# ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

**Fakulta elektrotechnická**
**Katedra počítačů**

**Administrační nástroj pro databázové systémy**
**(stromové procházení, rozdíly)**

**Administration tool for database systems**
**(tree view, diff)**

Bakalářská práce

Studijní program: Softwarové technologie a management
Studijní obor: Softwarové inženýrství

Vedoucí práce: RNDr. Ondřej Žára

**Petro Kostyuk**
**Praha 2017**

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Kostyuk Petro

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: Administrační nástroj pro databázové systémy (stromové procházení, rozdíly)

Pokyny pro vypracování:

Seznamte se s problematikou správy relačních databází a s frameworkem Nette pro psaní interaktivních webových stránek v jazyce PHP. Následně navrhněte a implementujte tři komponenty pro administrační rozhraní relačních databází: nástroj pro správu struktury databáze, stromový pohled na data (s využitím Foreign Key Constraints) a nástroj pro generování rozdílů (strukturálních i datových) mezi více databázemi. Prozkoumejte možnost využití databázové abstrakce tak, aby výsledným produktem bylo možno spravovat více různých druhů databází. Definujte množinu podporovaných databází a následně zdůvodněte svůj výběr. Vzniklý nástroj kvalitativně porovnejte s existujícími řešeními (phpMyAdmin, Adminer). Nedílnou součástí práce bude dokumentace, včetně programátorské a automaticky generované.

Seznam odborné literatury:

[1] Michael Kofler: Mistrovství v MySQL 5, Computer Press 2007, ISBN 978-80-251-1502-2
Steven D. Nowicki, Ed Lecky-Thomson: PHP 6 programujeme profesionálně, Computer Press 2010, ISBN 978-80-251-3127-5 Nette Framework 2.4, https://doc.nette.org/cs/2.4/

Vedoucí: RNDr. Ondřej Žára

Platnost zadání do konce letního semestru 2017/2018


prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 19.1.2017

## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Praha 24.05.2017

.....

Petro Kostyuk

## Poděkování

Tímto děkuji vedoucímu této bakalářské práce, RNDr. Ondřeji Žárovi, za jeho vstřícnost během vedení a za všechny jeho rady a připomínky. Dále děkuji své rodině a přátelům za podporu, kterou mi během této práce poskytli.

## Abstrakt

Tako práce se zabývá vytvořením systému pro správu databází. Předností tohoto systému bude jeho schopnost vyhledat a zobrazit data, která mezi sebou souvisí (záznamy tabulek, které jsou mezi sebou propojeny cizím klíčem) ve struktuře, která názorně zobrazuje jejich vzájemný vztah. Další předností tohoto systému bude jeho schopnost porovnat strukturu dvou databází a rozdíly zobrazit uživateli, aby mohl rozhodnout, zda se mají tyto rozdíly nahradit v jedné databází za strukturu, která je ve druhé databázi.



### Klíčová slova

systém pro správu databází, DBMS, zobrazení souvisejících dat, cizí klíče, porovnání schématu databází, úprava schématu databáze



## Abstract

This bachelor thesis deals with the creation of a database management system. The advantage of this system is its ability to search and view the data that are related to each other (Records of tables that are interconnected by a foreign key) in the structure, that illustrates their mutual relationship. Further advantage of this system is its ability to compare the structure of two databases and display their differences to the user, so that he can decide whether these differences should be replaced in one database structure by structure in another database.



### Keywords

database management system, DBMS, related data visualisation, foreign keys, database schema comparision, database schema editation

# Obsah

**1 Úvod** **5**

**2 Komponenty** **6**

2.1 Komponenta pro úpravu struktury databáze . . . . . 7

2.1.1 Funkcionalita . . . . . 8

2.2 Ukázková databáze . . . . . 9

2.3 Komponenta pro stromový pohled na data . . . . . 10

2.3.1 Pokročilé možnosti práce s cizími klíči . . . . . 11

2.4 Komponenta pro porovnání dvou databází . . . . . 13

2.4.1 Struktura porovnání . . . . . 13

2.4.2 Úpravy propojení a schématu . . . . . 14

2.5 Komponenta pro ukládání změn ve struktuře ES Database do databáze . . . . . 15

**3 Databázová abstrakce** **16**

3.1 DBAL . . . . . 16

3.2 Editable Schema . . . . . 16

3.2.1 Životní cyklus struktury Editable Schema . . . . . 17

3.2.2 Struktura tříd schématu databáze vrstvy Editable Schema . . . . . 18

3.2.3 Integrace třídy ES Database a session . . . . . 20

3.2.4 Integrace třídy ES Database a Doctrine . . . . . 21

3.3 Database Image . . . . . 23

3.4 Možnost spravovat více druhů databází . . . . . 24

**4 Závěr** **25**

4.1 Srovnání s existujícími řešeními . . . . . 25

4.2 Instalace . . . . . 25

**5 Použité zdroje a seznam příloh** **26**

# 1 Úvod

Tato práce se zabývá vytvořením systému pro správu databází. Vybral jsem si právě tuto práci pro to, že se při správě databází často potýkám s úkoly, které se za pomoci existujících systémů řeší zdlouhavě nebo je jejich řešení nepřehledné a může docházet k chybám.

Prvním takovým úkolem je zobrazení záznamu v tabulce včetně všech relevantních záznamů v tabulkách odkazovaných pomocí cizích klíčů. Představte si, že máte databázi knihovny. Pracovník knihovny zjistil, že je jedna kniha poškozená. A naším úkolem je odeslat posledním pěti lidem, kteří si knihu vypůjčili, email s dotazem, v jakém stavu knihu dostali.

V takovém případě nejprve budeme muset vyhledat v tabulce knih příslušnou knihu a zapsat si její id. Dále budeme muset ve spojovací tabulce knihy a osob kterým byla vypůjčena najít poslední záznamy a zapsat si id osob. Dále musíme projít tabulku osob a najít ty, jejichž id jsme si zapsali. Jak je vidět, práce je to zdlouhavá a náchylná na chyby (při špatném zápise nějakého id). A to se jedná o jednoduchou situaci. Pokud například budeme potřebovat odstranit všechny záznamy o zakázce v databázi eshopu, která obsahuje řádek se zakázkou, vybranými produkty, uživateli, jeho kontaktech, záznamech o dopravě atd. najednou se z toho stane úmorná práce.

Proto jednou z komponent, které si tato práce klade za cíl vytvořit je komponenta pro stromový pohled na data. Umožňuje vybrat výchozí záznam a vzhledem k němu zobrazovat další navazující záznamy jednoduše tím, že klikneme na odkaz vedle odkazované hodnoty. Všechny odkazované záznamy uvidíme přehledně na jedné stránce, takže si nebudeme muset nic zapisovat ani překlikávat mezi několika SQL dotazy.

Dalším běžným úkonem je nasazování databáze z testovacího prostředí do produkčního, nebo stahování změn provedených někým jiným do lokálního prostředí. Případně pokud jsou změny v databázi ukládány jako soubory s SQL skriptem a dojde k preskočení jednoho z nich, pak může být obtížné dohledat, co přesně je v živé databázi jinak než v lokální.

Právě za tímto účelem vznikla druhá komponenta, komponenta pro porovnání dvou databází. Ta si klade za cíl vzít dvě databáze, načíst jejich struktury, porovnat je a rozdíl zobrazit uživateli. Ten pak může rozhodnout, zda bude rozdíly nějak manuálně řešit, nebo zda nechá tuto komponentu nahradit strukturu jedné databáze strukturou druhé.

Tato bakalářská práce je součástí většího projektu, který obsahuje i další komponenty, které nejsou součástí mé bakalářské práce. Jedná se třeba o komponentu grafického editoru struktury dat, která je součástí bakalářské práce kolegyně Ksenii Raziny. Výsledný projekt si klade za cíl vytvoření takového systému pro správu databáze, který bude pokrývat jak základní operace nad databázemi, tak i používané komplexnější operace.

Celý projekt je napsany v PHP za využití frameworku Nette.

# 2 Komponenty

Než budu moct popsat funkcionalitu jednotlivých komponent, nejprve stručně popíši databázové abstrakce použité v tomto projektu, jelikož jsou využívané ve všech následujících komponentách. Podrobnému seznámení s databázovými abstrakcemi použitými v této práci je věnována kapitola 3.

Projekt využívá pro připojení k databázi abstraktní vrstvu Doctrine DBAL (database abstraction layer). Jedná se o knihovnu třetí strany, která je schopa se napojit na databázi pomocí třídy `Connection`. Kromě vykonávání SQL dotazů je tato třída také schopna vytvořit instanci třídy `AbstractSchemaManager`, která se stará o čtení a editaci schématu databáze.

Dále využívá databázovou abstrakci Editable Schema, která je vytvořená mnou v rámci tohoto projektu. Její hlavním úkoem je uložení celé struktury databáze do třídy `EditableSchemaDatabase`, která je serializovatelná a tím pádem se dá uložit do session. Tato třída obsahuje kolekci instancí tříd `EditableSchemaTable` a `EditableSchemaForeignKey`.

Jelikož Doctrine umožňuje připojení k databázi, ale neukládá databázové schéma a Editable Schema umí obsáhnout schéma databáze, ale neobsahuje připojení k databázi, byla vytvořena ještě jedna obalová třída `DatabaseImage`, která v sobě obsahuje jak instanci `EditableSchemaDatabase`, tak i instanci Doctrine `Connection`. Stejně jako `EditableSchemaDatabase` je i tato třída serializovatelná a uložitelná do session.

V zájmu přehlednosti budu psát názvy tříd s mezerami, přestože ve skutečnosti názvy tříd v PHP mezery obsahovat nesmějí. Dále pak předponu EditableSchema budu zkracovat na ES. Tedy třída dále označená jako `ES Foreign Key` je ve skutečnosti třída `EditableSchemaForeignKey`.

## 2.1 Komponenta pro úpravu struktury databáze

Komponenta pro úpravu struktury databáze klasickým způsobem není podstatnou komponentou tohoto projektu. Její funkcionalitu plně obsahuje komponenta Grafického editoru struktury databáze, která je součástí závěrečné práce kolegyně Raziny. Nicméně jelikož pro některé uživatele může být přirozenější používat klasický editor struktury, tak je tato komponenta zahrnuta do tohoto projektu.

Do klasického editoru struktury tabulky přejdeme z menu pomocí odkazu [S] vedle názvu tabulky, kterou chceme editovat

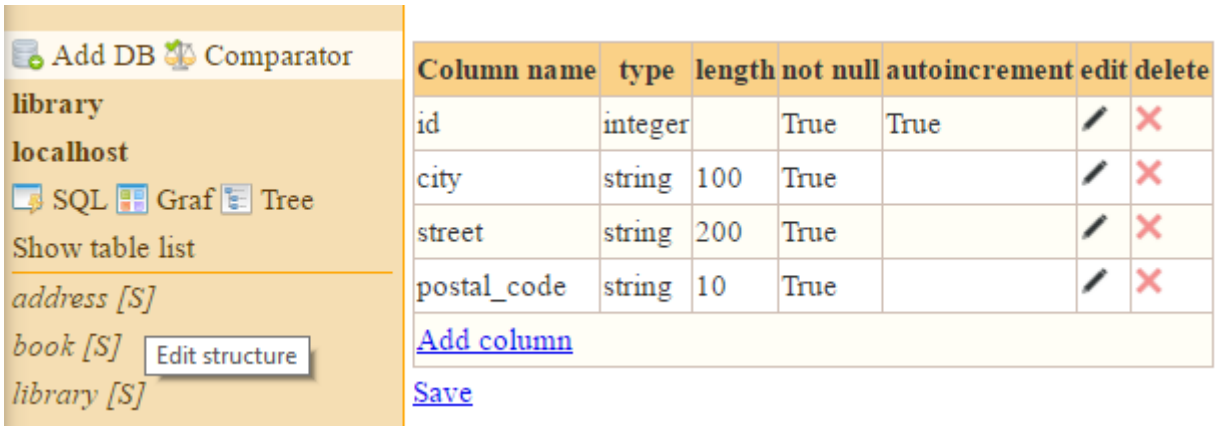



Diagram 1: Vlevo: odkaz na editaci struktury; Vpravo: struktura editované tabulky

Zde máme možnost provádět operace nad jednotlivými sloupci tabulky. S pomocí ikonek u každého záznamu můžeme záznam o sloupci tabulky buď editovat (ikonka tužky), nebo označit k odstranění (červené X). Jakýkoliv záznam označený k odstranění lze vrátit zpátky.

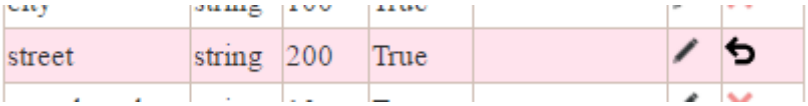



Diagram 2: Záznam označený k odstranění

Při stisku 'Add column' se nám zobrazí formulář, ve kterém nastavíme vlastnosti požadovaného sloupce. Pokud klikneme na editaci řádku, zobrazí se nám stejný formulář, akorát s již předvyplněnými daty o existujícím sloupci. Po vyplnění formuláře odešleme data tlačítkem save, nebo pro zahození změn přejdeme v prohlížeči na předchozí stránku.

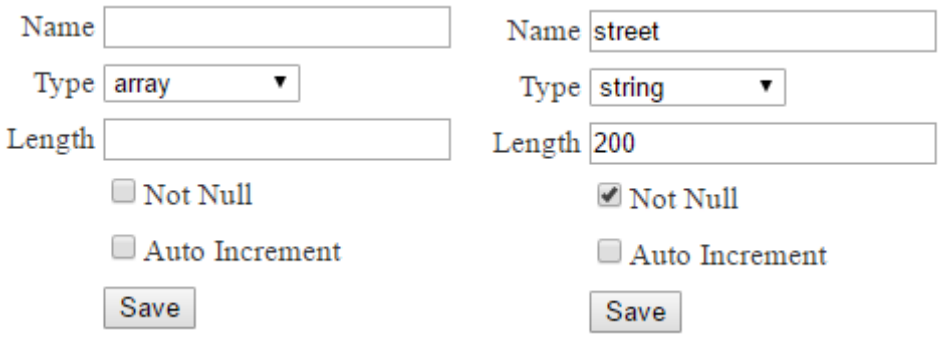



Diagram 3: Formuláře pro vytvoření a pro editaci sloupce tabulky

Po dokončení úprav struktury stiskneme odkaz 'save' a dojde k zobrazení komponenty pro ukládání změn (viz kapitola 2.5)

### 2.1.1 Funkcionalita

Stejně jako ostatní komponenty pracující se strukturou databáze, i tato používá pro uložení schématu třídu `Database Image`. Při stisknutí odkazu pro editaci schématu tabulky, dojde k načtení schématu z databáze do `Database Image`. Následně dojde k uložení `Database Image` do session, (během čehož získáme id instance v session) a k přesměrování na výchozí stránku editoru schématu tabulky.

Jakékoliv změny prováděné nad strukturou tabulky ve svém dotazu pokaždé načtou `Database Image` ze session, provedou změnu na struktuře `ES Database`, ze které si vezmou požadovanou tabulku nad kterou se provádí změna, a změnu provedou. Následně uloží `Database Image` zpět do session a opět dojde k přesměrování na výchozí stránku editoru.

Po dokončení úprav nad tabulkou a po stisku odkazu 'save' na výchozí stránce dojde k předání id, pod kterým je `Database Image` uložena v session, komponentě pro ukládání změn (více v kapitole 2.5).

Výhodou takovéhoto postupu je nezávaznost jakékoliv změny struktury až do poslední chvíle, dokud nedojde k uložení a možnost kdykoliv před uložením prostě zavřít stránku bez toho, aby došlo k poškození původní databáze. Nicméně použití `Database Image` má v tomto případě i svou nevýhodu. `Database Image` byla totiž navržena pro práci s celou databází pro modul porovnávání struktury dvou databází. Vzhledem k tomu při spuštění editoru struktury tabulky dojde k načtení celého schématu databáze do session, což má za následek mnohonásobně větší počet dotazů při spuštění editace, než kolik je nutné pro získání struktury tabulky. V budoucnu je možné tento problém řešit jedním ze dvou způsobů.

První je vytvoření struktury podobné `Database Image`, která by ale obsahovala informace pouze o tabulce. Výhodou by byla možnost naspecifikovat si tuto strukturu právě pro potřeby této komponenty, nevýhodou je to, že ve vrstvě ES je sice tabulka nezávislá na databázi, takže by bylo možné jí zabalit do vlastní struktury, jakési `Table Image`, která by obsahovala tabulku a `Connection Info`, ale zato `ES Table` neobsahuje údaje o cizích klíčích tabulky (úpravu cizích klíčů nemá komponenta pro klasickou editaci struktury tabulky ani teď, ale předpokládám, že pokud až se bude řešit problém načítání celé databáze, bude se tehdy i řešit rozšíření editoru o cizí klíče). Ty jsou uloženy v `ES Database`, takže by bylo potřeba vyřešit nejprve tento problém.

Druhým řešením by bylo upravit vrstvu ES tak, aby bylo možné lazy načítání prvků. V takovém případě bychom načetli pouze tabulku, a pokud bychom potřebovali cizí klíče načetli bychom i je, případně i referované tabulky. Dále bychom museli předělat i komponentu pro ukládání změn, aby neprocházela celou databázi, ale pouze načtené prvky.

Přestože druhé řešení může představovat větší kus práce, přijde mi jako elegantnější a univerzálnější a preferoval bych právě jej.

## 2.2 Ukázková databáze

V následujících ukázkách budu používat databázi o struktuře, kterou znázorňuje následující diagram.

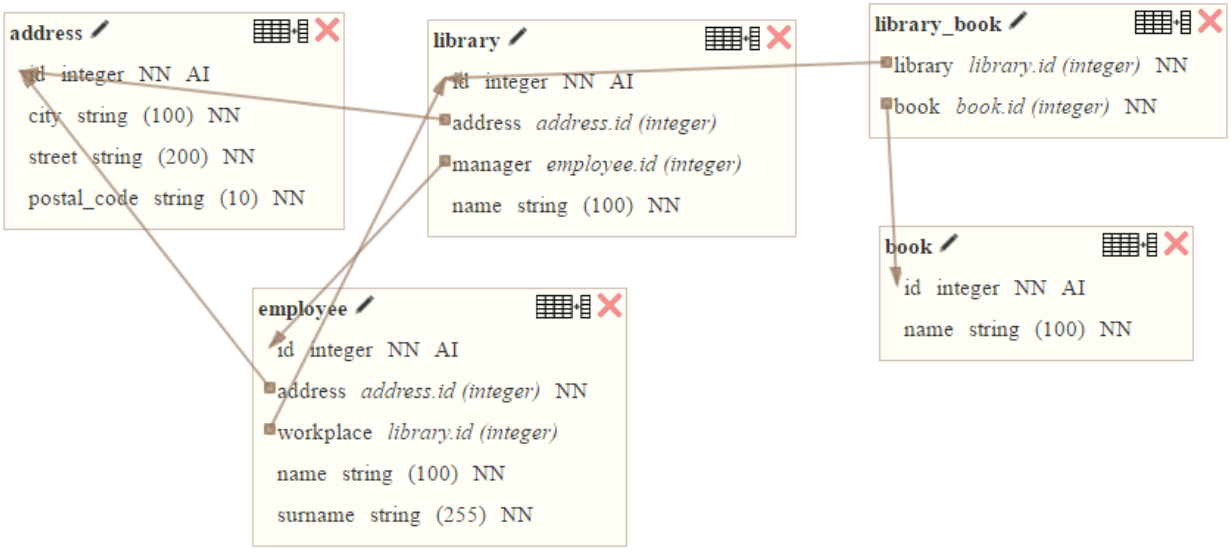



Diagram 4: Schéma ukázkové databáze

## 2.3 Komponenta pro stromový pohled na data

Tato komponenta je určena pro zobrazování dat, které jsou mezi sebou propojené cizími klíči.

Tento modul spustíme tlačítkem 'tree' v menu příslušné databáze. Tím se dostaneme na prázdnou stránku stromového pohledu, na které je pouze horní menu.

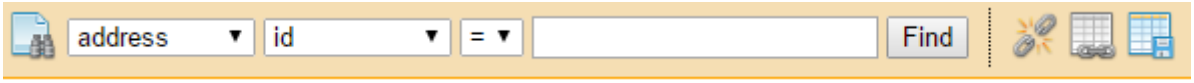
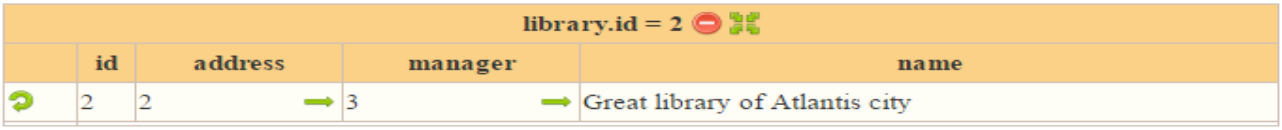



Diagram 5: Horní menu komponenty stromového pohledu na data

První ikonka nám umožňuje skrýt/zobrazit panel pro výběr prvotního záznamu, za ní následuje právě tento panel. Poslední tři ikonky jsou pro práci s cizími klíči, počínaje zleva: zobrazit/skrýt univerzální odkazy, zobrazit/skrýt tabulku cizích klíčů a nakonec uložit změny na cizích klíčích do databáze. Všechny tyto akce budou popsány níže.

Když se dostaneme na prázdnou stránku stromového pohledu, musíme vybrat prvotní záznam, ke kterému budeme následně zobrazovat související záznamy. Vyberme například záznam z tabulky 'library', který má ve sloupci 'id' hodnotu 2. Po výběru hodnot a stisku 'Find' dostaneme následující tabulku s výsledky:



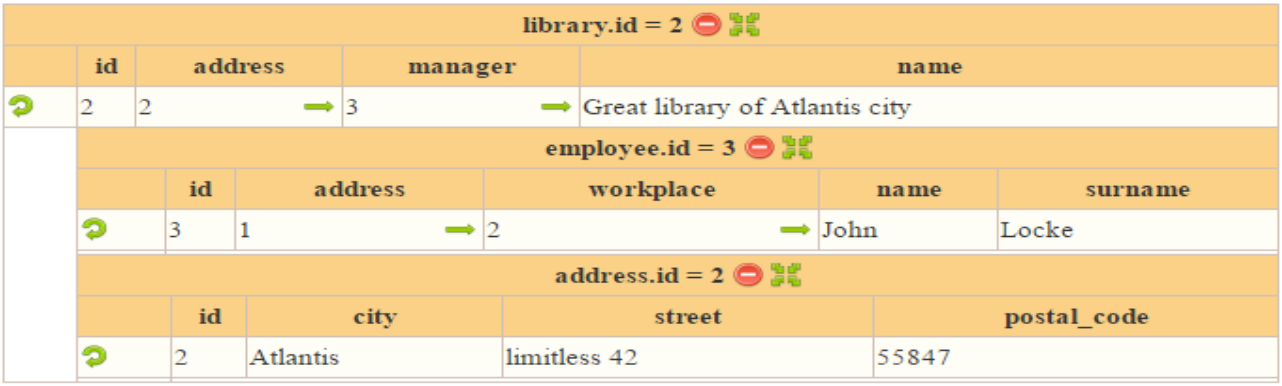
library.id = 2

| | id | address | manager | name |
|---|---|---|---|---|
| | 2 | 2 → | 3 → | Great library of Atlantis city |

Diagram 6: Výběr prvotního záznamu

V prvním řádku hlavičky vidíme podmínku, kterou jsme výsledek dostali. V tomto případě je to 'library.id = 2'. Za tím následují dvě ikonky, první červená pro zavření tabulky v výsledky, druhá zelená pro zobrazení pouze této tabulky (tuto funkcionalitu vysvětlím dále). Ve druhém řádku hlavičky jsou názvy sloupců, které v dané tabulce máme. Nakonec následují řádky se záznamy odpovídajícími zvolené podmínce (v tomto případě máme pouze jeden záznam kde id=2).

Na začátku řádku se záznamem je točená šipka. Pomocí ní můžeme zobrazit záznamy, které odkazují na tento zobrazený záznam. U hodnot 'address' a 'manager' vidíme rovné šipky. Ty signalizují, že daný sloupec odkazuje cizím klíčen na další záznamy. Pokud na ně klikneme, zobrazí se nám u daného záznamu další podtabulka, ve které budeme mít odkazovaný záznam. Pokud nejprve klikneme na šipku vedle manažera a následně na šipku vedle adresy, bude naše tabulka vypadat takto:



library.id = 2

| | id | address | manager | name |
|---|---|---|---|---|
| | 2 | 2 → | 3 → | Great library of Atlantis city |

employee.id = 3

| | id | address | workplace | name | surname |
|---|---|---|---|---|---|
| | 3 | 1 → | 2 → | John | Locke |

address.id = 2

| | id | city | street | postal_code |
|---|---|---|---|---|
| | 2 | Atlantis | limitless 42 | 55847 |

Diagram 7: Zobrazení záznamů odkazovaných v prvotním záznamu

U těchto nových záznamů můžeme opět otevřít další podzáznamy a tím se nám budou zobrazovat související záznamy jako strom. Podle toho se i tato komponenta jmenuje Komponenta pro stromový pohled na data. Aby screenshooty nebyly tak velké, tak tyto nové dvě tabulky záznamů zavřu (červenou ikonkou v hlavičce tabulky). Místo nich zobrazíme všechny zaměstnance, kteří pracují v knihovně v prvotním záznamu. Jelikož informace o tom, ve které knihovně zaměstnanec se nachází v tabulce 'employee', kde sloupec 'workplace' je cizí klíč na 'library.id', zobrazíme zaměstnance přes nabídku cizích klíčů odkazujících na záznam. Ty zobrazíme najetím myši na zelenou točenou šipku na začátku záznamu. tím se nám zobrazí seznam všech cizí klíčů odkazujících na tuto tabulku. Vybereme ten, který chceme zobrazit (v tompo případě 'emplayee.workplace') a kliknutím zobrazíme požadovaná data.

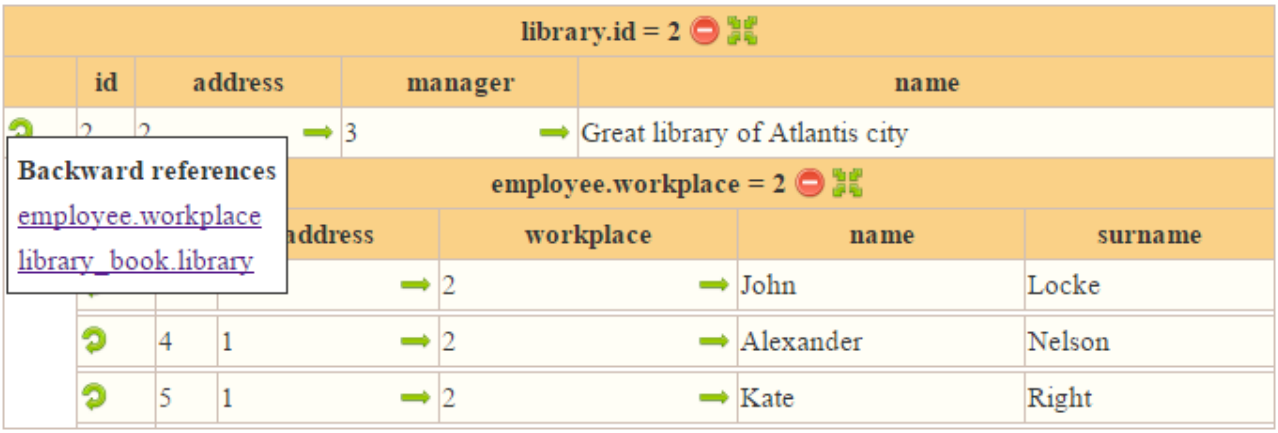



Diagram 8: Zobrazení záznamů odkazujících na prvotní záznam

Pokud bychom teď stiskli zelené tlačítko v hlavičce vnitřní tabulky, zobrazila by se nám pouze vnitřní tabulka (stal by se z ní prvotní záznam). O původní prvotní záznam bychom přišli, ale všechny podzáznamy z této nové tabulky by se přenesly společně s ní a zůstaly by uvnitř.

### 2.3.1 Pokročilé možnosti práce s cizími klíči

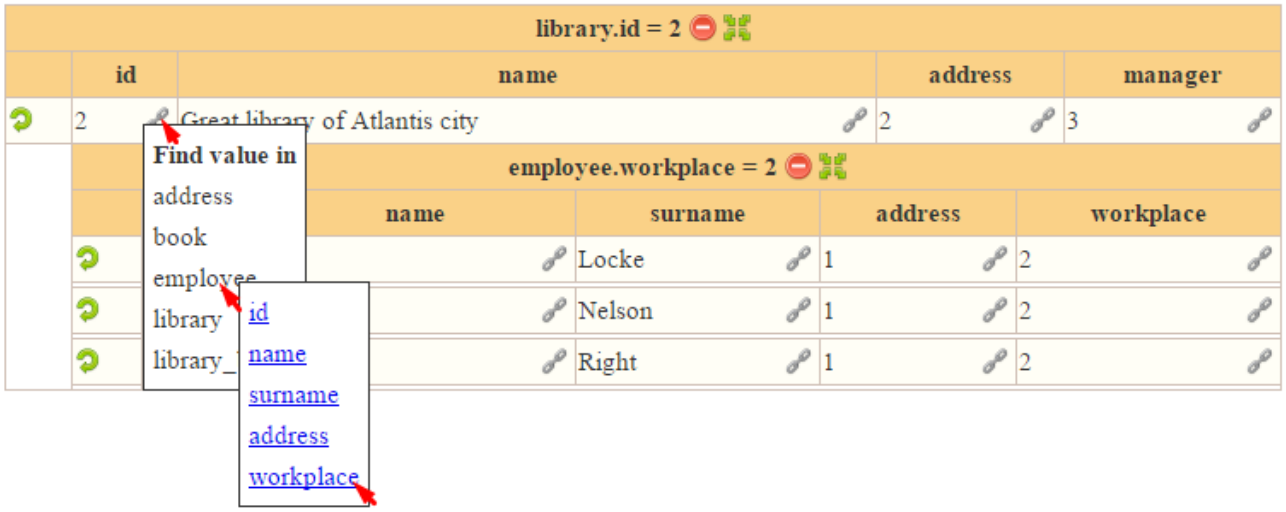



Diagram 9: Zobrazení záznamů pomocí univerzílních odkazů

Pokud se nám stane, že databáze vůbec neobsahuje cizí klíče, ať už nejsou podporované (např. u tabulek typu MyISAM), nebo jenom nejsou v databázi zadané, tak je možné nadefinovat v této komponentě vlastní cizí klíče. Uděláme to tak, že povolíme zobrazení univerzálních odkazů (třetí ikonka zprava v horním menu, s obrázkem řetězu a popiskem 'Toggle universal links' po najetí myši).

Pokud tuto vlastnost povolíme, u všech políček se nám zobrazí ikonka řetězu, po najetí na kterou se zobrazí seznam tabulek a sloupců, ve kterých budeme hledat odkazovanou hodnotu. Diagram výše zobrazuje postup pro zobrazení všech záznamů z tabulky 'empolyee', které mají ve sloupci 'workplace' hodnotu 2 (stejnou jako v políčku, ze kterého univerzální odkaz vychází).

Když odkaz stiskneme, komponenta se nás zeptá, zda chceme vybraný odkaz uložit jako cizí klíč. Pokud vybereme že ano, zeptá se, jestli se v jakém směru má ten cizí klíč uloží (zda původní tabulka odkazuje na vybranou, nebo zda vybraná odkazuje na původní)

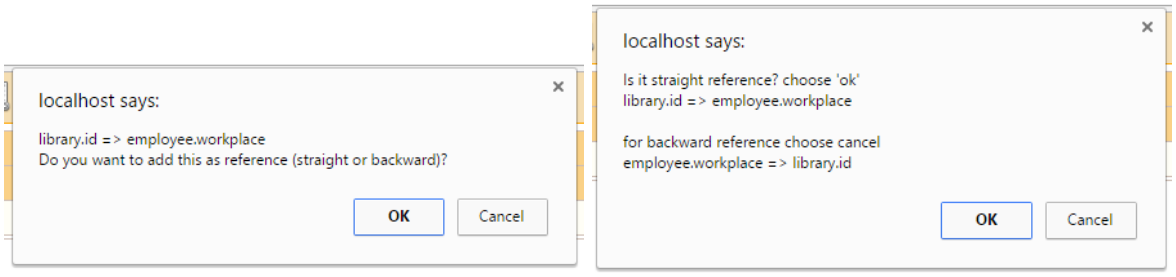



Diagram 10: Formulář přidání cizího klíče

Bohužel javascriptový dialog confirm neumožňuje zvolit si popisky tlačítek, tak musely být pro zvolení možností použity výchozí popisky 'ok' a 'cancel'.

**Seznam cizích klíčů** Další tlačítko v menu (tabulka s řetězem) zobrazí nebo skryje tabulku se seznamem cizích klíčů

| References | | | | |
|---|---|---|---|---|
| **id** | **table from** | **column from** | **table to** | **column to** |
| employee_ibfk_1 | employee | address | address | id |
| employee_ibfk_2 | employee | workplace | library | id |
| library_ibfk_1 | library | address | address | id |
| library_ibfk_2 | library | manager | employee | id |
| library_book_ibfk_1 | library_book | library | library | id |
| library_book_ibfk_2 | library_book | book | book | id |
| **Add new reference from:** address ▾ | id ▾ | **to:** address ▾ | id ▾ | Add |

Diagram 11: Seznam cizích klíčů

V této tabulce je přehled všech cizích klíčů které jsou v databázi i těch, které byly přidané uvnitř komponenty stromového pohledu. Kromě toho zápatí tabulky nabízí možnost vytváření nových cizích klíčů. Po vytvoření nového klíče se uvnitř stromové struktury objeví odkazy u příslušných políček.

**Uložení cizích klíčů** Poslední ikonka která je v horním menu (tabulka s disketou) slouží k uložení námi vytvořených cizích klíčů do databáze. Pro uložení se použije komponenta pro úpravu struktury databáze.

## 2.4 Komponenta pro porovnání dvou databází

Funkcí této komponenty je připojit se ke dvou databázím, porovnat jejich schéma, zobrazit v nich rozdíly a umožnit uživateli vybrané rozdíly sjednotit. Oproti původnímu zadání bylo zpracováno pouze porovnání struktury.

Prvním krokem je výběr databází, které chceme porovnávat. Vybírat můžeme z databází, které máme již připojené. Z toho vyplývá, že ještě před použitím tohoto nástroje musíme tyto databáze připojit (tlačítkem Add DB v levém menu).

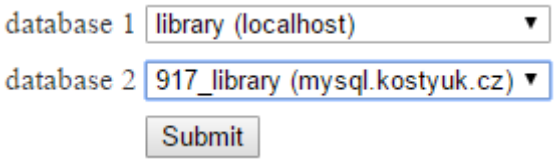



Diagram 12: Výběr databází k porovnání

### 2.4.1 Struktura porovnání

Až zvolíme databáze, které chceme porovnávat, zobrazí se nám porovnání jejich struktury

| library (localhost) | | | | | | | | 917_library (mysql.kostyuk.cz) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **PAIRED TABLES** | | | | | | | | | | | | |
| **address** | | | | | | | | **address** | | | | |
| id | integer | | Is AI | Not Null | | | | id | integer | | Is AI | Not Null |
| city | string | 150 | Not AI | Not Null | | | | city | string | 100 | Not AI | Not Null |
| street | string | 200 | Not AI | Is Null | | | | street | string | 200 | Not AI | Not Null |
| postCode | | | | | | | | | | | | |
| | | | | | | | | postal_code | | | | |
| **book** | | | | | | | | **book** | | | | |
| id | integer | | Is AI | Not Null | | | | id | integer | | Is AI | Not Null |
| name | string | 500 | Not AI | Not Null | | | | name | string | 100 | Not AI | Not Null |
| **library_book** | | | | | | | | **library_book** | | | | |
| library | integer | | Not AI | Not Null | | | | library | integer | | Not AI | Not Null |
| book | integer | | Not AI | Not Null | | | | book | integer | | Not AI | Not Null |
| **NON PAIRED TABLES** | | | | | | | | | | | | |
| **knihovna** | | | | | | | | | | | | |
| **zamestnanec** | | | | | | | | | | | | |
| | | | | | | | | **employee** | | | | |
| | | | | | | | | **library** | | | | |

Diagram 13: Výchozí přehled při porovnání struktury databází

Celá tabulka je horizontálně rozdělená do tří sloupců. Levý sloupec obsahuje informace o jedné databázi, pravý sloupec o druhé a uprostřed je sloupec s úpravami, které můžeme provádět.

Vertikálně je tabulka rozdělená na řádek s názvy databází a dále na dva bloky označené PAIRED TABLES a NON PAIRED TABLES. První blok obsahuje dvojice tabulek, které byly buď automaticky spárované dohromady. U každé dvojice tabulek je seznam jejich sloupců. Druhý blok obsahuje tabulky, kterým není přiřazena žádná z vedlejší databáze.

Na diagramu výše můžeme vidět nahoře pár tabulek address, které se tak jmenují v obou databázích. V něm vidíme seznam spárovaných sloupců 'id', 'city' a 'street' včetně toho, jak vypadá schéma těchto sloupců v jednotlivých databázích. Pro lepší přehled jsou shodné vlastnosti obou sloupců označeny zelenou barvou a rozdílné růžovou. Pod nimi vidíme v každé z tabulek jeden nespárovaný sloupec, v levé je to 'postCode', v pravé 'postal_code'.

Dále vidíme další dvojice tabulek ('book', 'library_book') a nakonec blok s tabulkami, které v jednotlivých databázích nejsou spárované. V levé databázi 'knihovna' a 'zamestnanec', v pravé 'employee' a 'library'.

### 2.4.2 Úpravy propojení a schématu

Tato komponenta nám nabízí dva typy nástrojů. První typ umožňuje upravovat propojení mezi jednotlivými tabulkami a sloupci. Druhý typ operací jsou operace nad strukturou sloupců.

**Úpravy propojení** V této kategorii máme čtyři možné operace: spojení dvou nespárovaných tabulek, rozpojení páru tabulek, spojení dvou nespárovaných sloupců a rozpojení páru sloupců. Tyto operace provádíme pomocí ikonek s oranžovými spojkami. Pro rozpojení páru tabulek je potřeba kliknout na ikonku spojek uprostřed řádku mezi názvy tabulek. Pro spojení dvou tabulek je potřeba najet myší na ikonku spojky vedle nespárované tabulky. Tím se zobrazí menu s možnými tabulkami ke spárování (seznam nespárovaných tabulek z protější databáze). Kliknutím na jednu z nich dojde k vytvoření nového propojení. Stejně tak je tomu i u rozpojování a propojování sloupců.

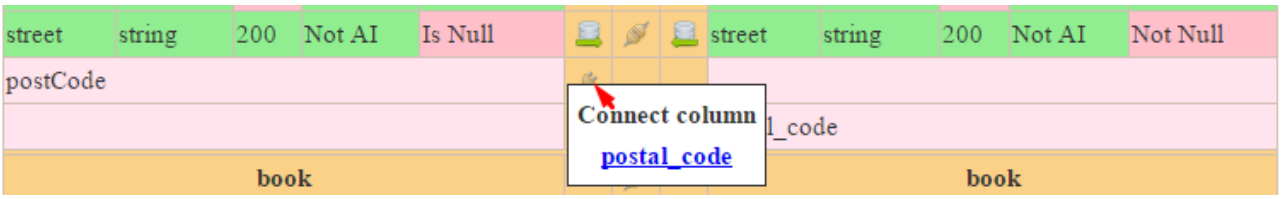



Diagram 14: Propojení sloupce postCode se sloupcem postal_code

**Úpravy schématu** Zde jsou podporované dvě operace: Nahrazení struktury v levé tabulce páru strukturou v pravé tabulce, nebo obráceně. Jedná se o ikonky databáze se šipkou vedle obou schémat sloupce. Kliknutím na tuto ikonku se změny provedou ve struktuře Database Image, která je uložena v session. Ke změně struktury skutečné databáze dojde až po uložení.

**Uložení** Poslední operací je uložení změn do skutečné databáze. Té docílíme stiskem tlačítka uložit u příslušné databáze v horním menu, které spustí komponentu pro ukládání změn

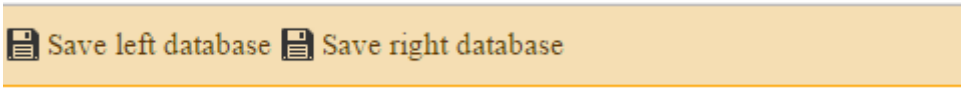



Diagram 15: Horní menu komponenty pro porovnání dvou databází

## 2.5 Komponenta pro ukládání změn ve struktuře ES Database do databáze

Přestože tato komponenta nebyla explicitně v zadání zmíněná, ostatní komponenty provádějí změny struktury Database Image a tak bylo potřeba vytvořit jednotný způsob pro ukládání změn do databáze. Jelikož každá instance Database Image, která je uložená v session, má svoje id, celý proces začíná předáním tohoto id metodě renderSaveSchema($id) třídy DatabasePresenter. Na tento modul nelze přejít přímo z menu, ale z ostatních modulů.

V prvním kroku se vytvoří ES Doctrine Saver, který je schopen porovnat strukturu ES Database se skutečnou databází. Pro zpracování změn předáme instanci třídy ES Doctrine Saver Visitor Logger, který všechny změny, které se mají provést, zapíše do vnitřního logu. Následně je z tohoto logu získáme a vypíšeme uživateli:

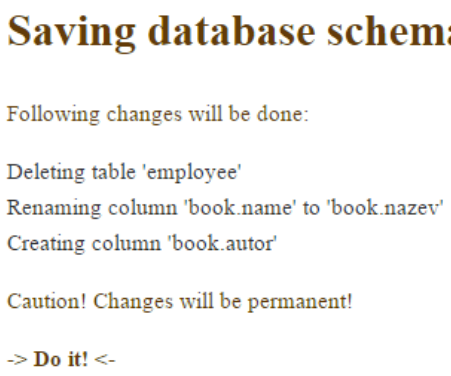



Diagram 16: Výpis změn k provedení

Zde si změny, které se mají provést, prohlédneme a pokud jsme s nimi spokojeni, necháme je provést. Tehdy dojde ke spuštění druhého kroku, který také vytvoří instanci ES Doctrine Saver, ale pro zpracování změn se tentokrát použije ES Doctrine Saver Visitor Database, který změny provede pomocí vrstvy Doctrine.

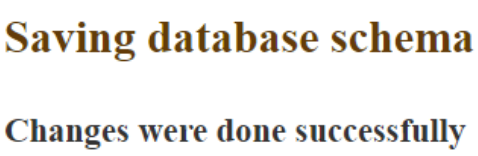



Diagram 17: Úspěšné provedení změn

Pokud během provádění změn nastane chyba, dostaneme chybovou hlášku od SQL serveru a vypíšeme ji uživateli

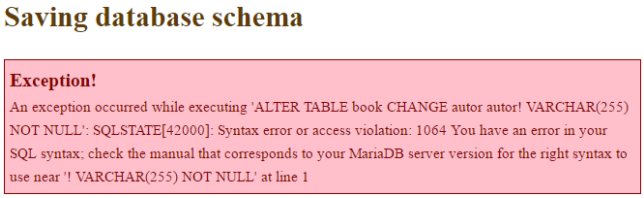



Diagram 18: Chybová hláška při přejmenování tabulky na název 'autor!'

# 3 Databázová abstrakce

Pokud chceme komunikovat s databázovým serverem, děláme to pomocí jazyku SQL. To znamená, že musíme ve své aplikaci vytvářet SQL dotazy. S tím ale souvisí některá úskalí. Jelikož výsledné dotazy obvykle závisí na vstupu uživatelů, může dojít k SQL injekci. Dále komunikace přímo pomocí SQL je v PHP nepraktická, především u práce se schématem databáze. Jelikož je PHP objektově orientovaný jazyk, je v něm přirozenější pracovat se schématem databáze jako s objekty. A všechny tyto problémy za nás řeší databázová vrstva DBAL (Doctrine database abstraction layer).

## 3.1 DBAL

Doctrine database abstraction layer je knihovna, která je součástí projektu Doctrine. Je napsaná v PHP a jejím úkolem je zařídit abstrakci komunikace mezi aplikací a databázovým serverem tak, že obsahuje sadu PHP tříd, na kterých voláme metody podle toho, co chceme, aby se s databází provedlo. DBAL se postará o korektní převod našich požadavků do jazyka SQL, včetně ochrany před SQL injekcí, jeho zavolání a dále zpracování výsledku dotazu, který vrátí opět jako PHP objekt. Zásadní výhodou knihovny DBAL a důvod pro její zvolení v této práci je její schopnost pracovat objektově se schématem databáze. Díky ní lze získat objekt, který obsahuje metody pro získání jednotlivých tabulek, sloupců nebo cizích klíčů a indexů. Každý z nich se vrací jako samostatný objekt, na kterém můžeme volat metody pro úpravu jeho struktury. Když provedeme na těchto objektech požadované změny, zavoláme metodu, která provedené změny uloží do databáze.

## 3.2 Editable Schema

Na rozdíl od Doctrine, která je knihovnou třetí strany, Editable schema je již součástí mé bakalářské práce. Jejím úkolem je přečíst schéma databáze s pomocí vrstvy DBAL a uložit je do struktury PHP objektů, které je možné (na rozdíl od struktury vrácené Doctrine) serializovat a deserializovat pro uložení do session a pro práci mezi více dotazy. Dále je její účelem poskytnout rozhraní pro editaci této struktury, která je přiměřená potřebám tohoto projektu. Posledním úkolem této struktury je porovnat schéma, které je v ní uloženo s aktuálním schématem databáze a s pomocí vrstvy DBAL provést úpravu schématu skutečné databáze do podoby, která je uložena ve struktuře Editable Schema.

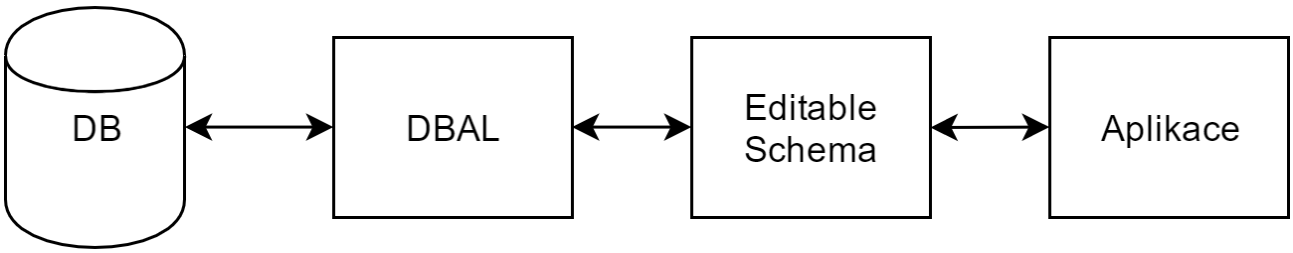



Diagram 19: Struktura abstrakce schématu databáze pomocí vrstvy DBAL a Editable Schema

**Pozn:** Zde bych chtěl upozornit na rozdíl mezi termíny struktura (ať už struktura Editable Schema nebo struktura Doctrine) a vrstva (Editable Schema nebo Doctrine). Jak ES tak DBAL vytvářejí abstrakci databáze a ukládají ji do vlastní struktury (sada provázaných objektů, reprezentujících databázi, tabulky, sloupce a cizí klíče). Kromě toho poskytují třídy pro práci s takovou strukturou. Všechny takové třídy společně se třídami struktury vytvářejí vrstvu pro práci s databází.

### 3.2.1 Životní cyklus struktury Editable Schema

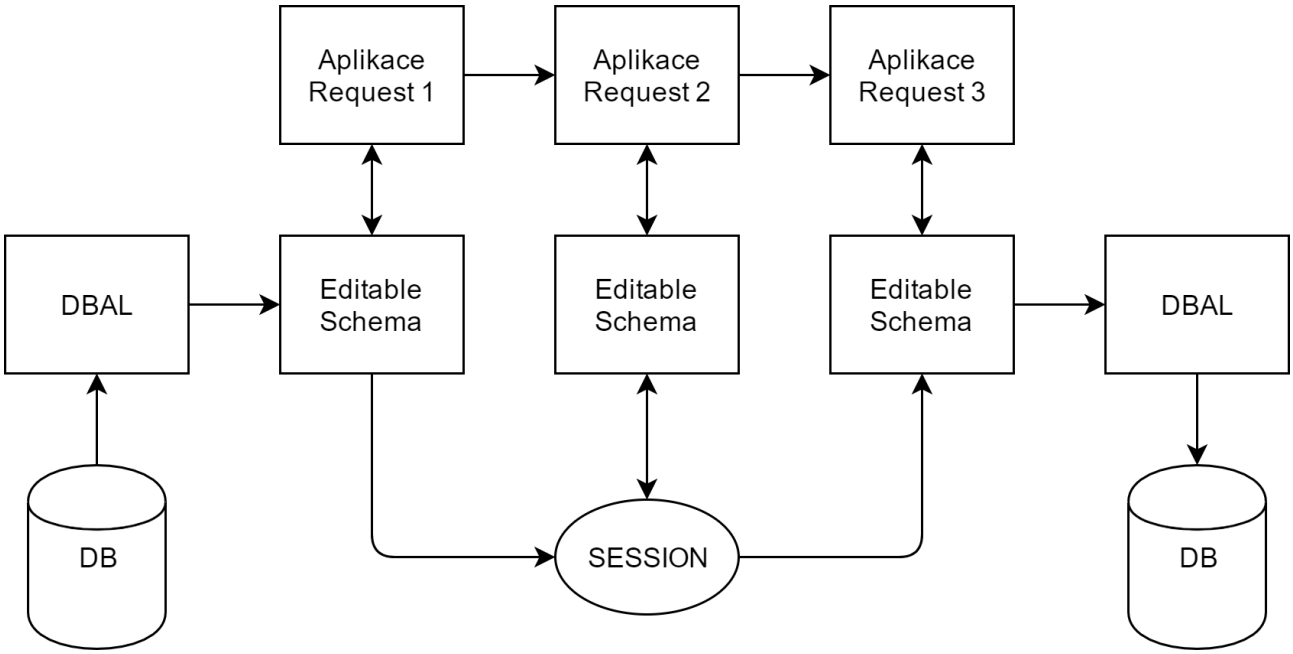



Diagram 20: Životní cyklus vrstvy Editable Schema

Pokud chceme v nějakém místě v aplikaci pracovat s databázovým schématem pomocí vrstvy Editable Schema (na diagramu 20 je to během požadavku 'Aplikace, Request 1') musíme nejprve vytvořit instanci třídy `EditableSchemaDatabase` (podrobný proces vytvoření je popsán dále v dokumentaci v kapitole 3.2.4). Pro vytvoření této třídy potřebujeme napojení na databázi pomocí vrstvy DBAL. Díky DBAL se napojíme na databázi a postupně přečteme, jaké obsahuje tabulky, sloupce a cizí klíče a na základě toho vytvoříme strukturu uvnitř instance třídy `EditableSchemaDatabase`, která zobecňuje schéma databáze. Na konci požadavku zavoláme na této instanci metodu pro uložení do session. Zde dojde k serializaci databázového schématu a vygenerování id, pod kterým bude v session přístupná (dále id schématu). Tato metoda vrátí přístupové id a požadavek musí zajistit, že se toto id dostane dalšímu požadavku.

Jakýkoliv požadavek, který získá id schématu (zde 'Aplikace, Request 2 a 3'), může požádat vrstvu Editable Schema o vytvoření struktury uložené pod odpovídajícím id. Vrstva Editable Schema načte serializované schéma ze sessionF a provede jeho deserializaci na instanci `EditableSchemaDatabase`. Poté již může požadavek pomocí metod na této instanci upravovat schéma databáze.

Požadavek, který se nakonec rozhodne uložit změny schématu do skutečné databáze (zde 'Aplikace, Request 3') musí nejprve pomocí vrstvy DBAL vytvořit připojení k této databázi. Poté zavolá na instanci `EditableSchemaDatabase` metodu pro uložení do databáze a předá jí toto připojení na databázi. Editable Schema pak s pomocí vrstvy DBAL zkontroluje, čím se schéma reálné databáze liší od té, které chceme docílit a na těchto rozdílech provede úpravy.

### 3.2.2 Struktura tříd schématu databáze vrstvy Editable Schema

Vrstva Editable Schema se skládá z několika PHP tříd. Ty, které se týkají schématu databáze, popíšu v této kapitole. V zájmu přehlednosti budu názvy tříd psát s mezerami, přestože ve skutečnosti názvy tříd v PHP mezery obsahovat nesmějí. Dále pak předponu EditableSchema budu zkracovat na ES. Tedy třída zde označená jako `ES Foreign Key` je ve skutečnosti třída `EditableSchemaForeignKey`.

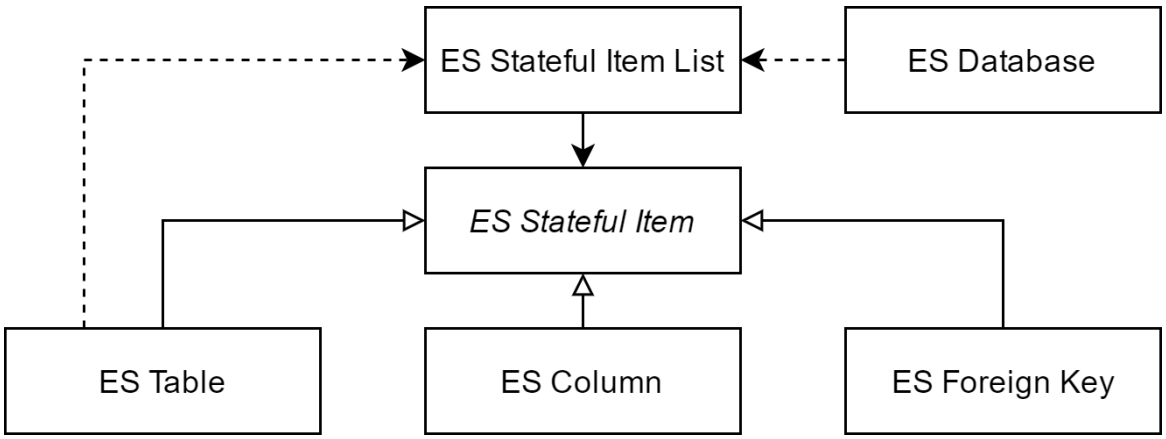



Diagram 21: Hierarchie tříd vrstvy Editable Schema

**ES Stateful Item** Je základním kamenem struktury pro uložení schématu databáze a je abstraktní třídou. Je rodičem pro třídy, které nesou informaci o jakémkoliv prvku databáze, který můžeme pomocí vrstvy ES upravovat. V rámci této práce se jedná o tabulky, sloupce a cizí klíče.

Třída `ES Stateful Item` v sobě uchovává identifikátor prvku, podle kterého jej můžeme dohledat v databázi (např. původní název tabulky). Ten se předá při vytvoření instance a je neměnný. Dále uchovává název prvku, který je na rozdíl od identifikátoru editovatelný (Pokud tedy z databáze načteme tabulku `book` a přejmenujeme ji na `kniha`, identifikátor zůstane `book`, abychom věděli, schéma které tabulky z databáze máme editovat, a název bude `kniha`, na kterou tuto tabulku při uložení přejmenujeme).

Dále obsahuje třída `ES Stateful Item` informace o stavu prvku. Ten je dán dvěma proměnnými, `new` a `deleted`. Proměnná `new` značí, zda daný prvek existuje v původní databázi, nebo zda je nově přidaný až úpravami ve vrstvě ES. Podle toho se k tomu prvku také budeme chovat při ukládání do databáze, buď jej upravíme, nebo přidáme. Proměnná `deleted` slouží k označení, zda byl prvek během práce ve vrstvě ES odstraněn. Výhodou tohoto přístupu oproti opravdovému smazání z ES je, že pokud se rozhodneme tuto změnu vrátit, nemusíme se obracet na databázi, abychom zjistili, jak takový prvek vypadal před smazáním. Další výhodou je to, že až dojde k uložení databáze, bude mít vrstva ES k dispozici seznam prvků, které má smazat a nebude muset porovnávat reálnou databázi s vlastním schématem aby zjistila, které tabulky v databázi přebývají.

**ES Stateful Item List** Jedná se o třídu, která obsahuje kolekci objektů `ES Stateful Item` a nabízí metody pro práci s nimi. Na rozdíl od standardních kolekcí, je tato optimalizovaná pro práci s `ES Stateful Item` a neobsahuje všechny CRUD operace. Operaci `Create` samozřejmě obsahuje, je tedy možné do kolekce vkládat nové prvky. Co se operace `Read` týká, nabízí jak možnost vrátit pole s referencemi na všechny objekty uvnitř kolekce, tak vyhledávání buď podle identifikátoru, nebo podle názvu. Operace `Update` ani `Delete` tato kolekce neobsahuje. `Update` neobsahuje proto, že editaci prvku je možné provést na referenci, kterou získáme při operaci `Read` a `Delete` neobsahuje proto, že u kolekcí `ES Stateful Item` nemá docházet o odstaňování prvků, ale pouze k jejich označení za `deleted`.

**ES Table** Potomek třídy `ES Stateful Item`. Nese informace o tabulce v databázi. Kromě identifikátoru a názvu obsahuje navíc kolekci sloupců (`ES Stateful Item List` obsahující objekty typu `ES Column`, viz dále).

**ES Column** Potomek třídy `ES Stateful Item`. Zobrazuje jeden sloupec databáze. O sloupci uchovává následující informace: Identifikátor, název, datový typ, délku, notnull a autoincrement (terminologie z Doctrine, označuje sloupce s primárním klíčem tabulky a s automatickým čítačem).

**ES Foreign Key** Poslední potomek třídy `ES Stateful Item`. Reprezentuje cizí klíč z výchozí tabulky do odkazované. Obě tabulky jsou zde uložené jako reference na třídu `ES Table`. Dále obsahuje pole sloupců z výchozí tabulky a pole sloupců tabulky odkazované. Který ze sloupců z prvního pole odkazuje na který sloupec z druhého je dáno jejich pořadím v poli. Pozor, v tomto případě už se nejedná o kolekci typu `ES Stateful Item List` ale o obyčejné PHP pole, které je indexováno čísly.

**ES Database** Jedná se o třídu představující databázi. Obsahuje pouze kolekci tabulek a kolekci cizích klíčů. Obě jsou typu `ES Staful Item List` a `ES Database` nabízí vlastní metody pro jejich správu, které volají metody na kolekci a následně se starají o přetypování `ES Stateful Item` z kolekce na `ES Table` nebo `ES Foreign Key`, podle toho o kterou kolekci se zrovna jedná.

### 3.2.3 Integrace třídy ES Database a session

Schopnost uložit celou strukturu editované databáze do session je hlavní předností vrstvy Editable Schema oproti vrstvě Doctrine. Právě kvůli této vlastnosti byla vytvořena a je to vlastnost, která se používá v obou modulech, které jsou součástí této práce, jelikož oba pracují s databází napříč více HTTP požadavky, které potřebují přistupovat ke stejnému objektu databáze a úpravy, které jednotlivé požadavky provedou se musí zachovat a přenést do dalšího požadavku.

Celý proces probíhá v několika krocích:

1. Export `ES Database` do obyčejného pole
2. Uložení pole do session
3. Načtení pole ze session
4. Rekonstrukce `ES Database` z pole

**Export ES Database do obyčejného pole** O export do obyčejného PHP pole se stará třída `ES Array Saver`. Obsahuje metody pro export prvků ES do pole. Vstupním bodem je metoda `saveDatabase`, která přebírá jako argument objekt `ES Database`. Tato metoda zavolá pro každou existující tabulku metodu `saveTable`, která uloží do pole informace o identifikátoru a o názvu tabulky a dále o zavolá metodu `saveColumn` pro jednotlivé sloupce tabulky a získá tak informace o každém sloupci. Po uložení informací o tabulkách projde metoda `saveDatabase` stejným způsobem všechny cizí klíče a uloží u nich informaci o identifikátoru referenční a referované tabulky a identifikátory příslušných sloupců. Identifikátory sloupců jsou uloženy jako dvě pole indexované čísly, pole referenčních a referovaných sloupců. Obě pole mají stejnou délku. Který sloupec z jednoho pole patří ke kterému poznáme podle stejných indexů v těchto polích.

**Uložení pole do session a načtení ze session** Uložení a načtení ze session není součástí vrstvy Editable Schema. V prvotních návrzích vrstvy ES bylo, že `ES database` bude obsahovat metody `toArray`, `fromArray` a dále metody `saveToSession`, `loadFromSession` které první dvě metody používají. Nicméně společně s další funkcionalitou pro načítání a ukládání do databáze tyto metody znepřehledňovaly kód vrstvy Editable Schema tím, že do třídy `ES Database` byla umístěna funkcionalita pro ukládání struktury, export/import do pole, práci se session a práci s databází. Naproti tomu ostatní prvky ES (tabulky, sloupce a cizí klíče) plnily pouze funkcionalitu ukládání struktury. Proto jsem se rozhodl ve třídě `ES Database` nechat pouze funkcionalitu pro ukládání struktury, a pro všechny ostatní funkcionality jsem vytvořil samostatné třídy.

**Rekonstrukce ES Database z pole** O rekonstrukci třídy `ES Database` se stará třída `ES Array Loader`, která obsahuje metodu `loadDatabase`, která vezme pole a vrátí instanci `ES Database`. Jelikož třídy `ES Database`, `ES Table` a `ES Column` neobsahují odkazy na nadřazené třídy (tedy `ES Column` neodkazuje na `ES Table` a ta neodkazuje na `ES Database`), stačí pro vytvoření těchto prvků předat pouze pole s příslušnými daty. Oproti tomu třída `ES Foreign Key` obsahuje odkazy na objekty `ES Table` a `ES Column`. Proto je při vytváření cizích klíčů potřeba předat kromě pole s názvy tabulek a sloupců i odkaz na objekt `ES Database`, ve kterém je možné tyto tabulky a sloupce dohledat. Z toho důvodu se při reprodukci databáze z pole musí nejprve načíst tabulky a sloupce, a pak až cizí klíče, které na ně budou odkazovat.

### 3.2.4 Integrace třídy ES Database a Doctrine

Integrace Vrstvy ES a Doctrine je komplexnější záležitostí než integrace se session. Opět zde máme dvě třídy které se starají o načtení a uložení, a to `ES Doctrine Loader` a `ES Doctrine Saver`. Nicméně `ES Doctrine Saver` sám o sobě neprovádí ukládání, ale pouze vyhledává změny a provádění změn deleguje na třídu, která implementuje rozhraní pro ukládání změn, viz dále.

**Načtení ES Database z Doctrine** Pro načtení struktury skutečné databáze je použita třída `ES Doctrine Loader`. Obsahuje metodu `extractDatabase`, která jako argument přebírá třídu s rozhraním `AbstractSchemaManager`. `AbstractSchemaManager` je rozhraní vrstvy Doctrine, které je navržené pro čtení a úpravu schématu databáze.

Samotné vytváření instance třídy `ES Database` se velice podobá Rekonstrukci `ES Database` z pole (viz předchozí kapitola 3.2.3). Nejdříve dojde k vytvoření prázdné instance `ES Database`, pak z projdeme všechny tabulky v `AbstractSchemaManager`, předáme je metodě `extractTable`, která vytvoří instanci `ES Table` a ty vložíme do prázdné databáze. A to samé uděláme i pro cizí klíče. Tady stojí za to zmínit, že vrstva ES a Doctrine ukládají cizí klíče v trochu odlišných formátech. Cizí klíč v Doctrine je objektem uvnitř objektu tabulky a nenese žádné informace o referenční tabulce, jen o referované tabulce a o sloupcích obou tabulek. Referenční tabulka je dána objektem tabulky, kterému cizí klíč patří. Oproti tomu ve vrstvě ES jsou cizí klíče uloženy přímo v objektu `ES Database` a obsahují odkazy na obě tabulky (referenční a referovanou) a na jejich sloupce.

**Uložení ES Database do Doctrine** Za uložení struktury ES do databáze pomocí vrstvy Doctrine se stará třída `ES Doctrine Saver`. Tato třída obsahuje metodu `saveDatabase`, která jako argument vyžaduje objekt `DatabaseImage` (jedná se obalovou třídu, která v sobě obsahuje jak instanci `ES Database`, tak Doctrine připojení k databázi. Třídě `DatabaseImage` je věnována kapitola 3.3). Ukládání databáze probíhá v následujících krocích:

1. Odstranění objektů k odstranění (počínaje cizími klíči)
2. Úprava struktury sloupců v tabulkách
3. Přejmenování prvků k přejmenování
4. Vytvoření nových prvků (konče cizími klíči)

Změny se provádějí postupně jedna za druhou ve výše zmíněném pořadí, nikoliv jako jedna změna. Z toho vyplývá, že v případě, že dojde k chybě v některé ze změn, tak již provedené změny zůstanou zachované. Nicméně následující změny se provádět nebudou, aby nedošlo k tomu, že kvůli jedné chybě nastanou další (např. aby se nestalo, že pokud přejmenujeme tabulku A na B, tak abychom se později nepokusili přidávat sloupce do tabulky B).

Odstranění probíhá jako první operace. Nejdříve odstraníme cizí klíče, jelikož mají návaznosti na tabulky, u kterých nevíme, co se s nimi dále bude dít. Např. pokud budeme chtít odstranit tabulku a cizí klíč, který na ní odkazoval, tak pokud se nejprve pokusíme odstranit tabulku, dostaneme SQL chybu, že porušujeme integritu cizího klíče. V případě, že nejprve odstraníme cizí klíč už při odstraňování tabulky nebudeme mít tyto potíže. Dalším důvodem, proč nejdříve odstraňujeme je pro uvolnění názvů. Např. pokud budeme chtít z databáze odstranit jednu tabulku a vytvořit místo ní jinou se stejným názvem, SQL nám nedovolí nejdříve vytvořit novou, kvůli duplicitě názvů.

Úprava struktury sloupců probíhá jako následující operace po odstranění. Jedná se pouze o změnu schématu sloupců, nikoliv názvů sloupců. Tato operace nemá žádné návaznosti na jiné operace, tedy při ní nemůže dojít ke kolizi s jinou operací (Jelikož neměníme názvy, nemůže dojít ke kolizi v názvech). Jedinou možnou kolizí by mohlo být, pokud by se měl zároveň i měnit cizí klíč odkazující na tento sloupec. Ve vrstvě ES ale k úpravě cizích klíčů nedochází. Pokud má dojít ke změně cizího klíče, provede se to odstraněním starého a přidáním nového klíče. Na rozdíl od podobného postupu u ostatních prvků databáze (např. úpravě sloupce tím že bychom jej odstranili a přidali nový, během čehož by došlo k permanentní ztrátě dat), u cizích klíčů ke ztrátě dat nedojde.

Přejmenování je jedinou operací, kde může dojít ke kolizi. V případě že například přejmenujeme tabulku A na B a existující tabulku B na A, dojde ke kolizi názvů a k vyhození výjimky vrstvou Doctrine. Další zajímavostí tohoto kroku je, že ve všech operacích před ním se na prvky databáze odkazujeme pomocí 'identifikátoru' (tím je v struktuře ES označen název prvku v původní databázi) a po tomto kroku se už na prvky odkazujeme pomocí 'názvu' (to je v ES název prvku tak, jak ho chceme mít po úpravách).

Vytvoření je poslední úpravou nad databází. Nejprve vytvoříme nové tabulky, poté sloupce a nakonec cizí klíče. Vytváření je poslední operací, jelikož na nové tabulky a sloupce nemůže odkazovat žádný původní cizí klíč (jak již bylo řečeno, úpravy cizích klíčů provádíme smazáním a novým vytvořením, takže takový klíč by byl v klíčích pro vytvoření), a nové cizí klíče jsou vytvořeny až nakonec po vytvoření nových tabulek a sloupců.

**ES Doctrine Saver Visitor** Je rozhraní pro zpracovánání jednotlivých změn v databázi. Přestože se rozhraní jmenuje Visitor, jedná se ve skutečnosti o rozhraní pro návrhový vzor Strategy. Jde o chybu v názvu, která vznikla na začátku návrhu a když jsem si jí uvědomil, byla už bohužel implementována na více místech v kódu. Rozhraní deklaruje metody pro zpracování jednotlivých změn nad databází zmíněných výše, jako například metody `deleteTable` nebo `createForeignKey`. Třída `ES Doctrine Saver` při vytváření vyžaduje v konstruktoru objekt s tímto rozhraním a stará se pouze o nalezení změn, které mají být v databázi provedeny. Samotné provedení změn deleguje `ES Doctrine Saver` na zmíněný objekt s rozhraním `ES Doctrine Saver Visitor`. Tohle řešení bylo navrženo proto, aby bylo možné nejprve zobrazit a zkontrolovat změny, které se mají provést, a až poté je opravdu provést. V rámci této práce jsou implementovány dvě třídy s tímto rozhraním

**ES Doctrine Saver Visitor Logger** Tato implementace rozhraní `ES Doctrine Saver Visitor` neprovádí žádné úpravy v databázi, ale zaznamenává změny, které by se měly provést, do vnitřního logu. Kromě všech metod vyžadovaných rozhraním, obsahuje ještě navíc metodu `getLog`, která vrací pole textových řetězců popisujících úpravy k provedení. Příkladem takového záznamu může být třeba 'Deleting column book.note', který znamená, že v tabulce `book` má dojít k odstranění sloupce `note`.

**ES Doctrine Saver Visitor Database** Oproti předchozí třídě, tato třída již provádí jednotlivé úpravy na skutečné databázi (pomocí abstrakce Doctrine). Mezi změnami v ES a Doctrine je určitý rozdíl a tato třída se stará a kompatibilní provedení ES změn v Doctrine vrstvě. Příkladem může být například úprava struktury sloupce. Zatímco v ES je úprava sloupce samostatná operace, v Doctrine jde se jedná o jednu operaci změny struktury tabulky, během které můžeme zároveň přidávat, odstraňovat a přejmenovávat více sloupců zároveň. Dalším příkladem je například rozdíl v práci s cizími klíči, jak bylo popsáno na začátku této kapitoly.

### 3.3 Database Image

Třída `Database Image` reprezentuje konkrétní existující databázi, na rozdíl od třídy `ES Database`, která reprezentuje pouze strukturu databáze, bez jakékoliv informace o tom, o jakou databázi se jedná. Třída je to velmi jednoduchá. Konstruktor třídy vyžaduje objekt typu `ConnectionInfo`, který obsahuje údaje pro připojení ke konkrétní databázi (host, username, password a database). Jakmile tyto údaje dostane, uloží si je, a čeká na další požadavky. Tato třída nabízí rozhraní pro získání DBAL připojení a `ES Database`.

Pokud požádáme o připojení k databázi (pomocí metody `getConnection`), tak se objekt `Database Image` podívá, zda jsme jej již o to dříve žádali. Pokud ano, tak je otevřené připojení již uložené v `Database Image` a objekt nám jej vrátí. Pokud je to poprvé, co žádáme o toto připojení, tak dojde k pokusu o připojení se k databázi, a pokud se podaří, tak si toto připojení uloží pro příští použití, a zároveň toto připojení vrátí.

Stejně je tomu i u metody `getEditableSchema`, která vrací třídu typu `ES Database`. Zde pokud schéma databáze ještě nebylo načteno, tak dojde v vytvoření Doctrine připojení a z něj pomocí třídy `ES Doctrine Loader` k načtení schématu databáze do struktury ES.

**Ukládání do session** Je to velice důležitý aspekt třídy `Database Image`. Zatímco Doctrine připojení k databázi není možné serializovat, `ES Database` serializovatelné je (pomocí konverze do pole a rekonstrukce z pole). Dále je možné serializovat `Connection Info`, jelikož se jedná pouze o několik textových řetězců. Při ukládání objektu `Database Image` tedy dochází k serializaci `Connection Info` a `ES Database`, v případě že `ES Database` již byla načtena. Pokud nebyla, uloží se jen `Conncection Info`. Doctrine connection se neukládá, ale po rekonstrukci objektu ze session je možné jej opět vytvořit na základě informací v `Connection Info`.

Tím se dostáváme k zajímavé vlastnosti `Database Image`, která jí umožňuje být používána jako struktura pro práci se strukturou databáze. Jedná se o to, že pro práci se strukturou databáze se stačí pouze jednou připojit k dané databázi, načíst její strukturu do `ES Database` a uložit do session. V dalších dotazech které editují schéma databáze již není potřeba se k databázi připojovat, protože celá struktura je načtena ze session. Změny se na konci dotazu opět uloží do session a je možné tak provádět úpravy nad schématem jedné databáze přes více dotazů. Nakonec až se rozhodnome naše úpravy uložit do skutečné databáze, budeme kromě nové struktury která je uložená v `ES Database` potřebovat i Doctrine připojení k databázi, a to je možné vytvořit z dat, které jsou opět uloženy v session, a to ze třídy `Connection Info`.

`Database Image` je tedy přizpůsobená k tomu, aby si uložila schéma databáze do session, pracovala s ním bez napojení na databázi a na konci vše opět uložila do příslušné databáze.

## 3.4 Možnost spravovat více druhů databází

Jedním z úkolů v zadání této bakalářské práce bylo zjistit, zda je možné upravit tuto aplikaci tak aby jí bylo možno spravovat více různých typů databází.

Připojení k různým typům databází je omezeno dvěma místy. Za prvé se jedná o možnosti, které nabízí vrstva Doctrine, a dále na tom, jaké z těchto možností jsou implementované konkrétně v této aplikaci

Jelikož Doctrine funguje na databázové vrstvě PDO, může se připojovat k těm databázím, které mají PDO ovladače. Kromě PDO podporuje i několik dalších možností připojení, viz úryvek z Doctrine dokumentace[1] zobrazující výpis podporovaných ovladačů:

1. pdo_mysql: A MySQL driver that uses the pdo_mysql PDO extension.
2. drizzle_pdo_mysql: A Drizzle driver that uses pdo_mysql PDO extension.
3. mysqli: A MySQL driver that uses the mysqli extension.
4. pdo_sqlite: An SQLite driver that uses the pdo_sqlite PDO extension.
5. pdo_pgsql: A PostgreSQL driver that uses the pdo_pgsql PDO extension.
6. pdo_oci: An Oracle driver that uses the pdo_oci PDO extension. Note that this driver caused problems in our tests. Prefer the oci8 driver if possible.
7. pdo_sqlsrv: A Microsoft SQL Server driver that uses pdo_sqlsrv PDO Note that this driver caused problems in our tests. Prefer the sqlsrv driver if possible.
8. sqlsrv: A Microsoft SQL Server driver that uses the sqlsrv PHP extension.
9. oci8: An Oracle driver that uses the oci8 PHP extension.
10. sqlanywhere: A SAP Sybase SQL Anywhere driver that uses the sqlanywhere PHP extension.

Nicméně v této práci je implementované pouze připojení k MySQL databázím, za použití ovladače pdo_mysql.

Podporu pro ostatní typy databází je možné v budoucnu přidat předěláním třídy `ConnectionInfo` na abstraktní třídu nebo rozhraní a rozšířením této třídy o metodu pro vytvoření připojení. Dále pro každý požadovaný typ databáze implementovat vlastního potomka třídy `ConnectionInfo`. Po takové úpravě by bylo možné používat další typy databází ve stávající aplikaci pouze s menšími změnami.

[1] Doctrine DBAL 2 documentation: Configuration, kapitola 3.1.2 [online]. [cit. 2017-05-24]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html

# 4 Závěr

Bakalářská práce byla vypracována dle zadání a obsahuje komponenty, které jsou v zadání uvedeny.

## 4.1 Srovnání s existujícími řešeními

Jelikož námětem pro tuto práci pro mne byly aplikace Adminer a phpMyAdmin, přidám na závěr i porovnání s těmito aplikacemi.

Hlavní rozdíl mezi tímto projektem a zmíněnými dvěma řešeními je ve způsobu přístupu k databázím. Zatímco Adminer a phpMyAdmine jsou určení k rychlým menším úpravám v databázi (vytvoření tabulky, vyfiltrování dat) systém v této práci se zaměřuje především na komplexnější operace. Zatímco existující systémy provádějí úpravy přímo v databázi, tento systém provádí většinu operací ve struktuře uložené v session a k databázi se obrací přes dvě abstraktní vrstvy (Doctrine a Editable Schema). Kvůli tomu je prvotní spouštění některých komponent poměrně zdlouhavé, jelikož musí dojít k načtení celé struktury přes tyto dvě abstraktní vrstvy. Na druhou stranu po načtení schématu do session je práce s takovým schématem rychlejší.

Co se týká standardních operací pro správu databáze, jsou nástroje v Admineru a phpMyAdminu propracovanější než v tomto projektu. Obsahují pokročilejší nástroje např. pro správu cizích klíčů, inexů nad tabulkami nebo triggery.

Naproti tomu tento projekt obsahuje nástroje, které jsou pro práci s databází velice užitečné, ale zmíněné systémy je vůbec neobsahují. To jsou právě komponenta pro stromové procházení dat a komponenta pro porovnání struktury dvou databází. Obě z nich zjednodušují úkoly, se kterými se při správě databáze nebo při vývoji databázové aplikace běžně setkáváme.

## 4.2 Instalace

Instalaci provedeme rozbalením složky src.zip a umístěním na webový server. Server musí povolovat .htaccess soubory a také musí mít povolený mod_rewrite.

Spuštění aplikace probíhá otevřením složky www umístěné v kořenovém adresáři projektu.

# 5 Použité zdroje a seznam příloh

## Použité zdroje


1. Framework Nette [online]. [cit. 2017-05-25]. Dostupné z: https://nette.org/cs/
2. Doctrine DBAL 2 documentation [online]. [cit. 2017-05-25]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/index.html
3. Dokumentace Nette 2.4 [online]. [cit. 2017-05-25]. Dostupné z: https://doc.nette.org/cs/2.4/
4. JQuery API Documentation [online]. [cit. 2017-05-25]. Dostupné z: https://api.jquery.com/


## Seznam příloh

K této bakalářské práci patří následující soubory:

1. src.zip - obsahuje zdrojový kód projektu
2. PHP_documentation.zip - obsahuje automaticky generovanou HTML dokumentaci PHP tříd
3. JS_documentation.zip - obsahuje automaticky generovanou HTML dokumentaci javascript funkcí
4. report.zip - obsahuje zdrojový kód tohoto dokumentu v jazyce LaTeX

**Pozn.:** Automaticky generovaná dokumentace je generovaná z celého projektu, ne jen z té části, kterou jsem vypracoval já. Obsahuje tedy jak třídy související s komponentami pro práci s klasickým editorem struktury, stromového procházení záznamů, porovnávání databází a ukládání změn ve struktuře databáze, které jsem vypracoval já, tak i třídy související s komponentami práce s daty tabulek, voláním přímých SQL dotazů a grafického editoru schématu databáze, které v rámci své bakalářské práce vypracovala Kseniia Razina.