a načítavať z XML, ktoré nie je uložené v databázy. Ja som si zvolil vyššie opísaný a som s ním spokojný.

### 5.3 Užívateľský návrh aplikácie

Táto kapitola nemá nič spoločné s databázami alebo PHP. Jej cieľom je vysvetliť hlavnú myšlienku celej aplikácie z pohľadu užívateľa. Vysvetliť prečo som postupoval tak a nie inak a podobne.

Prihlasovanie a registrácia mi prišla dobrý nápad, už len z dôvodu informatívneho. Tzn. koľko ľudí by využívalo databázu, zníženie relatívnej anonymity a pod. Čo sa týka samotnej hudobnej databázy, hlavná myšlienka je viditeľnosť všetkých skladieb. Aj tých, ktoré užívateľ sám nevložil. Výhoda tohto prístupu je v tom, že ak užívateľ je vyčerpaný a bez nápadov, vždy existuje kopec ľudí, ktorí môžu rozšíriť jeho obzory. Ak by tento prístup nefungoval, aplikácia by strácala logiku a nedochádzalo by k výmene informácii, tak ako som si predstavoval. Tie by boli izolované a viditeľné len pre daného užívateľa. Avšak treba brať do úvahy fakt, že vidieť je jedna vec, ale spravovať je druhá. Preto užívateľ má možnosť upravovať, zmazávať len svoje skladby. Čo sa týka profilov, ich zobrazovanie som obmedzil. Sú viditeľné iba pre toho, kto si ich vytvoril. To by sa dalo vylepšiť dobrovoľnou voľbou zobrazovania profilu. Pri samotnom vytváraní uložiť ako súkromný prípadne verejný. Záležalo by od samotného užívateľa, či sa chce pochváliť alebo nie.

### 5.3.1 Využitie aplikácie

Využitie aplikácie je zamerané hlavne na indoor cycling. To však nevyklučuje iné športy, ktoré využívajú hudbu ako doprovod. Ide hlavne o hudobnú inšpiráciu, ktorá v niektorých momentoch môže stagnovať a takáto aplikácia je podľa môjho názoru potrebná. Na profesionálnej úrovni sa využívajú komerčné softwareové riešenia. Za všetky môžem spomenúť napr. Mixmeister, ktorého však najlacnejšia verzia stojí 70 dolárov.

## 6 MOŽNÉ NÁVRHY NA VYLEPŠENIE

Chvíľu som zvažoval o pridaní tejto kapitoly, ale nakoniec som sa rozhodol, že ju pridám. Aj keď sa to môže zdať, že si budem sypať popol na hlavu, urobím to. Keďže som človek ktorý sa snaží robiť veci čo najlepšie, nie vždy sa však výsledný, chcený efekt dostaví tak ako si predstavujem. Aj tu vidím možnosti ďalšieho rozširovania tejto aplikácie. V nasledovnom texte načrtnem možné vylepšenia a návrhy, ako by sa dali uskutočniť.

### 6.1 Vyhľadávanie

Ako som už spomínal, možnosť vyhľadávania je do značnej miery limitovaná. Možný návrh riešenia som vysvetlil v kapitole 5.2.11 To bolo však hľadanie podľa reťazcov. Ďalšou možnosťou je vyhľadávanie podľa dĺžky. Opäť som v tejto časti narazil na problém. Menším je opäť konverzia času. Tá sa však podarila odstrániť. Posledný problém nakoniec vznikol v tom, že záverečný dotaz využíva vnorený dotaz, ktorý bližšie špecifikuje dĺžku podľa ktorej sa má vyhľadávať. Riešením by mohla byť zmena prístupu (nevyužitie dočasnej tabuľky, optimalizácia výberových dotazov a pod.)

### 6.2 Získavanie vzdialeného obsahu

Názov kapitoly nie je až taký jasný, pokúsim sa vysvetliť, čo som tým myslel. Na internete existuje server <http://musicbrainz.org>. Táto iniciatíva legálne zokupuje informácie o albumoch, interpretoch atď, z najznámejších hudobných portálov ako last.fm, myspace.com a iné. Pre vývojárov poskytuje tzv. XML Web service. Tá poskytuje vyhľadávanie na ich databázach, pričom vracia výsledky vo forme XML [14]. V mojej aplikácii by som tieto informácie využil pri vkladaní novej skladby, keď by užívateľ nevedel napr. dĺžku skladby, klikol by na odkaz, ktorý by mu zobrazil danú skladbu so základnými údajmi.

Na toto rozšírenie som vytvoril triedu *ClassExternalData()*, využívajúcu cURL. To je viac menej podpora pre konektivitu a komunikáciu cez rôzne protokoly na internete. Dôvodom nevyužitia tejto triedy bol fakt, že výsledný XML výstup bolo treba z optimalizovať pre správny dotaz. Pretože pri testovaní sa stávalo, že neboli nájdené správne hodnoty, alebo boli vrátené hodnoty, ktorých bolo tak veľa, že v nich by sa ešte muselo využiť vnoreného

vyhľadávania. Preto som tento postup zavrhol z časového hľadiska, ale je dobré vedieť, že takáto možnosť existuje.

### 6.3 Súkromné vs. verejné zobrazovanie

Už som to trochu načrtol v kapitole Užívateľský návrh aplikácie a aký je aj momentálny stav. Rozšírením by mohlo byť riadenie súkromia, podľa používateľa, ako si zvolí. Tzn. verejné alebo súkromné ukladanie profilov. Ďalšia možnosť, by bola ukladanie iba pre určitú skupinu ľudí, ktorých má užívateľ v skupine. Avšak pri tomto riešení už mám pocit, že by to spadalo do prvkov sociálnej aplikácie.

Určite nie posledná možnosť by mohla byť obyčajné prezeranie profilov účastníkmi indoor cycling-u. Mám na mysli zverejnený playlist a podľa neho môžu účastníci vidieť, aká hudba bude hraná, aká ťažká/ľahká hodina ich čaká a podobne. Taktiež by mohlo byť do aplikácie zakomponované menšie moderované fórum, kde by účastníci mohli vkladať svoje postrehy, hudobné návrhy, prípadne pochvaly atď. Takáto spätná väzba je veľmi potrebná a dobrá a to v každej oblasti.

Odpoveď, prečo som tieto nápady na zlepšenie nezrealizoval je pomerne jednoduchá. Všetko je to záležitosť času. Myslím si, že výhodou pre mňa je skutočnosť že o tom viem aké sú možnosti, viem ako to spraviť a prípadne pokračovať ďalej.

Na záver tejto kapitoly poviem už len pár viet. Nápadov a myšlienok by bolo viacej. Mám potvrdené, že takáto aplikácia by mala úspech.

## 7 SPOJAZDNIENIE NA SERVERI

Je to posledný bod, ktorý ma čaká vysvetliť.

### 7.1 Vytvorenie schémy

Nebudem tu spomínať čo je to schéma, to je vysvetlené v teoretickej časti tejto práce. Skôr sa zameriam na jej praktické vytvorenie. To som vytvoril pomocou webového rozhrania Oracle v tzv. Oracle Enterprise Manager. Bolo potrebné sa prihlásiť ako sysman, pretože iným užívateľským heslom, by vytvorenie nemuselo fungovať vzhľadom k privilégiám. Nový user (resp. schéma) sa vytvára v záložke „Users & Privileges“. Tlačidlom CREATE sa dostaneme do módu, umožňujúceho vytvorenie nového užívateľa. Po vyplnení základných údajov ako sú: názov, profil, heslo, defaultný a dočasný tablespace a odomknutie schémy je potrebné pridať systémové a objektové privilégiá. Okrem týchto základných nastavení sa môžu nastaviť kvóty pre ukladanie dát atď. Príklad schémy DP je možné vidieť na obrázku č. 13

The screenshot shows the Oracle Enterprise Manager interface for editing a user. At the top, it indicates 'Database Instance: DP > Users >' and 'Logged in As SYSMAN'. The page title is 'Edit User: DP'. Below the title, there are action buttons: 'Actions Create Like', 'Go', 'Show SQL', 'Revert', and 'Apply'. A navigation bar contains tabs for 'General', 'Roles', 'System Privileges', 'Object Privileges', 'Quotas', 'Consumer Group Privileges', and 'Proxy Users'. The 'General' tab is selected. The form fields are: Name: DP; Profile: DEFAULT; Authentication: Password; \* Enter Password: [masked]; \* Confirm Password: [masked]; For Password choice, the role is authorized via password. [ ] Expire Password now; Default Tablespace: USERS; Temporary Tablespace: TEMP; Status: [ ] Locked [x] Unlocked.

Obrázok 13: Vytvorenie schémy v Oracle

## 7.2 Adresárová štruktúra

Snažil som sa ju zvoliť čo najjednoduchšie, avšak aby spĺňala aj praktické aspekty návrhu. Základom je koreňový adresár, ktorý je viac menej nezávislý na serverovom názve. Ten sa naprieč celou aplikáciou používa cez konfiguračnú premennú - `$cfg['domain']`. Tá je definovaná v súbore `conf.php`. Výhodou je potom jedna jednoduchá zmena doménového mena, prípadne servera, na ktorom aplikácia beží. V mojom prípade by to mohlo vyzerat' nasledovne:

```
$cfg['domain'] = 'svoboda';
```

Vo vnútri rodičovskej zložky sa nachádza desať zložiek a ďalšie podzložky. Vyzerá to nasledovne:

- `cfg[domain]`
  - `classes`
  - `conf`
  - `core`
    - `includes`
    - `templates`
      - `design`
      - `form`
      - `tables`
  - `css`
  - `data`
    - `xml`
  - `images`
    - `icons`
  - `js`
    - `lib`
  - `log`
  - `modules`
    - `clearfield`
    - `jcart`
    - `login`
    - `registration`

- upload
- pages
  - basic
  - registration

V tejto kapitole chcem vysvetliť ešte konvenciu, ktorú som používal pri pomenovávaní jednotlivých súborov. Pri triedach, ako sa dalo všimnúť, je názov *ClassRozširujúciNázovTriedy*. Okrem toho, pomenovávam názvy includovaných súborov takto: *nazovSuboru.inc.php*. V prípade, že sa jedná o designový template, dá sa ľahko tušiť ako táto konvencia bude vyzerat': *nazovTemplate.tpl.php*. Jemnejšie členenie nastáva až pri rozdeľovaní na template tabuľky alebo formulára. Tam je formát nasledovný: *nazovTabulkyTbl.tpl.php* resp. *nazovFormularaForm.tpl.php*.

## ZÁVĚR

Hlavným cieľom tejto diplomovej práce bolo vytvorenie webovej aplikácie využívajúcej XML funkcie databázy Oracle. Navrhnutá aplikácia túto funkcionálnosť využíva a zameriava sa na jej praktické využitie. V prvej časti práce vysvetľujem teoretické základy, potrebné pre pochopenie tak Oracle databázy, ako aj ostatných pojmov týkajúcich sa témy diplomovej práce. Medzi tieto pojmy patrí problematika a využitie XML, výber a stručný úvod do programovacieho jazyka. Okrem týchto základných teoretických podkladov bolo potrebné navrhnutie aplikácie, ktorá by vyššie spomínanú funkcionálnosť spĺňala. To sa mi podarilo a postupný vývoj je opísaný v druhej, praktickej časti tejto práce.

Okrem týchto pojmov a informácií bolo počas práce potrebné naštudovať rozsiahlejšiu problematiku, hlavne v oblasti systému Oracle. Táto problematika priamo nesúvisela so zadaním práce, ale mala do určitej miery vplyv na celkový priebeh a výsledok práce. Toto rozšírenie oblasti, si myslím, že má veľmi pozitívny charakter, pretože mi umožnilo sa naučiť širšie spektrum vecí počas diplomovej práce. Preto aj prínos, ktorý okrem webovej aplikácie vznikol, je aj prínos pre mňa samotného, ktorý oceňujem.

## ZÁVĚR V ANGLIČTINĚ

The main goal of my diploma thesis was creating a web application which used XML functions in the Oracle database. The design application used this functionality and is focused on its practical use. This thesis in its first part is describing theoretical basics, which are needed for better understanding of the Oracle database. This basics are also important for other subject in this thesis. Among these subjects belong using XML, choosing and brief introduction into programming language. Besides of this basic theoretical backgrounds, it was needed design of application, which would be mentioned functionality used. I realized it and progressive development is described in second, practical part of this thesis.

Beside this information, during the work extensive issues needed to be studied, mainly in the Oracle system field. This information was not related directly to the theme of this thesis, but it has certain influences on the whole process and the result of my work. I think, this extended field of work have had big and positive character, because it helped me to learn a wider spectrum of things during my thesis. So besides of the web application that was created, this benefited me which I really appreciated.

**SEZNAM POUŽITÉ LITERATURY**

- [1] LACKO, Luboslav. ORACLE: Správa, programování a použití databázového systému. Vyd. 1. Brno: Computer Press, 2003, 464 s. ISBN 80-722-6699-3.
- [2] LECKY-THOMPSON, Ed a Steven D NOWICKI. PHP 6: programujeme profesionálně. Vyd. 1. Překlad Ondřej Gibl. Brno: Computer Press, 2010, 718 s. Programujeme profesionálně. ISBN 978-802-5131-275.
- [3] KOSEK, Jiří. PHP - tvorba interaktivních internetových aplikací: podrobný průvodce. Vyd. 1. Praha: Grada, 1999, 490 s. Průvodce (Grada). ISBN 80-716-9373-1.
- [4] MLÝNKOVÁ, Irena a Jaroslav POKORNÝ. XML technologie: principy a aplikace v praxi. 1. vyd. Praha: Grada, 2008, 267 s. Průvodce (Grada). ISBN 978-802-4727-257.
- [5] URMAN, Scott, Ron HARDMAN a Michael MCLAUGHLIN. Oracle: programování v PL/SQL. Vyd. 1. Překlad Jiří Fadrný. Brno: Computer Press, 2007, 720 s. ISBN 978-802-5118-702.
- [6] Schema Objects. *5 Schema Objects* [online]. 2003 [cit. 2012-05-24]. Dostupné z: <http://www.stanford.edu/dept/itss/docs/oracle/10g/server.101/b10743/schema.htm>
- [7] CYRAN, Michele. 1 Introduction to the Oracle Database. ORACLE. *Introduction to the Oracle Database* [online]. 2005 [cit. 2012-05-24]. Dostupné z: [http://docs.oracle.com/cd/B19306\\_01/server.102/b14220/intro.htm](http://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm)
- [8] BELDEN, Eric a Janis GREENBERG. Key Features of the Object-Relational Model. ORACLE. *Introduction to Oracle Objects* [online]. 2008 [cit. 2012-05-24]. Dostupné z: [http://docs.oracle.com/cd/B28359\\_01/appdev.111/b28371/adobjint.htm#CHDDCDBF](http://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjint.htm#CHDDCDBF)
- [9] JONES, Christopher a Alison HOLLOWAY. The Underground PHP and Oracle Manual. ORACLE. *Oracle php* [online]. 2008 [cit. 2012-05-24]. Dostupné z: [http://www.scribd.com/doc/49763950/204/Fetching-XMLType-Columns#outer\\_page\\_21](http://www.scribd.com/doc/49763950/204/Fetching-XMLType-Columns#outer_page_21)
- [20] SMITH, Rick. Storing PHP Sessions in a Database. *Devshed.com* [online]. 2007 [cit. 2012-05-24]. Dostupné z: <http://www.devshed.com/c/a/PHP/Storing-PHP-Sessions-in-a-Database/>

- [31] LIECHTI, André. Secure Token Grid Authentication. *Secure Token Grid Authentication, Generate grid of tokens, authenticate user with it - PHP Classes* [online]. [2008] [cit. 2012-05-24]. Dostupné z: <http://www.phpclasses.org/package/4518-PHP-Generate-grid-of-tokens-authenticate-user-with-it.html>
- [42] Prepared statements. *PHP: Prepared Statements - Manual* [online]. 2012, 24.5.2012 [cit. 2012-05-24]. Dostupné z: <http://php.net/manual/en/mysqli.quickstart.prepared-statements.php>
- [53] WHITNEY, Doug. JCart free jQuery/PHP web cart. *JCart - Free PHP/Ajax shopping cart downloaded over 60,000 times* [online]. 2007 [cit. 2012-05-24]. Dostupné z: <http://conceptlogic.com/jcart/>
- [64] XML Web Service / Version 2 / Search. *Search - MusicBrainz* [online]. 2002 [cit. 2012-05-24]. Dostupné z: [http://musicbrainz.org/doc/XML\\_Web\\_Service/Version\\_2/Search](http://musicbrainz.org/doc/XML_Web_Service/Version_2/Search)
- [75] BRÁZA, Jiří. *XML - praktické příklady*. Vyd. 1. Praha: Grada, 2003, 211 s. ISBN 80-247-0699-7.

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

DBMS    DataBase Management System.

SRBD    Systém Riadenia Báže Dát.

XML    eXtensible Markup Language.

DTD    Document Type Definition.

DDL    Data Definiton Language.

DML    Data Manipulation Language.

DCL    Data Control Language.

TCC    Transaction Control Commands.

LOB    Large Objects.

PHP    PHP Hyptertext preprocessor

PDO    PHP Data Objects

OOP    Object Oriented Programming

OCI    Oracle Call Interface

ODBC    Open DataBase Connectivity

SQL    Structured Query Language

**SEZNAM OBRÁZKŮ**

Obrázok 1: Štruktúra dát v databázy.....	13
dostupný z, <a href="http://basicoracledatabase.blogspot.com/2009/02/oracle-physical-structure.html">http://basicoracledatabase.blogspot.com/2009/02/oracle-physical-structure.html</a>	
Obrázok 2: Oracle logické štruktúry.....	14
Obrázok 3: XML schéma – príklad .....	20
Obrázok 4: Spojenie PHP s Oracle pomocou Oracle Client Libraries .....	23
Obrázok 5: Databázové tabuľky - návrh.....	28
Obrázok 6: Registračný formulár - ukážka.....	38
Obrázok 7: Formulár pre prihlásenie .....	39
Obrázok 8: užívateľské rozhranie .....	41
Obrázok 9: Tabuľka skladieb konkrétneho užívateľa.....	43
Obrázok 10: Súhrn užívateľských informácií .....	45
Obrázok 11: Formulár pre užívateľské úpravy .....	45
Obrázok 12: Ukážka vytvorenia playlistu .....	50
Obrázok 13: Vytvorenie schémy v Oracle.....	57

## SEZNAM TABULEK

## SEZNAM PŘÍLOH

Príloha P I: Štruktúra XML

Príloha P II: XPath dotazy pri vyhľadávani

Príloha P III: Dotaz vlozenia žánrov do tabuľky GENRES

Príloha P IV: Štruktúra XML a vkladanie údajov do tabuľky NADPISY

## PŘÍLOHA P I: ŠTRUKTÚRA XML

```
$songNode = $newTag->appendChild($songNode);

$newSong = $newTag->createElement('track');
$newSong = $songNode->appendChild($newSong);
$newSong->setAttribute("idTrack", $idSong-
>autoIncrement('id_song', 'track_list'));

$artistNode = $newTag->createElement("artist",
$interpret);
$artistNode = $newSong->appendChild($artistNode);

$titleNode = $newTag->createElement("titleTrack",
$nazov);
$titleNode = $newSong->appendChild($titleNode);

$durationNode = $newTag->createElement("duration",
$duration);
$durationNode = $newSong->appendChild($durationNode);

$genreNode = $newTag->createElement("genre", $zaner);
$genreNode = $newSong->appendChild($genreNode);

$typeNode = $newTag->createElement("type", $typ);
$typeNode = $newSong->appendChild($typeNode);

$userNode = $newTag->createElement("username",
$user_lastLog[1]);
$userNode = $newSong->appendChild($userNode);
```

## **PŘÍLOHA P II: XPATH DOTAZY PRI VYHLADAVANI**

```
select  extract(track,  '/songs/*')  from  track_list  where  
existsNode(track, '$existsNodePath' )=:cislo
```

```
select  extract(track,  '/songs/*')  from  track_list  where  
existsNode(track, '$existsNodePath' )=:cislo  and  
existsNode(track, '$existsNodePath2')=:cislo
```

```
select  extract(track,  '/songs/*')  from  track_list  where  
existsNode(track, '$existsNodePath' )=:cislo
```

```
select  extract(track,  '/songs/*')  from  track_list  where  
existsNode(track, '$existsNodePath' )=:cislo
```

```
select  extract(track,  '/songs/*')  from  track_list  where  
existsNode(track, '$existsNodePath' )=:cislo  and  
existsNode(track, '$existsNodePath2' )=:cislo
```

## **PŘÍLOHA P III: DOTAZ VLOŽENIA ŽÁNROV DO TABULKY GENRES**

```
insert into genres values(1, 'Alternative');  
insert into genres values(2, 'Blues');  
insert into genres values(3, 'Classical');  
insert into genres values(4, 'Country');  
insert into genres values(5, 'Dance');  
insert into genres values(6, 'Electronic');  
insert into genres values(7, 'Hip - hop/Rap');  
insert into genres values(8, 'Latino');  
insert into genres values(9, 'New Age');  
insert into genres values(10, 'Pop');  
insert into genres values(11, 'RnB');  
insert into genres values(12, 'Rock');  
insert into genres values(13, 'Iné');  
commit;
```

## PŘÍLOHA P IV: ŠTRUKTÚRA XML A VKLADANIE ÚDAJOV DO TABUĽKY NADPISY

```
<?xml version="1.0"?>
```

```
  <nadpisy>
```

```
    <home>Diplomová práca 2012 ::</home>
```

```
  </nadpisy>
```

```
insert into nadpisy (id_nadpis, nadpis)
```

```
values (1, XMLType('<?xml version="1.0"?>
```

```
  <nadpisy>
```

```
    <home> Diplomová práca 2012 ::</home>
```

```
  </nadpisy>
```

```
  '));
```

```
commit;
```

## PŘÍLOHA P V: ŠTRUKTÚRA XML A VKLADANIE ÚDAJOV DO TABUĽKY STATEMENTS

```
<?xml version="1.0"?>
  <statements>
    <statement>
      <createPlaylistERR>Pri ukladaní playlistu sa vyskytli
chyby!</createPlaylistERR>
    </statement>
  </statements>
```

```
insert into statements (id_statement, statement)
values (1, XMLType('<?xml version="1.0"?>
  <statements>
    <statement>
      < createPlaylistERR >Meno alebo heslo
nie je správne!</ createPlaylistERR >
    </statement>
  </statements>
  '));
commit;
```