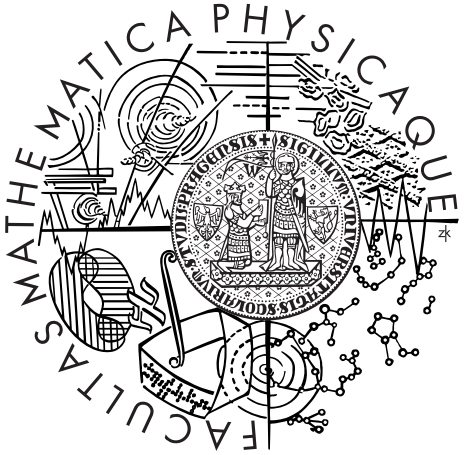
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Richard Ejem

Framework pro práci s relační databází se stromovou strukturou v jazyce PHP

Středisko informatické sítě a laboratoří (SISAL)

Vedoucí bakalářské práce: RNDr. Libor Forst
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2013

Poděkování.

RNDr. Liboru Forstovi za vedení této práce a konzultace.

Jakubu Stuchlíkovi a Adamu Kadeřávkovi za poskytnutí materiálů pro ukázkovou aplikaci `prekazky.cz`.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne ..... Podpis autora

Název práce: Framework pro práci s relační databází se stromovou strukturou v jazyce PHP

Autor: Richard Ejem

Katedra: Středisko informatické sítě a laboratoří (SISAL)

Vedoucí bakalářské práce: RNDr. Libor Forst, Středisko informatické sítě a laboratoří (SISAL)

Abstrakt:

Tato práce se zabývá tvorbou frameworku, který umožní efektivně vytvářet webové databázové aplikace v jazyce PHP. Klíčovým prvkem tohoto frameworku je možnost vygenerovat webové uživatelské rozhraní z datového modelu s minimální nutností psaní dodatečného kódu. U existujících frameworků jsme nenašli takto komplexní řešení, většinou se zabývají pouze částí problematiky (robustní modelová vrstva či webové rozhraní) nebo neposkytují plně implicitní propojení těchto vrstev.

Tento framework využívá některé osvědčené praktiky a návrhové vzory existujících frameworků, a navíc poskytuje nové nástroje usnadňující vývoj. Hlavními přínosy jsou zmíněné automaticky připravené webové rozhraní a objektový model pro snadnou strojovou tvorbu a úpravu SQL dotazů.

Framework je založen na myšlence *konvence před konfigurací*, což znamená, že programátor používající tento framework nemusí vytvářet podrobný kód popisující standardní chování aplikace. K vytvoření funkční databázové aplikace stačí jen načíst framework, který nabízí standardní funkcionalitu pokrývající celou problematiku databázové aplikace. Programátor pak pouze přizpůsobí ty části aplikace, u kterých vyžaduje jiné než standardní chování.

Klíčová slova: PHP, framework relační databáze, objektově-relační mapování, SQL, ORM, CRUD, webové uživatelské rozhraní

Title: PHP framework for working with relational databases with logical tree structure in PHP

Author: Richard Ejem

Department: Network and Labs Management Center

Supervisor: RNDr. Libor Forst, Network and Labs Management Center

Abstract:

In this thesis we have developed a framework which allows programmers to effectively create web database applications with PHP. The key advantage of this framework is that it can create web user interface from data model with almost no need of writing additional code. We have not found such complex solution in other existing frameworks. They usually deal only with a part of this problem (robust model layer or web interface) or do not implicitly link these layers.

This framework uses some time-proven practices and design paterns used in existing frameworks. In addition, it provides some new tools and techniques making web application development easier. Main advantages are generated web user interface and object model to easily create and edit SQL queries by machine.

Main concept of this framework is *convention over configuration*, which means that programmer using this framework does not have to write detailed code describing standard application behaviour. Loading the framework is almost sufficient to create working web database application. Programmer then only customizes the parts of application which need other than default behaviour.

Keywords: PHP, relational database framework, object-relation mapping, SQL, ORM, CRUD, web user interface

Obsah

Úvod

Existuje mnoho různých frameworků a databázových knihoven pro PHP. Doposud ale chyběl takový framework, který by skloubil celou problematiku webové databázové aplikace od datového modelu až po webové rozhraní, a navíc zjednodušil vývoj kostry aplikace pouze na vytvoření struktury relační databáze a stručné zkonfigurování frameworku.

Náš framework tyto přednosti přináší. Základní myšlenkou je snaha, aby vývoj první verze či konceptu aplikace na něm postavené byl co možná nejjednodušší, čehož dosahujeme implicitním chováním všude, kde je to možné. Jednotlivé podproblémy může programátor řešit „na míru“ až ve chvíli, kdy je to potřeba. Na základě toho jsme se rozhodli framework pojmenovat **WheelDB Framework**. Název symbolizuje, že programátor ušetří čas a zbytečnou námahu, protože nemusí znovu „vynalézat kolo“. Obzvláště ve fázi vývoje prvotní verze, která má především levně ověřit, zda aplikace plní svůj účel, se ušetří vývoj rozsáhlého řešení, které se nakonec nemusí ukázat jako správné.

Framework se od knihovny liší tím, že zatímco knihovna je pouze seskupení tříd a metod volaných programátorem, framework přebírá řízení toku programu a programátor naopak dopisuje metody, které framework volá. Rozhodnutí zařadit WheelDB mezi frameworky podrobněji rozebereme v kapitole 2.3.

Vyšší náročnost na výpočetní kapacitu při použití robustního frameworku, pokud nezajde do extrému, již nepředstavuje vzhledem k rychlosti současných serverů zásadní problém. Naopak vzhledem k tomu, že se zadání aplikace během vývoje obvykle mění a nedostatky se často odhalí teprve po vytvoření základní funkční verze, vývoj zefektivní takový framework, který maximálně automatizuje vytvoření základní verze aplikace. Vývojář se tak může věnovat implementačním detailům tam, kde není výchozí chování frameworku pro aplikaci optimální, až poté, co zadavateli odprezentuje základní verzi a ten potvrdí, že struktura aplikace odpovídá jeho představám.

Při vývoji frameworku jsme použili některé návrhové vzory a osvědčené postupy z jiných frameworků, např. v databázových aplikacích užitečný návrhový vzor ORM (object-relational mapper) a do jisté míry i MVC (model-view-controller), nechceme se však těchto vzorů striktně držet — posloužily pouze jako inspirace pro návrh frameworku. Jednotlivé vrstvy WheelDB Frameworku jsou psány tak, aby mohly být při vývoji aplikace založené na frameworku snadno upraveny nebo zcela nahrazeny. Prvotní verze frameworku pokrývá konstrukce (datové typy, SQL dotazy apod.) nejčastěji používané ve webových databázových aplikacích. Na základě zkušeností z používání frameworku bude dále rozvíjena podpora i méně obvyklých typů tak, aby framework pokryl co nejvíce scénářů skutečných aplikací.

V této práci jsme se zaměřili na co nejsnazší a zároveň efektivní vývoj webových databázových aplikací v jazyce PHP. PHP je po Javě nejpoužívanějším jazykem pro tvorbu webových aplikací (Langpop, 2013). Oproti Javě a její technologii JavaServer Pages je díky jak jednoduchosti jazyka, tak nenáročnosti na provoz na serveru oblíbený u jednodušších aplikací.

Formátování

- Cesty k souborům `c:/esta/k/soub.oru` jsou v této práci uvedeny neproporcionálním `teletype` fontem šedou barvou.
- *Odborné termíny a pojmy* jsou psány kurzívou tmavě modrou barvou.
- technické názvy (názvy příkazů apod.) jsou psány bezpatkovým písmem tmavě modrou barvou.
- Krátké úseky zdrojového kódu `$foo->bar();` jsou psány neproporcionálním písmem barvou okolního textu (černou).
- Delší úseky zdrojového kódu jsou formátovány do speciálních bloků se zvýrazněním syntaxe.

1. Použité technologie

1.1 Platforma

Pro vývoj frameworku jsme zvolili programovací jazyk PHP z několika důvodů:

- poskytuje všechny prostředky, které pro vývoj frameworku potřebujeme: objektový model s podporou *třídních rozhraní*, knihovny pro mnoho různých SQL databází a jmenné prostory pro přehledné rozdělení tříd;
- je podporován velkým množstvím veřejných webových serverů ve srovnání s ostatními serverovými jazyky;
- jeho myšlenka je nejblíže cílům, kterých chceme ve WheelDB Frameworku dosáhnout, tj. vývoj v něm je snadný a intuitivní.

Minimální verze PHP vyžadovaná frameworkem je 5.3.0, teprve od této verze jsou podporovány jmenné prostory. Framework nicméně nepoužívá další prvky dostupné až od této verze (například anonymní funkce), aby se usnadnilo případné vytvoření verze funkční i na starších verzích PHP 5.

1.2 Databáze

Samotný framework je nezávislý na použité databázi, s databází pracuje pomocí ovladačů. V počáteční implementaci frameworku jsme vytvořili ovladač pro databázové systémy MySQL a Oracle, protože představují velký kontrast v přístupu k databázovým aplikacím. Databáze MySQL je velmi jednoduchá na používání, ale není vhodná pro profesionální aplikace pracující s velkým množstvím dat nebo vyžadující různou pokročilou funkcionalitu. Oracle složité databáze zvládá podstatně lépe, chybí mu ale některé jednoduché nástroje, kterými MySQL disponuje.

MySQL jsme vybrali z toho důvodu, že jde o databázi nejčastěji používanou v kombinaci s PHP pro jednodušší aplikace a je podporována téměř všemi veřejnými webovými servery, které nabízí hosting s PHP. Nabízí mnoho rozšíření dotazovacího jazyka oproti SQL standardům, které zjednodušují obvyklé úkony (např. klauzule LIMIT pro omezení rozsahu vrácených záznamů, klauzule ON DUPLICATE KEY UPDATE pro jednoduché vkládání či aktualizaci záznamu, pokud již existuje záznam se stejným klíčem). Některé z těchto konstrukcí mohou svým zvláštním chováním vést k chybám programátora, například když použije klauzuli ON DUPLICATE KEY UPDATE na tabulku s více unikátními klíči — pokud budeme mít tabulku uživatelů s unikátními klíči přihlašovacím jménem a e–mailem, programátor snadno omylem použije tuto klauzuli pro detekci vložení nového uživatele se stejným přihlašovacím jménem již vedeným v databázi, ovšem při vložení již existujícího e-mailu se přepíše špatný záznam. MySQL databáze existuje ve verzi Community Edition dostupné zdarma, což je pro jednoduché programy také důležité.

Databázi Oracle jsme vybrali proto, že v mnoha ohledech kontrastuje s MySQL databází. Ačkoli také nabízí zdarma dostupnou verzi Express, její použití je mnohem obvyklejší u profesionálních aplikací pracujících s velkým objemem dat.

Vyznačuje se vyšší rychlostí a spolehlivostí. Místo jednoduchých konstrukcí jako např. LIMIT v MySQL je v Oracle databázích nutné si tyto časté úkony ošetřit sám — alternativu ke klauzuli LIMIT z MySQL databáze lze implementovat pomocí očíslování řádků výsledku dotazu a vnoření tohoto dotazu do dalšího, který rozsah záznamů určí z čísel řádků, chování ON DUPLICATE KEY UPDATE lze implementovat pomocí složitější Oracle klauzule MERGE. Zápis je sice o něco komplikovanější, technicky ale mnohem průhlednější a nesvádí programátora k chybám nebo různým sporným situacím, jako jsme popsali u MySQL. Rozhraní WheelDB frameworku je z velké části inspirováno rozšířeními MySQL a některé funkce na ostatních databázích emuluje (řešeno ovladačem databáze). Pokud ovladač nedokáže určitou funkci emulovat, vyhodí výjimku WDB\Exception\UnsupportedByDriver.

Tabulka 1.1 obsahuje stručné srovnání vybraných aspektů, které uvádí Trujillo (2008).

	MySQL	**Oracle**
použití	Dostatečně výkonné levné řešení pro širokou škálu aplikací	Profesionální řešení pro aplikace s vysokými nároky na výkon, spolehlivost nebo funkcionalitu
nevýhody	Pokročilé funkce, především procedurální SQL, sice poskytuje, ale nemá zdaleka tak propracované	Zbytečně složité pro jednoduché aplikace, chybí některé konstrukce zjednodušující časté úlohy
možnost nastavení a škálování	malá	velká
příklady rozšíření jazyka oproti SQL-92	LIMIT INSERT IGNORE INSERT ... ON DUPLICATE KEY UPDATE REPLACE	MERGE — univerzální řešení pokrývající nejen problémy řešící v MySQL různé varianty příkazu INSERT, ovšem zápis je výrazně složitější
nejčastěji používané jazykem	PHP	Java
organizace tabulek	**schémata** (také se jim říká „databáze“, to je ale zavádějící pojem pletoucí se s databází jako takovou)	Tradičně se dělí **dle uživatele** vlastnícího tabulku, je možné použít tablespaces (jmenné prostory tabulek) — jde ale o úplně jiný princip dělení než schémata MySQL, jak by se mohlo na první pohled zdát.

	MySQL	Oracle
Příklady datových typů mimo SQL standard	ENUM — výčet (hodnotou je jeden prvek množiny), SET — množina (hodnotou je podmnožina množiny)	VARCHAR2 považuje prázdný řetězec a NULL za ekvivalentní, stejně jako v současnosti VARCHAR. Oracle plánuje v budoucnu pro klasický VARCHAR implementovat odlišení těchto hodnot, jak je vyžadováno standardem ANSI. VARCHAR2 zachová současné chování a tak zajistí dopřednou kompatibilitu současných skriptů.

Tabulka 1.1: Srovnání databáze MySQL a Oracle

1.3 Knihovny

- PHP
  - Mysqli — na ní je postaven ovladač MySQL pro WheelDB, `http://cz2.php.net/manual/en/mysqli.installation.php`
  - Oci8 — na ní je postaven ovladač Oracle pro WheelDB, `http://cz2.php.net/manual/en/mysqli.installation.php`
- JavaScript
  - jQuery — použita validátorem formulářů ve webovém rozhraní, `http://www.jquery.com`
  - Tigra Calendar — HTML kalendář pro výběr data použitý ve webovém rozhraní, `http://www.softcomplex.com/products/tigra_calendar/`
- CSS
  - Twitter Bootstrap — sada stylů použitá pro vytvoření výchozího vizuálního stylu uživatelského rozhraní, `http://twitter.github.io/bootstrap/`

2. Zásady a vývojové přístupy

2.1 Konvence před konfigurací

Anglicky se nazývá *Convention over configuration*. Tento přístup má za cíl snížit množství rozhodnutí, které musí programátor při vývoji aplikace učinit, nikoli ale na úkor flexibility knihovny či frameworku, která se tímto modelem řídí. Komponenta toho dosahuje tak, že všude, kde je to možné, nabízí rozumné výchozí chování, ale zároveň možnost toto chování změnit. Jednoduchým příkladem jsou implicitní parametry funkcí, tj takové, které nemusíme zadávat, pokud požadujeme obvyklou variantu chování funkce. Tímto způsobem ale mohou být psány celé konfigurační soubory nebo objektový model. (Convention over Configuration, 2013)

WheelDB framework nabízí rozhraní tříd a k nim výchozí třídy pro všechny vrstvy webové aplikace. Programátor může použít tyto výchozí třídy, napsat vlastního potomka některé z těchto tříd nebo vlastní třídu implementující odpovídající rozhraní — vlastní nezbytná práce programátora je tedy minimální, ale je mu zároveň ponechána maximální možnost rozhodnutí, do jaké míry výchozí chování frameworku využije. Výsledkem použití tohoto modelu je zaprvé kratší a jednodušší zdrojový kód a zadruhé mnoho ušetřené lidské práce.

2.2 Méně kódu, více bezpečnosti

Anglicky se nazývá *less code, more security*. Tento přístup je principielně podobný výše uvedenému *konvence před konfigurací*. Jeho cílem je nastavit výchozí chování frameworku takové, aby poskytovalo maximální úroveň bezpečnosti. Oba přístupy ve výsledku zjednoduší zdrojový kód použitím implicitních parametrů. Rozdíl je ten, že zatímco účel *konvence před konfigurací* je přímo zjednodušení zdrojového kódu a usnadnění práce programátora, *méně kódu, více bezpečnosti* určuje, že jako výchozí má být zvoleno takové chování funkce, které v případě chybného opomenutí parametru programátorem způsobí minimální (nejen bezpečnostní) problémy.

Jednoduchý příklad aplikace tohoto principu je volba *escapování*[1] řídících znaků. Jako špatný příklad tohoto přístupu uvedeme šablonový systém Smarty (`http://www.smarty.net`), který používá tuto syntaxi pro výpis proměnné na výstup: `hodnota proměnné a je {$a}`. Pro výstup s escapovanými speciálními znaky je nutné použít modifikátor `escape`, tj. `hodnota proměnné a je {$a|escape}`. Takové řešení nepovažujeme za dobré. Výpis escapovaného řetězce, který být escapovaný neměl, má obvykle za následek chybné chování programu — např. u HTML návštěvník stránky uvidí místo formátování řídící znaky, to ale není zdaleka takový bezpečnostní problém, jako chybně neoescapovaný vstup. Neoescapovaný uživatelský vstup často umožní útočníkovi vložit do programu vlastní výkonný kód, např. při zápisu do databáze nad ní může získat až neomezenou kontrolu. Jako mnohem vhodnější se jeví varianta, kdy vypsání proměnné na

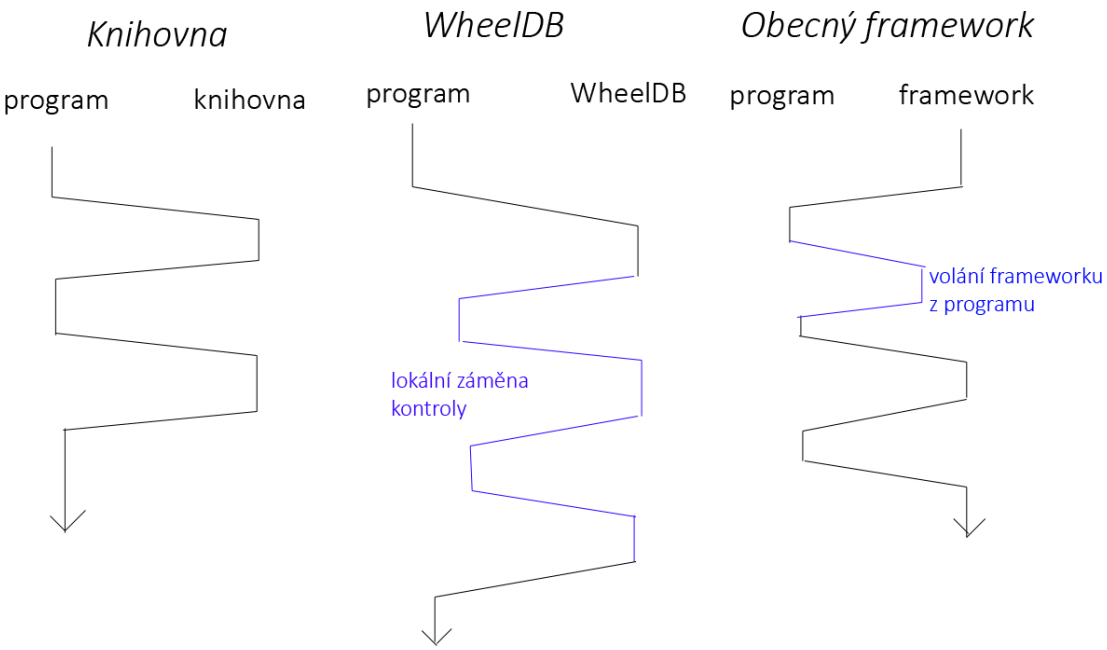
[1] *Escapování* je nahrazení řídích znaků speciální sekvencí, která je chápána jako původní tisknutelný a nikoli řídící znak.

výstup je implicitně escapované a explicitně jej lze vypnout. Tím je minimalizováno bezpečnostní riziko, když se programátor nad escapováním nezamyslí a chování nespecifikuje.

Ve WheelDB frameworku tento přístup aplikujeme především v modulu pro vytváření SQL dotazů. Všechny identifikátory a hodnoty jsou automaticky escapovány a ošetřeny na neplatný vstup. Pokud chce programátor vytvořit část dotazu ručně a tedy do něj vložit neescapovanou část, musí k tomu použít speciální třídu pro vepisování vlastního kódu. Nemůže se tedy stát, že by něco zapomněl oescapovat při použití základního dotazovacího rozhraní frameworku.

2.3 Framework, nebo knihovna?

Framework je podobně jako knihovna seskupení podprogramů či tříd řešících nějakou množinu spolu souvisejících softwarových problémů. Knihovna ovšem nenabízí nic víc. Oproti tomu framework zároveň hlouběji definuje filozofii, jak budou tyto třídy používány a jak spolu budou navzájem komunikovat. Hlavním rysem frameworku odlišujícím jej od knihovny je *záměna řízení aplikace* (anglicky se nazývá *inversion of control*)— zatímco při použití knihovny jsou tělo a řídící struktura definovány v programu a z nich jsou volány funkce knihoven, framework naopak definuje sám řídící strukturu a výchozí chování. Program postavený na frameworku definuje pouze podprogramy implementující řešení úloh, u kterých se výchozí řešení frameworku nehodí, a tyto podprogramy jsou frameworkem volány automaticky. Aplikace používající framework ale také může volat a často volá metody frameworku, jako by to byla knihovna.



Obrázek 2.1: Porovnání knihovny a frameworku dle toku programu

Srovnání demonstruje obrázek 2.1. Z obrázku je patrné, že zařazení WheelDB je problematické. WheelDB sice nedefinuje tělo celé webové aplikace, ale pro jistou její část se chová jako framework - definuje rozsáhlý kus kódu odvolávající

se na funkce vložené programem, tedy princip *inversion of control*. Ačkoli předpokládáme jeho integraci do složitější struktury nebo nadřazeného frameworku pro webové aplikace, může stejně dobře fungovat i samostatně. Pojem framework se v praxi používá celkem volně. Jeho definice zatím není ustálená, zmíněnou charakteristiku známe spíše z praxe za již celkem obecně uznávanou. Z těchto důvodů jsme se projekt WheelDB rozhodli označit za databázový framework, podle nás nejde o prostou knihovnu a k povaze frameworku má mnohem blíže. (Riehle, 2000)

3. Srovnání s konkurenčními produkty

3.1 NotORM

Tato knihovna od Jakuba Vrány posloužila jako inspirace pro jednoduchý přístup k databázovým datům včetně stromové struktury, kterou dokáže procházet podobně jako například nástroj simpleXML pro čtení XML souborů v PHP. Název pramení z toho, že knihovna nepoužívá návrhový vzor ORM mapování relací na objekty.

V době počátku vývoje WheelDB frameworku bylo velkou nevýhodou knihovny NotORM to, že vůbec neřešila zápis dat do databáze, ale pouze čtení.

V současné době NotORM používá podobný přístup, jaký jsme připravili i ve WheelDB frameworku — výsledky čtecích dotazů jsou mapovány na jednoduché vestavěné objekty, nad kterými je možné provádět CRUD (create, update, delete) operace. WheelDB navíc nabízí přechod o úroveň výše a vytvoření potomků těchto vestavěných tříd, čímž realizuje návrhový vzor ORM, nebo naopak o úroveň níže a pro zápis dat použít přímo rozhraní WheelDB frameworku pro tvorbu dotazů úplně mimo mapování relací.

Výhodou NotORM je odvážnější zápis názvů tabulek, sloupců a klíčů v PHP kódu. NotORM umožňuje psát tyto názvy přímo jako klíče pole, názvy metod a atributů:

```php
//iterace přes řádky tabulky
foreach ($db->application() as $id => $application) {
    //výběr sloupce
    echo $application['title']."-";
    //cizí klíč author
    echo $application->author['name']."\n";
}
```

Zdrojový kód 3.1: Výpis pomocí NotORM

Ve WheelDB jsme se rozhodli nejít tak daleko, protože jde o komplexnější framework a vestavěné třídy mají více vlastních metod a atributů, které by se mohly zbytečně plést s identifikátory databáze. Ve WheelDB kód pro stejný výpis názvů a autorů aplikací vypadá takto:

```php
// získání tabulky
$applications = WDB\Wrapper\TableFactory::fromName('applications');
//iterace přes řádky tabulky
foreach ($applications as $application) {
    //výběr sloupce
    echo $application['title']."-";
    //cizí klíč author (odkazuje se přes název
    //CONSTRAINT ... FOREIGN KEY podmínky, nikoli
    //přes název sloupce jako v NotORM)
    $author = $application['fk_applications_author'];
    //vypíšeme jméno autora ze záznamu získaného přes cizí klíč
    echo $author['name']."\n";
}
```

Zdrojový kód 3.2: Výpis pomocí WheelDB

Zápis je syntakticky o něco delší, ale z hlediska přístupu velmi podobný NotORM verzi. Délka ale nemusí být na škodu — takto je lépe viditelné, co je tabulka, co je sloupec ve které tabulce a co cizí klíč. Tím, že jsme se vyhli mapování databázových identifikátorů na PHP identifikátory, nedochází ke konfliktům mezi identifikátory databáze a zdrojového kódu PHP.

NotORM také vůbec neřeší webové rozhraní, tedy HTML část aplikace. V tomto ohledu tedy NotORM nelze srovnat s předpřipraveným webovým rozhraním, které nabízí WheelDB.

NotORM standardně předpokládá konvenční pojmenovávání sloupců a tabulek (primárních a cizích klíčů atd), ikdyž nabízí i možnost si názvy klíčů apod. z databáze sama načíst za cenu snížení výkonu. Oproti tomu WheelDB se primárně snaží maximum meta–informací načíst z databáze sám a ty si pak cachuje. To považujeme za v praxi užitečnější, protože konvencí vytváření databázových identifikátorů je nespočet a mnozí programátoři se bohužel nedrží vůbec žádné. WheelDB má zatím jen částečnou možnost zadat strukturu databáze ručně, dalšímu rozvoji se budeme věnovat až v budoucích verzích — například určení cizích klíčů tam, kde v databázi nejsou přímo definované, třeba proto, že formát nebo engine tabulky cizí klíče nemusí podporovat.

Asi největší výhodou NotORM proti WheelDB je optimalizace dotazů, které jsme se při vývoji současné verze WheelDB příliš nevěnovali na úkor mnohem pestřejší funkcionality a API.

3.2 Doctrine

Doctrine je velmi známá a rozšířená knihovna pro práci s databází v PHP. Představuje stabilní a osvědčenou knihovnu implementující tradiční návrhové vzory, které jsme se ve WheelDB snažili překonat inovativním přístupem k vývoji databázové aplikace.

Skládá se z vrstev odpovídajícím návrhovým vzorům DBAL (database abstraction layer), nad ním postavený ORM (object-relational mapper) a ODM (object document mapper). Všechny tyto vrstvy jsou komplexně propracované, oproti WheelDB frameworku ale chybí především generátor webového rozhraní a vestavěné CRUD objekty. Veškeré třídy, na které chceme mapovat tabulky z databáze, musíme knihovně Doctrine dodat sami. Doctrine také nepodporuje načtení datového modelu z databáze, což je základní stavební kámen WheelDB. Naopak WheelDB framework neobsahuje ODM vrstvu, která slouží ke kompletnímu oddělení aplikační vrstvy od databáze. Její princip odporuje určení WheelDB pro efektivní vývoj jednodušších aplikací, proto jsme její implementaci při návrhu WheelDB zavrhli.

3.3 Dibi

Dibi je zajímavý jednoduchý DBAL pro PHP. WheelDB je inspirován jeho jednoduchostí při práci s databází. Bohužel Dibi má několik základních nedostatků, kvůli kterým se příliš neuchytil.

Dibi poskytuje zajímavý přístup ke čtení relačních dat v podobě třídy DibiDataSource, který se dobře hodí k frameworkům na architektuře MVC (model,

view, controller) nebo MVP (model, viev, presenter). V datovém modelu aplikace je jakožto zdroj dat připraven dotaz na příslušnou tabulku nebo spojení tabulek. V aplikační vrstvě, controlleru nebo presenteru se vyfiltrují požadované záznamy (WHERE klauzule) a prezentační či zobrazovací část aplikace se postará o řazení a stránkování záznamů. DibiDataSource je přímo určen pro konstrukci dotazu do databáze tímto způsobem. Bohužel je takřka celý postaven na vnořených SELECT dotazech a ty jsou v MySQL databázi špatně optimalizované (s indexy na tabulkách nepracují vůbec nebo pracují špatně), proto je pro vývojáře používající MySQL takřka nepoužitelný.

Dibi také poskytuje jednoduchý a na první pohled šikovný nástroj pro jednoduché skládání dotazů s názvem DibiFluent. Například UPDATE dotaz se dá v DibiFluent zapsat takto:

```
$record = array(
        'title'  => 'Vyrobek',
        'price'  => 318,
        'active' => TRUE,
);
dibi::update('products', $record)
        ->where('product_id = %d', $id);
        ->execute();
```

Zdrojový kód 3.3: Update dotaz v DibiFluent

DibiFluent je ale ve skutečnosti jen *syntaktický cukr* (anglicky *syntactic sugar*) pro zápis dotazů. Proto DibiDataSource nedokáže pracovat nad jedním dotazem a řeší jednotlivé úkony vnořováním dotazů.

Ve WheelDB jsme místo toho připravili rozhraní reprezentující SQL dotaz pomocí hierarchie objektů, kde narozdíl od DibiFluent můžeme kdykoli pohodlně měnit kteroukoli část dotazu, a tedy řešit úkoly jako řazení, filtrování a stránkování v libovolném pořadí v rámci jednoho dotazu. Díky tomu je možné dotazovací rozhraní WheelDB přímo používat podobně jako DibiDataSource, čímž je eliminován problém s vnořenými dotazy.

Kvůli zmíněným problémům se Dibi nikdy příliš neuchytil a již se téměř nepoužívá. Nejčastěji se používal ve spojení s webovým Nette frameworkem (od stejného autora), který má již v současnosti vlastní jednoduchou databázovou knihovnu.

	WheelDB	NotORM	Doctrine	Dibi
Konstrukce dotazů	objektová reprezentace s možností snadných úprav existujícího dotazu	dotazy tvoří vnitřně, není možno tvořit vlastní	jazyk DQL podobný SQL optimalizovaný pro přístup k objektům perzistovaným v databázi	vlastní dialekt SQL pro sjednocení syntaxe různých databázových systémů, pomocné objektové rozhraní DibiFluent
Realizace ORM	implementace rozhraní vrstvy Wrapper pro vybrané tabulky	ne	ano, je na ni primárně zaměřen	ne
Optimalizace dotazů	ne	vybírá pouze použité sloupce, optimalizuje množinu vybíraných řádků při průchodu cizími klíči	ne	ne, naopak generuje vnořené dotazy, které jsou pro některé databáze problematicky optimalizaovatelné
Webové rozhraní	Generuje automaticky	ne	ne	ne
Načtení struktury databáze	automaticky	volitelně, jinak předpokládá konvenční pojmenování identifikátorů	strukturu databáze generuje z anotovaných PHP tříd, opačný přístup oproti WheelDB	ne
Validace vstupu	ano	ne	verze 1 skrz Doctrine_Record, ve verzi 2 vypuštěna	ne
Záznam dotazů	do speciální databáze se snadným vyhledáváním	ne	rozhraní Doctrine\DBAL\Logging\SQLLogger	ne

Tabulka 3.1: Srovnání s konkurenčními frameworky

4. Uživatelská dokumentace

Jelikož je tento produkt frameworkem používaným k programování koncových aplikací a uživatel je zároveň programátorem, nelze jednoznačně odlišit uživatelskou a programátorskou dokumentaci.

Rozhodli jsme se v uživatelské dokumentaci popsat použití frameworku při vývoji koncových aplikací a v programátorské dokumentaci detailně popsat architekturu frameworku a vysvětlit, jak framework rozšiřovat o další funkce.

4.1 Vytvoření jednoduché aplikace

Aplikací, kterou ve WheelDB můžeme nejsnáze vytvořit, je administrační rozhraní dat v relační databázi. Pokud nepočítáme konfigurační soubor s nastavením připojení databáze, stačí k tomu pouhé tři řádky zdrojového kódu:

```
1  <?php
2  //1. načtení tříd a autoloaderu frameworku
3  require '/path/to/wdb/loader.php';
4
5  //2. vytvoření objektu generátoru webového rozhraní
6  //   pro správu celého schématu tabulek
7  $schema = WDB\WebUI\Schema::forCurrentSchema();
8
9  //3. vygenerování a výpis HTML kódu
10 $schema->show();
11 ?>
```

Zdrojový kód 4.1: Příklad WheelDB aplikace

Tento kód postačí k vytvoření funkčního webového rozhraní. Nejprve tyto tři řádky kódu podrobněji vysvětlíme.

1. Načte soubor `loader.php`, ve kterém je třída WDBLoader. Tato třída se stará o *autoloading* (automatické načtení třídy do prostředí PHP ve chvíli, kdy je použita) tříd frameworku.

2. Vytvoří instanci třídy WDB\WebUI\Schema. Tato třída se stará o generování uživatelského rozhraní pro práci s celým schématem tabulek. *Tovární metoda*[1] forCurrentSchema() vytvoří instanci odkazující na výchozí schéma tabulek výchozího připojení k databázi. Voláním jiných továrních metod můžeme vytvořit instance odkazující na jiná schémata.

3. Vygeneruje a vypíše HTML kód přímo na výstup. Toto je nejjednodušší varianta vytvoření uživatelského rozhraní. Alternativou je metoda generateHTML(), která HTML kód vygeneruje a vrátí jako řetězec.

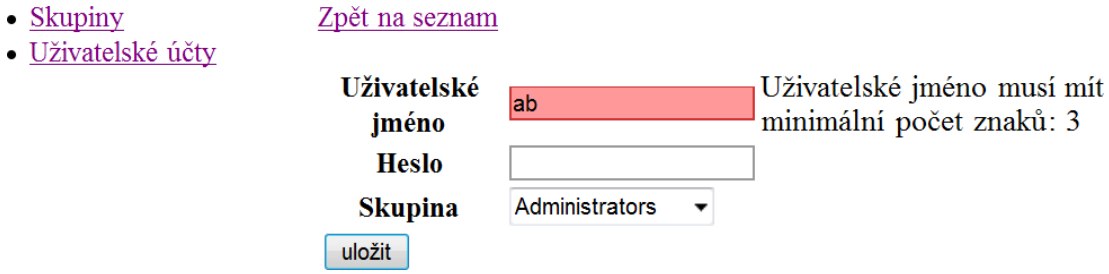
WheelDB ale sám negeneruje hlavičku HTML stránky. Předpokládá se začlenění do větší webové struktury, která bude obsahovat vlastní hlavičky. Navíc WheelDB pro plnou funkčnost vyžaduje načtení knihovny jQuery, vlastního javascriptového kódu a CSS stylů. Kompletní kód jednoduché aplikace včetně HTML bude tedy vypadat takto:

[1] anglicky *Factory method*, metoda dle návrhového vzoru *Object Factory*

```php
<?php
require '/path/to/wdb/loader.php';
$wa = WDB\WebUI\Schema::forCurrentSchema();
?>
<html>
<head>
    <link rel="stylesheet" type="text/css"
      href="/public/wdb.css" />

    <script type="text/javascript"
      src="http://code.jquery.com/jquery-1.7.1.min.js">
    </script>

    <script type="text/javascript"
      src="/public/wdb.js"></script>

    <meta http-equiv="content-type" content="text/html;charset=utf8" />
    <title>Moje první WheelDB aplikace</title>
</head>

<body>
  <?php $wa->show(); ?>
</body>
</html>
```

Zdrojový kód 4.2: Příklad WheelDB aplikace včetně HTML hlavičky

Výsledný program nad databází uživatelů a skupin bude vypadat zhruba takto:



Obrázek 4.1: Vygenerované uživatelské rozhraní

4.2 Konfigurace frameworku

4.2.1 Webový server

Framework umístíme do složky, která je přístupná (alespoň pro čtení) webovému serveru s nainstalovaným PHP. Pokud má webový server omezená práva k zápisu do systému souborů, povolíme možnost zápisu do složky `/var/` a všech podsložek.

4.2.2 PHP Konfigurační soubor

Soubor `config.php` v kořenovém adresáři frameworku obsahuje základní konfiguraci frameworku. Všechna nastavení jsou v něm řádně okomentována, vyjmenujeme jen ty nejpodstatnější.

```
'connections'=>array(
      'default'=>array( //identifikátor spojení. Pomoci něj
                        //je možné volat spojení z~aplikace
          'driver'=>'MySQL',     //platforma databáze
          'host'=>'localhost',   //ip nebo uri adresa serveru
          'user'=>'root',        //uživatelské jméno
          'password'=>'root',    //heslo
          'schema'=>'test',      //schéma tabulek

          //úroveň logování, viz WDB\Query\Log
          'loglevel'=>  WDB\Query\Log::LOG_ALL,
      ),
      'log'=>array( //další spojení s identifikátorem "log"
          ...
      ),
  ),
```

Zdrojový kód 4.3: Konfigurace připojení k databázi

Spojení lze přednastavit libovolný počet. Speciální význam mají spojení default a log.

Default je výchozí připojení, pokud žádné neuvedeme jako argument u rozličných metod, které s připojením pracují. Pokud default není nastaveno, je nutné u každé takové metody explicitně připojení zvolit.

Spojení log slouží pro zaznamenávání dotazů zaslaných do databáze a vrácených chyb, o které se stará sám framework. Bude podrobněji vysvětleno v kapitole 4.8. Pokud toto spojení není zadáno nebo nefunguje, záznam se neprovádí.

Dalším důležitým nastavením je položka lang. Podle ní se vybere výchozí jazyk frameworku (především generovaného webového rozhraní a různých hlášek generovaných ostatními vrstvami) z adresáře s jazykovými soubory `/lang/`.

Dále zde nalezneme nastavení různých adresářů frameworku, prefixů tříd a další nastavení.

4.2.3 Konfigurace JavaScriptu

Výchozí konfigurace javascriptu se nachází v souboru `/public/wdb.js` od řádku začínajícího `this.config =` . Parametry můžeme měnit přímo zde nebo odkudkoli ze stránky po natažení souboru `/public/wdb.js` pomocí `wdb.config.`*název_atributu*. V této verzi je zatím jen jediný konfigurační parametr ajaxUrl udávající relativní url adresu, ze které je přístupný soubor `/public/ajax.php`.

4.2.4 Zavedení frameworku do aplikace

Pro zavedení *autoloadingu* tříd frameworku vložíme soubor `loader.php` do kódu PHP aplikace. To postačí k fungování celého frameworku vyjma javascriptové části a CSS stylů generovaného webového rozhraní. K těm je ještě potřeba vložit v HTML hlavičce CSS soubor `/public/wdb.css`, knihovnu jQuery a javascriptový soubor `/public/wdb.js`. Framework je testován s jQuery verzí 1.7.1 z adresy `http://code.jquery.com/jquery-1.7.1.min.js`, kopie též přiložena u frameworku v souboru `/public/jquery-1.7.1.min.js`.

4.3 Dotazovací rozhraní (DBAL vrstva)

WheelDB framework nabízí plně objektové rozhraní pro tvorbu, spouštění a procházení výsledků SQL dotazů typu SELECT, INSERT, UPDATE, DELETE.

Není implementováno rozhraní pro práci s dotazy měnícími strukturu databáze — nepředpokládá se, že by je aplikace využívala, a jde o zcela odlišnou úlohu od práce s daty v databázi, na kterou je framework zaměřen. S dotazem pracujeme jako s objektovou strukturou, nikoli řetězcem. Řetězec dotazu je vygenerován z objektu dotazu až těsně před jeho spuštěním.

4.3.1 Proč objektová reprezentace

Cílem při vytváření dotazovacího rozhraní WheelDB bylo reprezentovat dotaz tak. aby kteroukoli jeho část bylo možné kdykoli pohodlně změnit. Sestavit strojově dotaz tímto způsobem je snazší a méně náchylné k chybám programátora než pokoušet se strojově něco doplnit doprostřed dotazu reprezentovaného řetězcem.

Představme si, že chceme napsat funkci, na jejímž vstupu bude libovolný dotaz typu SELECT a výstupem bude tentýž dotaz se změněnou množinou vypisovaných sloupců nebo přidanou podmínkou do klauzule WHERE.

Pokud je dotaz reprezentován jako řetězec, musíme v něm najít, na kterou pozici lze přidat sloupec do množiny vypisovaných sloupců, kde se nachází sloupec, který chceme z množiny odebrat, zda jsou všechny čárky okolo vpořádku abychom zachovali správnost syntaxe atd. U WHERE klauzule navíc budeme řešit, zda vůbec nějaká existuje, pokud ano — připojíme novou podmínku operátorem AND, pokud ne — vytvoříme ji.

Pokud je však dotaz reprezentován objektově, je jeho libovolná úprava triviální:

```
1  use WDB\Query\Element;
2  function uprav(WDB\Query\Select $select)
3  {
4      $select->addCondition(Element\Compare::Equals(
5          Element\ColumnIdentifier::create('foo'),
6          new Element\Datatype\String('bar'))
7      );
8      $select->removeField('baz');
9
10 }
```

Zdrojový kód 4.4: Úprava SELECT dotazu ve WheelDB

Zápis podmínky je sice zdlouhavý (tři řádky vytváření objektů namísto jednoduchého `foo='bar'`), ale ve srovnání s tím, kolik kódu bychom museli napsat, abychom stejnou operaci mohli provést s dotazem v řetězci, je stále podstatně kratší a přehlednější.

4.3.2 Princip a použití

Každý dotaz je reprezentován instancí potomka WDB\Query\Query odpovídající typu SQL dotazu, tedy WDB\Query\Select, WDB\Query\Update, WDB\Query\Insert, WDB\Query\Delete. Třída může reprezentovat i různé varianty dotazů — např. nastavení určitého módu INSERT dotazu může vyústit v to, že skutečný

dotaz položený do MySQL bude REPLACE, v Oracle MERGE atd. Dotaz vygenerovaný uvedenými třídami tedy vždy nemusí interně být shodný s názvem třídy, jak by se na první pohled mohlo zdát.

Spuštění dotazu proběhne ve chvíli zavolání metody WDB\Query\Query::run ([WDB\Database $database]) na objektu příslušného dotazu. Návratovou hodnotou volání je instance rozhraní WDB\Query\iQueryResult, v případě SELECT dotazu konkrétně WDB\Query\iSelectResult, u zapisujících dotazů WDB\Query\iWriteResult.

```php
// výběr sloupce "name" z tabulky "users"
$q = new WDB\Query\Select(array('name'), 'users');

//výsledek bude seřazen podle jména
//vybereme pouze uživatele ze skupiny administrátorů
$q
  ->sort('name')
  ->filter(array('grp'=>'adm'));

//spuštění na výchozím databázovém připojení
$users = $q->run();

//výpis uživatelů
foreach ($users as $user)
{
  echo $user['name']."\n";
}
```

Zdrojový kód 4.5: Příklad SELECT dotazu

Následuje příklad použití ostatních typů dotazů:

```php
$ins = new WDB\Query\Insert('users', array(
                'name'=>'Jan Novák',
                'grp'=>'adm'
              ));
$ins->run();

//nyní je vložen administrátor Jan Novák do tabulky uživatelů

$upd = new WDB\Query\Update('users',
           array('grp'=>'usr'),
           array('name'=>'Jan Novák'));
$upd->run();

//přeřadili jsme Jana Nováka do skupiny běžných uživatelů

$del = new WDB\Query\Delete('users',
           array('name'=>'Jan Novák'));
$del->run();
```

Zdrojový kód 4.6: Příklad INSERT UPDATE a DELETE dotazu

Je zde velký prostor pro rozšíření dotazovacího rozhraní především pro kratší zápis zdrojového kódu cílové aplikace, se kterým počítáme v dalších verzích WheelDB frameworku, například instanciace dotazu přímo z objektu databázového spojení nebo statické třídy s jednoduchým názvem pro tvorbu dotazů a iterace přímo přes objekt dotazu, jejíž započetí vyvolá spuštění dotazu. Zkrácený kód v budoucí verzi by mohl vypadat třeba takto:

```php
use WDB\Query\B;
foreach(B::select(array('name'), 'users')
    ->sort('name')->filter(array('grp'=>'adm')) as $user)
{
  echo $user['name']."\n";
}
```

Zdrojový kód 4.7: zjednodušený SELECT dotaz

4.4 Model 3 vrstev a mapování entit databáze

Hlavní část frameworku pracující s relační databází je složena ze tří vrstev. Každá vrstva obsahuje několik tříd definovaných *rozhraními tříd*.

Programátor cílové aplikace může pro každou tabulku použít tyto vestavěné třídy, vytvořit jejich potomky nebo v případě potřeby zásadně odlišného chování pouze implementovat odpovídající rozhraní.

4.4.1 Analyzer

Analyzer je nejnižší vrstva, která se stará o analýzu struktury databáze a poskytuje objektové schéma reprezentující tuto strukturu vyšším vrstvám. Strukturou databáze je myšlen například seznam tabulek a jejich sloupců, u sloupců datový typ, výchozí hodnota, cizí klíče a další údaje.

Zde předpokládáme, že cílová aplikace si vystačí s vestavěnými třídami nebo je bude rozšiřovat zcela minimálně, spíše v případě rozšiřování frameworku o podporu složitějších nestandardních datových typů apod.

4.4.2 Wrapper

Wrapper je vrstva blízká návrhovému vzoru ORM. Zde předpokládáme, že programátor cílové aplikace bude často používat vlastní potomky vestavěných tříd a rodičovská třída se postará o ORM mapování. Vestavěné třídy jsou pro vývoj základu aplikace často postačující a mohou být nahrazeny vlastní implementací až při dalším rozvoji.

4.4.3 WebUI

WebUI je automatický generátor webového rozhraní pro tabulky načtené do vrstvy Wrapper. Podobně jako u vrstvy Wrapper předpokládáme časté používání vlastních tříd, vestavěné třídy ale postačí k vytvoření jednoduchého prohlížeče dat či administračního rozhraní databáze.

Základní myšlenka generátoru je rozdělení práce s tabulkami do tří pohledů:

- **výpis všech záznamů** s možností řazení a stránkování, v budoucích verzích frameworku počítáme např. s filtrováním podle hodnot a dalšími rozšířeními. Ve výpisu záznamu se objevují pouze nejdůležitější položky, komplexnější data se zobrazí zkráceně. Například u tabulky článků to může být název článku, autor, datum a první věta. U každého záznamu je také odkaz pro zobrazení detailu, úpravu nebo smazání záznamu, pokud je k tomu uživatel oprávněn.

- **detail záznamu** obsahuje všechny uživateli dostupné informace o záznamu, tedy například celý text článku, další informace jako související články, klíčová slova atd.

- **editace záznamu** obsahuje všechny položky záznamu, které má uživatel oprávnění upravit, a vhodné rozhraní pro možnost úprav.

Vestavěné WebUI rozhraní se řídí tímto modelem pohledů, ale programátor cílové aplikace si může samozřejmě zvolit jiný způsob. Do budoucích verzí WheelDB frameworku je možným vylepšením rozhraní podobné tabulkovému procesoru, jako je například spreadsheet v aplikaci Google Documents, kde jsou všechny tři výše uvedené pohledy sloučeny do jedné interaktivní tabulky.

4.4.4 Entity vrstev

Vrstvy jsou rozdělené do 4 entit schematicky znázorněných na obrázku 4.2.

- *Schema* reprezentuje celé schéma tabulek. Ve vrstvě Analyzer slouží jako kontejner obsahující všechny tabulky, ve vrstvě WebUI poskytuje uživatelské rozhraní pro procházení tabulek. Ve vrstvě Wrapper nemá schéma žádný význam, a proto není implementováno.

- *Table* reflektuje tabulku v databázi. Obsahuje operace jako vytvoření nového záznamu, vygenerování HTML kódu webového rozhraní apod. Entita tabulky se nachází ve všech vrstvách.

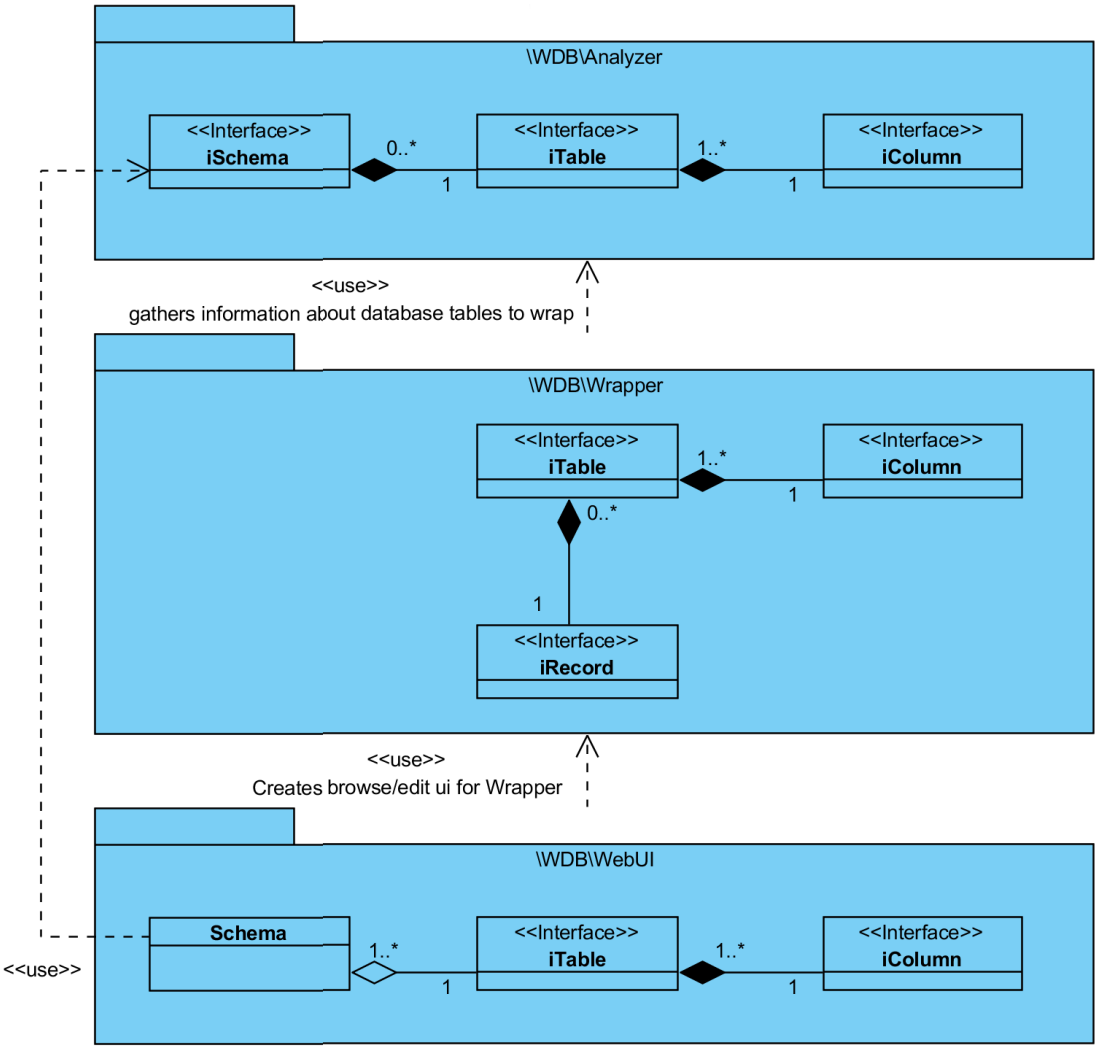
- *Column* odpovídá sloupci tabulky. Obsahuje informace o jeho názvu, datovém typu, omezeních (např. délky řetězce) jak daných databází, tak nastavených programátorem a další. Stejně jako tabulka se nachází ve všech vrstvách.

- *Record* reprezentuje řádek z dané tabulky. Obsahuje hodnoty uložené v tomto řádku a nabízí rozhraní pro práci s nimi na základě návrhových vzorů ORM (mapování relací na objekty) a ActiveRecord (zejména CRUD operace). Tato entita se vyskytuje pouze ve vrstvě Wrapper.

4.5 Šablony webového rozhraní

Webové rozhraní WheelDB frameworku je generováno pomocí šablon a adaptéru, který umožňuje použít různé šablonové systémy. Adaptér je libovolná třída implementující rozhraní WDB\WebUI\iTemplateAdapter. Pro danou tabulku lze nastavit třídu adaptéru pomocí vlastnosti WDB\WebUI\Table::$TemplateAdapterClass, volitelně sadu šablon pomocí vlastnosti Table::$templateSet.

Ve frameworku je vestavěný jednoduchý adaptér, který jako šablony používá HTML soubory s PHP kódem. Programátor cílové aplikace může využít existující šablony, napsat si vlastní sadu nebo vytvořit adaptér pro jiný šablonový systém, např. Smarty (`http://smarty.net`). Šablony se nachází v adresáři `/templates/`.



Obrázek 4.2: Schéma vrstev WheelDB frameworku

4.6 Validátory dat

WheelDB framework obsahuje vestavěné validátory dat zapisovaných do databáze pomocí vrstvy Wrapper. Validátory jsou úzce provázané s generátorem webového rozhraní, kde nabízí komfortní validaci pomocí JavaScriptu.

Velká výhoda tohoto přístupu je, že validační pravidla stačí definovat na jednom místě pro celou aplikaci a není nutné psát zvlášť javascriptový validátor, který je dnes téměř nutnou součástí uživatelsky přívětivé webové aplikace.

Omezení daná strukturou databáze (maximální délka řetězce, rozsah čísel atd.) si dokonce validátory nastaví samy načtením údajů z vrstvy Analyzer, takže o chyby vzniklé zadáním příliš dlouhého uživatelského vstupu se cílová aplikace nemusí starat, framework je automaticky zařadí mezi validační pravidla.

Hlavním a nejčastěji používaným vestavěným validátorem je třída WDB\Validation\ColumnValidator pro validaci obvyklých omezení na jednotlivých sloupcích. Pravidla si načte instance WDB\Wrapper\Column::$rules třídy WDB\Validation\Rules. Pomocí instance třídy `Column` je možné pro daný sloupec dané tabulky pravidla dle potřeby zkonfigurovat. V souboru třídy `Rules` jsou popsána všechna vestavěná pravidla.

Dalším vestavěným validátorem je WDB\Validation\UniqueConstraint. Ten je také konfigurován automaticky a stará se o kontrolu duplicity u všech unikátních klíčů dané tabulky včetně klíčů přes více sloupců.

Je možné si napsat vlastní validátory implementující rozhraní WDB\Validation\iValidator a případně je automaticky načítat pro všechny záznamy dané tabulky tak, že pro tabulku vytvoříme vlastní třídu na úrovni vrstvy Wrapper a validátor zkonfigurujeme v metodě fetchValidators.

4.7 Anotace

Chování frameworku pro tabulky a sloupce v databázi je možné konfigurovat pomocí anotací. **U názvů anotací nezáleží na velikosti písmen**. Mohou se nacházet na těchto místech:

- přímo ve struktuře databáze v komentáři dané tabulky nebo sloupce.
- v *PhpDoc*[2] komentáři třídy (u většiny anotací wrapper třída tabulky nebo sloupce), která je použita pro danou tabulku/sloupec. Tuto možnost lze použít, pouze pokud cílová aplikace definuje vlastní třídu pro danou tabulku/sloupec, a tedy nepoužívá pouze vestavěnou třídu frameworku.
- v souboru konfigurace modelu ve formátu NeonWheel — viz kapitola 4.7.4

4.7.1 Struktury pomocí anotací

WheelDB používá shodné anotace @property s dokumentačním nástrojem PhpDoc a na jejich základě umožňuje vytvářet inteligentní struktury, které si hlídají, jaké atributy mají definované a zda je k nim povolen přístup pro čtení nebo zápis. Tyto anotace zadáváme do PhpDoc komentáře potomka třídy WDB\Structure\Structure.

Anotace	Použití
@property typ název	atribut objektu
@property-read typ název	atribut pouze pro čtení
@property-write typ název	atribut pouze pro zápis

Tabulka 4.1: Anotace pro konfiguraci struktur

- **@property** lze použít v phpDoc komentáři potomků třídy WDB\Structure\Structure. Definuje atribut pro čtení i zápis.
- **@property-read** se používá stejně jako @property, ale takto označené vlastnosti jsou pouze pro čtení. Definovat jejich hodnotu lze pouze skrz parametr konstruktoru nebo uvnitř struktury.
- **@property-write** je obdobně vlastnost pouze pro zápis. Číst ji je možné pouze uvnitř struktury.

[2] Dokumentační komentář PHP třídy, začíná sekvencí `/**` a končí `*/`

Příklad 4.8 ukazuje jednoduché vytvoření struktury pomocí anotací. Podrobněji tento mechanismus vysvětlíme v kapitole 5.2.2.

```php
/**
 * @property-read int $id
 * @property string $jmeno
 * @property int $vek
 */
class Osoba extends WDB\Structure\Structure {}

$osoba = new Osoba(array('id'=>1, 'jmeno'=>'Jan Novák'));
$osoba->vek = 18;
echo "{$osoba->id}:{$osoba->jmeno},{$osoba->vek}"; //1:Jan Novák,18
$osoba->id = 0; //atribut pouze pro čtení, vyhodí výjimku
```

Zdrojový kód 4.8: Jednoduchá struktura vytvořená pomocí anotací

4.7.2 Konfigurace datového modelu

Všechny následující anotace je možné použít v komentáři sloupce nebo tabulky v databázi, v konfiguračním souboru NeonWheel nebo v PhpDoc komentáři Wrapper třídy sloupce nebo tabulky (podle typu anotace).

Anotace	Použití
@wrapper jménoTřídy	Wrapper třída sloupce nebo tabulky
@webUI jménoTřídy	WebUI třída sloupce nebo tabulky
@recordWrapper jménoTřídy	Wrapper třída záznamu tabulky
@noDisplay	zakáže zobrazování v seznamu tabulek
@noAccess	zakáže přístup k tabulce z WebUI vrstvy
@title název	název tabulky nebo sloupce
@display módy	mód zobrazení sloupce
@displayDefault módy	výchozí mód sloupců tabulky
@auto	hodnota sloupce doplňovaná automaticky
@nameColumn	sloupec tabulky reprezentující jméno záznamu

Tabulka 4.2: Anotace pro konfiguraci datového modelu

- **@wrapper** lze použít v komentáři databázové tabulky se jménem třídy implementující WDB\Wrapper\iDBTable, stejně tak v komentáři sloupce tabulky se jménem třídy implementující WDB\Wrapper\iDBColumn. Třída uvedená jako parametr bude ve wrapper vrstvě použita k práci s touto tabulkou či tímto sloupcem.
- **@webUI** lze použít v komentáři databázové tabulky se jménem třídy implementující WDB\WebUI\iTable, stejně tak v komentáři sloupce tabulky se jménem třídy implementující WDB\WebUI\iColumn. Třída uvedená jako parametr bude ve webUI vrstvě použita k práci s touto tabulkou či tímto sloupcem.

- **@recordWrapper** lze použít v komentáři databázové tabulky se jménem třídy implementující WDB\Wrapper\iRecord. Třída uvedená jako parametr bude ve wrapper vrstvě použita k práci s řádky této tabulky.

- **@noDisplay** lze použít v komentáři databázové tabulky. Tato anotace odstraní tabulku ze seznamu tabulek, které generuje komponenta vrstvy webUI pro generování webového rozhraní pro práci s celým schématem tabulek.

- **@noAccess** lze použít v komentáři databázové tabulky. Tato anotace zakáže vrstvě webUI přístup k této tabulce pomocí vygenerovaného webového rozhraní.

- **@title** lze použít v komentáři databázové tabulky nebo sloupce pro zadání člověkem čitelného názvu. Například tabulka *contact_info* by se mohla jmenovat „Kontaktní informace“.

- **@display** lze použít v komentáři sloupce databázové tabulky pro nastavení, ve kterých pohledech se má sloupec zobrazit. Syntaxe anotace je **display**`( [+|-]{list|detail|edit|editread|editwrite})*`. Je možné uvést několik direktiv najednou. Plus značí explicitní aktivaci tohoto módu, minus deaktivaci. Výchozí nastavení je u základních implementací rozhraní třídy iColumn plný přístup, tedy `@display +list +detail +edit`. Například **display** `+detail -editwrite -list` nastaví zobrazení sloupce pouze na detailu záznamu a při editaci pouze pro čtení, sloupec nebude zobrazen ve výpisu záznamů. Následuje podrobnější vysvětlení jednotlivých možností:

Direktiva	Význam
list	zobrazit ve výpisu záznamů
detail	zobrazit v detailu záznamu
edit	zobrazit ve formuláři pro editaci záznamu; ekvivalentní sjednocení direktiv editread|editwrite
editread	zobrazit ve formuláři pro editaci záznamu pouze pro čtení
editwrite	povolit ve formuláři pro editaci záznamu zápis[3]

Tabulka 4.3: Direktivy pro anotaci @display

- **@displayDefault** lze použít v komentáři tabulky stejně jako @display, nastaví se tím výchozí mód pro všechny sloupce dané tabulky.

- **@auto** označuje sloupec, jehož hodnota je doplňována automaticky, například AUTO_INCREMENT sloupce MySQL databáze nebo kombinací sekvence a triggeru v Oracle. Konfiguraci AUTO_INCREMENT lze detekovat automaticky, takže anotace @auto pro ni není potřebná. Automatická de-

[3] Direktivou `display -editwrite` je možné zakázat zápis do sloupce, jehož výchozí nastavení je `+edit`. Varianta `+editwrite` nedává smysl, sloupec, který není v editačním formuláři zobrazen, nebude ani zapsán.

tekce přiřazení triggerů a různých dalších možností automaticky vyplňovaného sloupce by ale byla obtížná. Takové sloupce je potřeba označit touto anotací, aby k nim framework správně přistupoval a nesnažil se ukládat do nich hodnoty.

- **@nameColumn** v komentáři tabulky označuje sloupec (nejčastěji textový), jehož hodnota reprezentuje název záznamu. U tabulky osob to může být jméno a příjmení, u tabulky filmů název filmu. Tento název záznamu se používá v uživatelském rozhraní zejména v případě, že na tuto tabulku odkazuje cizí klíč z jiné tabulky. Pokud v konfiguračním souboru definujeme položku recordNameColumn, poslouží jako výchozí identifikátor sloupce nesoucího název záznamu. Takový sloupec se často jmenuje např. *name*, *nazev*, *jmeno*, proto pokud se budeme držet nějaké konvence, vyhneme se nutnosti používat anotaci @nameColumn pro mnoho tabulek.

4.7.3 Konfigurace validátoru

Všechny následující anotace je možné použít stejně jako anotace pro konfiguraci datového modelu, tedy v komentáři sloupce v databázi, v konfiguračním souboru NeonWheel nebo v PhpDoc komentáři Wrapper třídy sloupce. Určují omezující podmínky pro tyto sloupce, které musí uživatelský vstup splnit.

Anotace	Použití
@minlength number	minimální počet znaků
@maxlength number	maximální počet znaků
@minvalue number	minimální číselná hodnota
@maxvalue number	maximální číselná hodnota
@required	povinné pole
@integer	musí být celé číslo
@float	musí být reálné číslo
@equals jménoSloupce	musí být rovno jinému sloupci
@pattern regex	musí odpovídat regulárnímu výrazu PCRE[4]
@email	musí být platný e-mail
@url	musí být platná URL adresa

Tabulka 4.4: Anotace pro konfiguraci validátoru

4.7.4 Konfigurační soubory NeonWheel

WheelDB nabízí možnost pro aplikace, pro něž by konfigurace datového modelu pomocí komentářů v databázi nebo u Wrapper tříd byla nepřehledná, konfigurovat datový model pomocí speciálního konfiguračního souboru.

Snažili jsme se nalézt formát, který by byl co nejjednodušší a pokud možno kompatibilní se syntaxí PhpDoc, který WheelDB používá. Nejvhodnějším nalezeným formátem byl formát Neon[5] pocházející z Nette frameworku. Neon je velmi

[4] viz `http://php.net/manual/en/book.pcre.php`
[5] viz `http://ne-on.org`

podobný známému formátu YAML, oproti němuž je ještě o něco jednodušší.

Jediný problém, který formát Neon pro toto použití má, jsou dvojtečky oddělující klíč a hodnotu. V anotacích PhpDoc slouží k oddělení klíče a hodnoty první mezera. Proto jsme formát Neon mírně upravili a ještě zjednodušili tak, aby dvojtečky nevyžadoval, čímž se stal téměř kompatibilním s formátem anotací PhpDoc. Nový formát jsme pojmenovali NeonWheel, soubory v tomto formátu označujeme příponou `.nw`.

Anotace zapsané do souboru ve formátu NeonWheel mohou, ale nemusí být uvozeny zavináčem — toto jsme zavedli pro zachování kompatibility s formátem PhpDoc anotací. Jediný případ, ve kterém formát NeonWheel není kompatibilní s vnitřním formátem anotací, je více anotací na jednom řádku. Ty NeonWheel nedokáže rozpoznat, myslíme si ale, že při vyčlenění konfigurace modelu do jednoho souboru je vhodné tento soubor formátovat přehledně a nešetřit zbytečně řádky na úkor přehlednosti, proto tuto nekompatibilitu nepovažujeme za problém.

Příklad souboru NeonWheel

Tato ukázka je část NeonWheel konfigurace ukázkové aplikace „půjčovna“, která bude představena v kapitole 6.3.

```
# tabulky
zakaznici
	title Zákazníci
	nameColumn jmeno
	jmeno
		title Jméno
		required
		minlength 3
	telefon
		title Telefon
		pattern "~^[0-9 +]*$~"

vypujcky
	title Výpůjčky
	wrapper VypujckyTW
	id_zakaznik
		title Zákazník
	vracena
		title Vrácena?
		webui WDB\WebUI\ColumnBoolean

# pohledy
volne_tituly
	title Volné tituly
	id_titul
		auto
	nazev
		title Název

_vypujcky_titulu
	noDisplay
```

Zdrojový kód 4.9: Ukázka syntaxe NeonWheel

Použití NeonWheel modelu pro schéma

V konfiguračním souboru frameworku v nastavení přípojení, které bylo popsáno v kapitole 4.2.2, můžeme ke schématu přidat klíč model a jako hodnotu zadáme název souboru, ve kterém se nachází NeonWheel konfigurace modelu. Není nutné uvádět příponu `.nw` a pokud se soubor nachází v podsložce `models` složky frameworku, není nutné uvádět ani cestu, stačí tedy vyplnit pouze název souboru bez přípony.

V případě, že v konfiguraci připojení neuvedeme klíč model, WheelDB použije soubor `models/`*`název_schématu`*`.nw`, pokud existuje. Chceme-li tomuto chování zamezit, nastavíme hodnotu klíče model na FALSE. S tímto nastavením nebude použit žádný, tedy ani automaticky načítaný model.

4.8 Záznam dotazů (log)

WheelDB nabízí inovativní způsob zaznamenání provedených dotazů a chyb do zvláštní databáze. Tento způsob jsme nenalezli u žádného jiného frameworku. Je výhodný především v tom, že provedené dotazy a chyby na nich lze procházet pomocí SQL dotazů. Tak je můžeme jednoduše filtrovat, řadit, seskupit dle potřeby a snadněji zjistit hledanou informaci nebo chybu.

Zřejmou nevýhodou záznamu do databáze je to, že nezaznamená fatální chyby znemožňující spojení s databází apod., kdy se pochopitelně nezdaří ani zápis chyby. Taková situace ale nastane pouze při špatné konfiguraci databázového spojení, čehož si programátor většinou všimne velmi brzy a snadno, nebo při fyzických výpadcích spojení s databází, na jejichž vině není chyba v programu. Přínos při ladění chyb s obtížně nalezitelnou příčinou je ale tak velký, že je tato drobná nevýhoda akceptovatelná a také jednoduše řešitelná, aplikace nebo budoucí verze frameworku mohou řešit tyto fatální chyby jednoduchým klasickým záznamem do textového souboru.

4.8.1 Nastavení záznamu

Pro úspěšné fungování záznamu je nutné nastavit databázové připojení s identifikátorem log, viz ukázka kódu 4.3. Databáze odkazovaná tímto připojením musí obsahovat tabulky pro záznam WheelDB. Skripty pro jejich vytvoření jsou umístěny v souboru `!setup/database.*.sql`, za hvězdičku jsou doplněny jednotlivé databázové platformy (MySQL, Oracle).

Pro MySQL je vytvořeno schéma tabulek s názvem *wdb*. Jelikož Oracle schémata tabulek nenabízí, jsou pro něj tabulky prefixovány *wdb_*.

Úroveň záznamu pro každé připojení lze nastavit pomocí vlastnosti loglevel daného připojení (opět viz ukázka kódu 4.3). Může nabývat následujících hodnot (konstanty třídy WDB\Query\Log):

Hodnota	**Význam**
LOG_ALL	Zaznamenává vše. Vhodné **pouze pro ladění**, může být velmi pomalé vzhledem k počtu zaznamenaných dotazů.

Hodnota	Význam
LOG_WRITE	Zaznamenává pouze dotazy zapisující data (insert, delete, update) nebo ty, které skončily chybou. Vhodné pro ladění nebo sledování změn v databázi.
LOG_ERRONEOUS	Zaznamenává pouze dotazy, které skončily chybou. Vhodné **pro produkční nasazení**.
LOG_NOTHING	Vypne záznam pro danou databázi.

Tabulka 4.5: Konstanty pro nastavení úrovně záznamu

Vyšší úrovně záznamu poskytují více informací o aktivitě databáze, ale mohou mít znatelný vliv na rychlost aplikace a v případě produkčního nasazení generovat nežádoucí velký objem dat, proto je třeba zvolit vhodnou úroveň záznamu.

Další zajímavou funkcí záznamu, kterou WheelDB nabízí, je **přiřazení čísla uživatele ke každé položce záznamu**. V kombinaci s úrovní alespoň LOG_WRITE získáme záznam o tom, který uživatel provedl v databázi jaké změny. To může být velmi cenné a v praxi se nám to několikrát osvědčilo například při hledání viníka, který provedl ve firemní databázi nežádoucí změny.

Samozřejmě dobrá aplikace je napsána tak, aby většinu nežádoucích změn ani neumožnila, nikdy tomu ale nelze zcela zabránit. V ideálním případě má aplikace, která potřebuje nástroj sledování změn, svůj vlastní na míru napsaný, to ale vyžaduje nemalé úsilí. WheelDB v souladu s jeho určením pro rychlý vývoj jednodušších aplikací poskytne dobře použitelnou verzi takového nástroje s minimálním úsilím programátora koncové aplikace.

Číslo uživatele nastavíme zavoláním `getLog()->setIdUser($value)` na požadovaném objektu databáze, na výchozím připojení tedy WDB\Database::getDefault()->getLog()->setIdUser($value). Volání této metody musí předcházet dotazy, u nichž chceme sledovat, který uživatel je provedl, tedy co nejdříve poté, kdy zjistíme číslo aktuálně přihlášeného uživatele.

4.8.2 Struktura záznamové databáze

WheelDB nenabízí žádné uživatelské rozhraní pro procházení záznamů. Záznamy mohou být procházeny nástroji pro správu databáze např. phpMyAdmin, Adminer nebo Oracle SQL Developer. Pro pochopení formátu záznamů uvádíme vysvětlení jednotlivých sloupců tabulek záznamu:

Sloupec	Význam
id_query	primární číselný klíč určený pro identifikaci položky záznamu
session	id PHP session navrácené funkcí `session_id()`[6]
request	identifikátor generovaný WheelDB společný pro právě jeden běh PHP skriptu (zjednodušeně jeden HTTP požadavek). Umožňuje rozlišit, které dotazy byly spuštěny v rámci jednoho běhu skriptu.
id_user	číslo uživatele, viz vysvětlení v kapitole 4.8.1

[6] viz `http://php.net/session-id`

Sloupec	Význam
ts	časové razítko provedení dotazu
query	textová reprezentace SQL dotazu
type	typ dotazu - jedna z hodnot 'select', 'insert', 'update', 'delete', 'other'
success	1 pokud byl dotaz úspěšný, 0 pokud došlo k chybě

Tabulka 4.6: Struktura tabulky query_log

Sloupec	Význam
id_query	Cizí klíč do tabulky query_log ukazující, který dotaz chybu vyvolal
class	Název třídy výjimky (potomek třídy WDB\Exception\QueryError), která chybu vyvolala
message	Chybová zpráva generovaná databází
schema	Schéma tabulek ke kterému se chyba vztahuje[7]
table	Tabulka, ke které se chyba vztahuje[7]
column	Sloupec, ke kterému se chyba vztahuje[7]
constraint	Omezující podmínka, ke které se chyba vztahuje[7]
value	Hodnota nebo výraz, který způsobil chybu[7]

Tabulka 4.7: Struktura tabulky query_errors

[7] vyplněno pouze pokud je pro daný typ chyby relevantní a lze určit

5. Programátorská dokumentace

5.1 Hierarchie tříd

Třídy a rozhraní tříd (pro účel této podkapitoly dále jen třídy) projektu jsou rozděleny do hierarchie jmenných prostorů, prostor nejvyšší úrovně je označen WDB. Třídy jsou načítány pomocí PHP autoloaderu, který je zaveden skriptem `loader.php`. Každá třída je autoloaderem hledána nejprve v adresáři odpovídajícím hierarchii jmenného prostoru a souboru *`název_třídy`*`.php`, dále postupně jako celý jmenný prostor v hierarchii nadřazených jmenných prostorů v souboru *`jmenný_prostor`*`.ns.php`. Například třída WDB\Query\Select bude hledána v těchto souborech v tomto pořadí:

```
WDB/Query/Select.php
WDB/Query.ns.php
WDB.ns.php
```

Možnost sloučit jmenný prostor do jednoho souboru je zde například kvůli jmennému prostoru WDB\Exception, ve kterém se nachází spousta tříd často zcela bez těla a pro každou tvořit samostatný soubor by bylo zbytečné. Navíc PHP, obzvláště spuštěné v prostředí Apache na systému Windows, velmi výrazně zpomalí, pokud musí načítat mnoho malých souborů ve srovnání s jedním velkým souborem.

Následuje přehled jmenných podprostorů prostoru WDB:

Jmenný prostor	**Popis**
\	základní třídy frameworku, důležitá třída Database zprostředkující práci s databází přes framework
\Ajax	třídy obsluhující AJAX požadavky webového rozhraní implementující rozhraní iHandler
\Analyzer	třídy reprezentující logickou strukturu databáze — tabulky, sloupce a jejich konfigurace, viz kapitola 4.4.1
\Annotation	třídy pro čtení konfigurace z anotací (viz kapitola 4.7)
\Exception	výjimky frameworku; popsány v API referenci
\Event	třída Event reprezentující událost, jíž je možné přiřadit naslouchající objekty, které jsou zavolány, pokud k události dojde
\Query	třídy pro tvorbu a generování databázových dotazů, viz kapitola 4.3
\Query\Element	třídy pro objektovou reprezentaci prvků databázového dotazu
\Query\Element\Datatype	třídy reprezentující jednotlivé datové typy pro použití v dotazu

Jmenný prostor	Popis
\SQLDriver	databázové ovladače
\SQLDriver\Result	část databázových ovladačů reprezentující výsledek dotazu (WDB\Query\iSelectResult)
\Structure	struktury, viz kapitola 5.2.2
\Utils	třídy obsahující utility užitečné v různých částech kódu
\Validation	třídy obsahující logiku validace uživatelského vstupu, viz kapitola 4.6
\WebUI	generátor webového rozhraní, viz kapitola 4.4.3
\WebUI\Structure	Struktury používané vrstvou WebUI
\Wrapper	ORM vrstva, viz kapitola 4.4.2
\Wrapper\Structure	struktury používané vrstvou Wrapper

Tabulka 5.1: Jmenné prostory

5.2 Základní objekty a utility

Nyní si popíšeme několik základních stavebních kamenů, které byly použity při vývoji WheelDB frameworku a které lze také využít jak při psaní vlastních aplikací, tak při rozšiřování frameworku.

5.2.1 Základní objekt

WDB\BaseObject je abstraktní třída sloužící jako základ většiny ostatních tříd frameworku. V současné verzi podporuje *magické gettery a settery*, v budoucích verzích může přibýt další funkcionalita užitečná pro všechny objekty.

PHP umožňuje definovat chování objektu při pokusu o přístup k atributu objektu, který není definován, pomocí metod `__get($name)` a `__set($name, $value)`. Při pokusu o čtení atributu provede PHP následující[1]:

1. Pokud je požadovaný atribut definován, vrátí jeho hodnotu.
2. Pokud je definována metoda `__get`, zavolá ji, jako parametr předá název požadovaného atributu a vrátí výsledek této metody.
3. Pokud není splněn ani jeden z předchozích bodů, vyvolá PHP chybu typu NOTICE a vrátí NULL.

Zápis do atributu PHP provádí analogicky takto:

1. Pokud je požadovaný atribut definován, přiřadí mu zapisovanou hodnotu.
2. Pokud je definována metoda `__set`, zavolá ji, jako první parametr předá název atributu a druhý parametr zapisovanou hodnotou.

[1] popsané chování se nevztahuje na statické atributy

3. Pokud není splněn ani jeden z předchozích bodů, atribut je dynamicky vytvořen a hodnota přiřazena dle bodu 1.

Třída WDB\BaseObject pomocí metod `__get` a `__set` implementuje funkcionalitu inspirovanou Nette frameworkem. Přístup k nedefinovaným atributům `$object->property` deleguje na metody `$object->getProperty()` a `$object->setProperty($value)`, pokud jsou tyto metody definovány. Pro snazší pochopení uvádíme příklad:

```
class Pes extends WDB\BaseObject
{
  private $delkaOcasu;
  private $hracky = array('kost','jina kost');
  public function setDelkaOcasu($value)
  {
    if ($value > 100) throw new TailTooLongException();
    else $this->delkaOcasu = $value;
  }
  public function getDelkaOcasu()
  {
    return $this->delkaOcasu;
  }
  public function getHracky()
  {
    return $this->hracky;
  }
}
$p = new Pes();
echo $p->hracky[0]; //vypíše "kost"
$p->delkaOcasu = 101; //vyvolá TailTooLongException

//pokud bychom neměli BaseObject, museli bychom použít tento zápis:
$h = $p->getHracky();
echo $h[0];
$p->setDelkaOcasu(101);
```

Zdrojový kód 5.1: Použití getteru a setteru

Gettery a settery slouží to jako *syntaktický cukr* pro pohodlnější zápis kódu. V případě PHP 5.3 je ještě jedna velká výhoda — pokud nějaký getter vrací pole, je možné k jeho prvkům přímo přistoupit pomocí `$p->hracky[0]`. Zápis `$p->getHracky()[0]` bohužel funguje až od PHP verze 5.4.

5.2.2 Struktury

Abstraktní třída WDB\Structure\Structure je základem pro návrhový vzor Data transfer object, jinak řečeno něco podobného jako konstrukce struct jazyka C.

V PHP lze použít jako datové úložiště pole nebo prázdnou třídu (stdClass), ale není možné mít kontrolu nad tím, jaké atributy bude taková struktura mít. V PHP lze používat atributy, které nebyly ve třídě definovány, ale není to považováno za dobrý způsob programování.

Třída Structure přidává kontrolu nad tím, jaké členské proměnné budou objekty mít a zda k nim je přístup pro čtení a zápis. Členské proměnné jsou deklarovány pomocí *PhpDoc* anotací, což bylo popsáno v kapitole 4.7.1. Na příkladu ukážeme, jak využít vlastností této třídy:

```php
/**
 * @property-read string $author
 * @property string $text
 * @property-write string $secretText
 */
class Message extends WDB\Structure\Structure
{
  public function readSecretText($password)
  {
    if ($password='secret')
    {
      return $this->data['secretText'];
    }
  }

  public function changeAuthor($password, $newAuthor)
  {
    if ($password='secret')
    {
      //readonly vlastnosti se nastavují protected metodou
      //_setReadonly nebo v konstruktoru
      return $this->_setReadonly('author', $newAuthor);
    }

  }

  public function __construct($author, $text)
  {
    //původní konstruktor přijímá jediný parametr
    //asociativní pole ve tvaru název=>hodnota
    parent::__construct(
      array('author'=>$author, 'text'=>$text)
    );
  }
}
$m = new Message('John Doe', 'Hello world');
$m->secretText = 'goodbye world';

echo $m->author;
echo $m->text;
//jediný způsob, jak získat 'goodbye world' zpět
echo $m->readSecretText('secret');


//všechny následující příkazy způsobí výjimku:

//proměnná pouze pro čtení
//vyhodí výjimku WDB\Exception\ReadOnlyProperty
$m->author = 'Intruder';
//proměnná pouze pro zápis
//vyhodí výjimku WDB\Exception\WriteOnlyProperty
echo $m->secretText;
//neexistující proměnná vyhodí výjimku
//WDB\Exception\NotExistingProperty
$m->foo = 'bar';
```

Zdrojový kód 5.2: Ukázka použití Structure

5.2.3 Události

WheelDB nabízí jednoduchý model událostí a jejich obsluhy pomocí třídy WDB\Event\Event. K objektu této třídy můžete připojit několik objektů implementujících WDB\Event\iEventListener. Po zavolání metody raise() na objektu Event bude zavolána metoda raise se stejnými parametry na všech připojených listenerech.

Abychom nemuseli pokaždé implementovat iEventListener, nabízí WheelDB jednoduchou třídu WDB\Event\CallbackListener implementující toto rozhraní. V konstruktoru jí předáme cokoli, co přijme PHP funkce call_user_func_array (viz `http://php.net/call-user-func-array`), a právě to bude při vyvolání události zavoláno.

Následuje jednoduchá ukázka použití událostí:

```
class SquaringListener
      implements WDB\Event\iEventListener
{
  public function raise($cislo = 2)
  {
    echo "druhá mocnina čísla $cislo je ".
      ($cislo*$cislo)."\n";
  }
}
function cube($cislo = 2)
{
  echo "třetí mocnina čísla $cislo je ".
    ($cislo*$cislo*$cislo)."\n";
}
$redButton = new WDB\Event\Event();
$redButton->addListener(new SquaringListener());
$redButton->addListener(
  new WDB\Event\CallbackListener('cube')
);

$redButton->raise(3); //vypíše:
//druhá mocnina čísla 3 je 9
//třetí mocnina čísla 3 je 27
```

Zdrojový kód 5.3: Použití událostí

5.2.4 Anotace

WheelDB nabízí rozhraní pro čtení anotací v PHP komentářích. Podporovaný formát anotací je:

```
plaintext
@klic1 parametr
  pokračující na další řádce
@klic2 @klic3 parametr
```

Uvedeme příklad, jak lze anotace jednoduše načíst:

```php
/**
 * @foo bar
 * baz
 * @hello world @nice to meet you
 * @hello again
 */
class Annotated {}

$reflClass = new ReflectionClass('Annotated');

$annotations = WDB\Annotation\PhpDocReflection
                ::fromReflObject($reflClass);

echo $annotations->last('foo')."\n"; //bar\n baz
echo $annotations->hello[0]."\n"; //world
echo $annotations->hello[1]."\n"; //again
echo $annotations->first('nice'); //to meet you
```

Zdrojový kód 5.4: Čtení anotací

Metoda fromReflObject() přijímá jako parametr jakýkoli objekt poskytující metodu getDocComment(), tedy standardně všechny objekty PHP *reflection*[2] (bohužel spolu objekty reflection nesdílejí žádné rozhraní třídy, je tedy nutné kontrolovat existenci metody, nikoli implementaci rozhraní, což by více odpovídalo dobrým zvyklostem objektového programování).

Protože se jedna anotace může vyskytnout vícekrát a tedy mít více hodnot, jsou hodnoty pod každým klíčem uchovávány jako pole. Jak si můžete všimnout v ukázce 5.4, při přístupu k anotacím se používá index pole (zde 0, 1) nebo metody first a last, které vrátí hodnotu prvního resp. posledního výskytu anotace nebo NULL, pokud se anotace s daným klíčem nevyskytla ani jednou.

Zdrojem anotace nemusí být nutně PHP komentář, anotace se stejnou syntaxí lze načíst odkudkoli z řetězce, a to pomocí:

```php
$annotations = WDB\Annotation\Reflection::fromString($string);
```

Všimněte si, že zde používáme třídu Reflection místo PhpDocReflection. Toto je základní třída pro načítání anotací s jednoduchým textovým parametrem. Výraz `$annotations->hello[0]` ve skutečnosti nevrátí string, ale objekt WDB\Annotation\Data, který se metodou __toString() automaticky na string přetypuje.

Rozšiřující třídy dědící Reflection mohou anotace reprezentovat složitějšími objekty. Příkladem je právě třída PhpDocReflection, která je určená pro reprezentaci anotací dle *PhpDoc*. V současné verzi WheelDB načítá anotace `@property typ $nazev`, `@property-read typ $nazev` a `@property-write typ $nazev` do objektu třídy WDB\Annotation\Property. Ten pak umožňuje snadno objektově číst z anotace konfiguraci, ukážeme na následujícím příkladu:

```php
/**
 * @property-read string $zprava
 */
class Annotated {}
$reflClass = new ReflectionClass('Annotated');
$annotations = WDB\Annotation\PhpDocReflection::fromReflObject
```

[2] Rozhraní umožňující programu získat informace z vlastního zdrojového kódu

```
                                                    ($reflClass);

$prop = $annotations->first('property');

echo $prop->propName."\n"; //zprava
echo $prop->propType."\n"; //string
var_dump($prop->read); echo "\n"; //bool(true)
var_dump($prop->write); echo "\n"; //bool(false)
```

Zdrojový kód 5.5: Čtení anotací

5.2.5 Konfigurační soubor

Konfigurace frameworku se nachází v souboru `config.php` v kořenovém adresáři frameworku. Soubor je ve formátu PHP skriptu vracejícího vícerozměrné asociativní pole. Nejdůležitější položky konfigurace byly již popsány v kapitole 4.2.2. V této kapitole vysvětlíme, jak komponenty frameworku konfigurační soubor čtou.

Ke čtení konfigurace slouží statická třída WDB\Config se dvěma veřejnými metodami exists a read. Obě tyto metody přijímají jeden parametr — identifikátor položky konfigurace. Pro zápis identifikátoru používáme *tečkovou notaci*, tedy:

- Pokud identifikátor neobsahuje tečku, je přímo indexem pole v souboru `config.php`.
- Pokud identifikátor obsahuje tečku, použijí se symboly před tečkou jako index nejvyšší úrovně pole a za každou tečkou index nižší úrovně.

Opět vysvětlíme na příkladu:

```
var_dump(WDB\Config::exists('connections'));
//bool(true) pokud existuje klíč 'connections'

var_dump(WDB\Config::read('connections'));
//array{...}, protože pod klíčem connections je pole

var_dump(WDB\Config::read('debug'));
// bool(true|false), dle aktuálního nastavení

var_dump(WDB\Config::read('connections.default.driver'));
// např. string(MySQL), nebo NULL pokud neexistuje; ve skutečnosti
// vrací Config::$data['connections']['default']['driver']
```

Zdrojový kód 5.6: Čtení konfigurace

Výhoda tečkové notace oproti PHP zápisu přes indexy pole je v tom, že pokud na některé úrovni klíč neexistuje, metoda read() vrátí NULL a nevypisuje žádné chyby. V PHP bychom museli použít poměrně divoký zápis, abychom předešli vypisování chyb:

```

if (isset($config['connections'])
    && isset($config['connections']['default']
    && isset($config['connections']['default']['driver']))
    return $config['connections']['default']['driver']
else return NULL;
```

Zdrojový kód 5.7: Přístup k nižší úrovni vícerozměrného pole

5.3 Implementace vlastního ovladače databáze

Pro implementaci vlastního ovladače databáze je potřeba vytvořit třídu implementující třídní rozhraní WDB\SQLDriver\iSQLDriver jako ovladač databáze a třídu implementující třídní rozhraní WDB\Query\iSelectResult jako adaptér výsledku dotazu typu SELECT. V API dokumentaci je popsáno, co má která metoda dělat a jaký datový typ vracet.

Pokud implementujeme ovladač pro databázi splňující přibližně standardy SQL (říkáme přibližně, protože většina dnešních databází se standardům pouze blíží), může tento ovladač dědit třídu WDB\SQLDriver\BaseSQL. Ta poskytuje implementaci některých metod rozhraní iSQLDriver, které obsluhují to, co je obvykle SQL databázemi implementováno standardním způsobem a tudíž to ovladače SQL mohou mít společné. Ačkoli je framework navržen pro SQL databáze, je teoreticky možné napsat ovladač emulující SQL nad libovolným datovým úložištěm. V takovém případě se více hodí implementovat rozhraní iSQLDriver od začátku a třídu BaseSQL nepoužít.

5.4 Rozšiřování vrstev Wrapper a WebUI, nezávislost na vrstvě Analyzer

Na obrázku 4.2 bylo nastíněno schéma vrstev WheelDB frameworku. Vrstva Analyzer reprezentuje automaticky načtený datový model a při programování aplikace postavené na WheelDB frameworku do ní obvykle není potřeba zasahovat. Vrstva Wrapper se stará o objektový model a chování aplikace (v architektuře MVP odpovídá vrstvě Presenter a z části Model) a vrstva WebUI o generování uživatelského rozhraní (v architektuře MVC odpovídá zhruba vrstvě View). Předpokládáme, že vrstvy Wrapper a WebUI budou programátory nejčastěji rozšiřovány o vlastní implementace poskytnutých rozhraní.

Vrstva Wrapper obsahuje třídní rozhraní ve jmenném prostoru WDB\Wrapper, která můžeme implementovat při psaní vlastních aplikací používajících WheelDB framework pro přizpůsobení chování určitých tabulek a sloupců:

- iTable je třídní rozhraní objektů reprezentujících tabulku. Implementující třídy poskytují funkcionalitu např. získání dat z tabulky, kontejner sloupců, vytvoření nového řádku (záznamu) v tabulce a další. Výchozí implementace WDB\Wrapper\Table reprezentuje jednu tabulku z SQL databáze, resp. z vrstvy Analyzer. Vlastní implementace tohoto rozhraní nebo rozšíření třídy může buďto přidávat/upravovat nějakou funkcionalitu, nebo třeba načítat data z úplně jiného zdroje. **Je tedy možné vytvořit wrapper tabulky zcela nezávislý na vrstvě Analyzer**, který bude data uchovávat jiným způsobem (vzdálené úložiště, soubory, paměť)...

- iColumn reprezentuje sloupec v tabulce. Poskytuje např. informace o názvu, typu a omezeních sloupce (výchozí implementace WDB\Wrapper\Column tyto informace získává od vrstvy Analyzer) a umožňuje zkonfigurovat pravidla validace záznamu zadaného uživatelem.

- iRecord reprezentuje řádek (záznam) v tabulce. Ten získáme například jako výsledek metody getRecordByKey() volané na třídě implementující iTable

nebo z kolekce více záznamů, viz ukázka kódu 5.8. Metoda elevateRecords rozhraní iTable převede kolekci řádků výsledku SELECT dotazu z dané tabulky na kolekci záznamů.

Tyto třídy jsou řádně zdokumentovány v API dokumentaci, kde je popsáno, co má která metoda dělat a jaký datový typ vracet.

```
1 /**@var WDB\Wrapper\iTable $table*/
2 /**@var WDB\Wrapper\iRecord[] $result*/
3 $result = $table->elevateRecords(
4               $table->getDatasource()->fetch()
5             );
```

Zdrojový kód 5.8: Získání kolekce objektů WDB\Wrapper\iRecord

Vrstva WebUI obsahuje dvě obdobná třídní rozhraní (iColumn a iTable) ve jmenném prostoru WDB\WebUI, která budete implementovat, pokud budete chtít upravit výchozí uživatelské rozhraní pro práci s tabulkami:

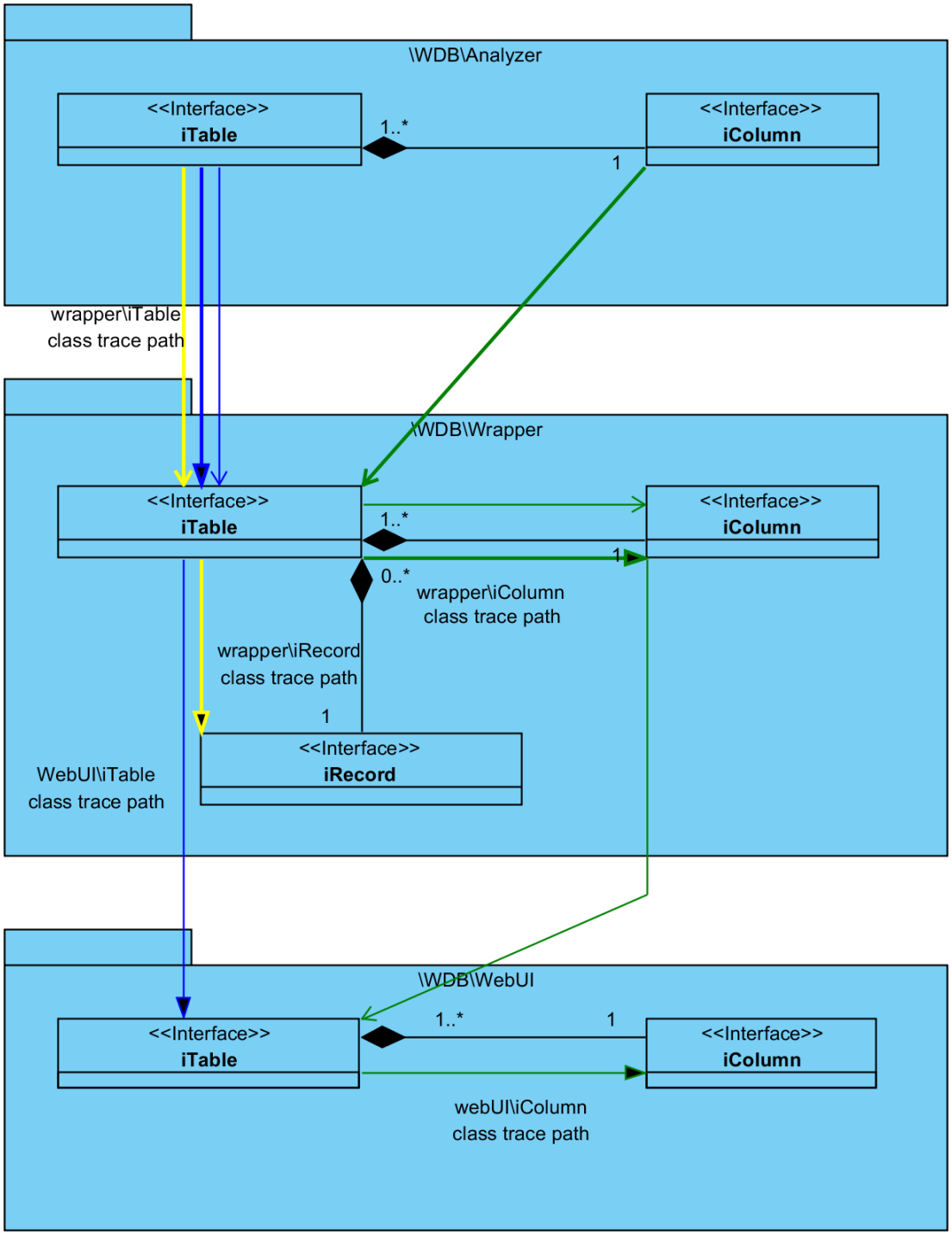
- Třída Schema reprezentuje kolekci tabulek a generuje rozhraní pro procházení tabulek. Pokud je napojeno na Wrapper vrstvu používající Analyzer, ve výchozím stavu odpovídá schématu tabulek v databázi. Zde není potřeba žádné třídní rozhraní, jelikož třídu Schema už používá přímo uživatelský kód. Komponenty frameworku na ní nejsou nijak navázány. Můžete implementovat podobnou třídu nebo rozšířit tuto a použít ji ve svém kódu dle potřeby.

- iTable reprezentuje uživatelské rozhraní tabulky. Implementující třídy poskytují propojení tabulky s objektem třídy implementující WDB\WebUI\iRequest, který zpracovává uživatelský vstup. Výchozí implementace WDB\WebUI\Table reprezentuje jednu tabulku z vrstvy Wrapper.

- iColumn poskytuje funkcionalitu uživatelského rozhraní určitého typu sloupce v tabulce. Pro různé datové typy nebo logické pohledy na data jsou různé implementace tohoto třídního rozhraní, například obyčejný sloupec s textovými daty obsluhuje třída WDB\WebUI\ColumnText.

Objekty vyšší vrstvy se standardně staví nad objekty vrstev nižších a je možné jim vyhledat příslušný protějšek v databázi podle názvu (např. metoda WDB\WebUI\TableFactory::fromName() vyhledá tabulku s daným názvem v databázi, získá k ní objekt tabulky z vrstvy Analyzer, nad ním postaví objekt vrstvy Wrapper a nad ním WebUI). Nižší vrstvy mohou pomocí anotací nebo přepsáním metod poskytujících informaci o třídě vyšších vrstev explicitně určit, jaká třída se má použít při konstrukci vyšší vrstvy.

Na schématu 5.1 je znázorněno, v jakém pořadí nižší vrstvy předávají do vyšších vrstev informaci o tom, kterou třídu mají použít. Výchozí chování většiny tříd je takové, že poskytnou informaci z nižší vrstvy nebo vrátí NULL a ponechají tak rozhodnutí o třídě na vyšší vrstvě. Například pokud objekt sloupce vrstvy Analyzer i vrstvy Wrapper explicitně určuje, jakou třídu mají použít sloupce vrstvy WebUI, přednost dostane třída určená vrstvou Wrapper.



Obrázek 5.1: Schéma konfigurace tříd vrstev Wrapper a WebUI

5.5 Vlastní validátory uživatelského vstupu

Validátory fungují na vrstvě Wrapper, kde při vyvolání metody save() na implementaci iRecord (pokud validaci daná implementace podporuje a nemá ji vypnutou) validují data uložená v objektu třídy implementující iRecord. Pokud jsou data validní, povolí se jejich uložení. Pokud ne, je vyhozena výjimka WDB\Exception\ValidationFailed. Vrstva WebUI validátory využívá k pohodlné validaci uživatelského vstupu javascriptem v prohlížeči.

5.5.1 PHP část ve vrstvě Wrapper

WheelDB v sobě obsahuje validátory popsané v kapitole 4.6. Další validátory je možné vytvářet implementací třídního rozhraní WDB\Validation\iValidator. Toto rozhraní je rozšířením WDB\Event\iEventListener, protože validace je událost, na kterou jsou navěšeny validátory a která je jako událost vyvolána. Validace tedy probíhá v metodě raise() z třídního rozhraní iEventListener, která vrací hodnotu typu boolean podle toho, zda záznam validací úspěšně prošel.

Na příkladu 5.9 si ukážeme, jak naimplementovat jednoduchý validátor, který ověří, zda je hodnota zadaného sloupce rovna „a“.

```
1  class AValidator implements WDB\Validation\iValidator
2  {
3    private $columnName;
4    public function __construct($columnName)
5    {
6      $this->columnName = $columnName;
7    }
8
9    //význam této metody bude vysvětlen v javascriptové části
10   public function fetchValidatorData
11       (WDB\Wrapper\iRecord $record, &$data)
12   {
13     if (!isset($data['aValidator']))
14     {
15       $data['aValidator'] = array();
16     }
17     $data['aValidator'][] = $this->columnName;
18   }
19
20   public function raise (WDB\Wrapper\iRecord $record = NULL)
21   {
22     if ($record === NULL)
23     {
24       throw new Exception\BadArgument(
25       "Validator event raise() method needs".
26       "WDB\\Wrapper\\iRecord as a raise argument");
27     }
28     if ($record[$this->columnName] == 'a')
29     {
30       return true;
31     }
32     else
33     {
34       $columns = $record->getTable()->getColumns();
35       $column = $columns[$this->columnName];
```

```
36          $record->getField($this->columnName)->
37              addValidationError('Sloupec '.
38                $column->getTitle().
39                ' nemá hodnotu "a" ');
40          return false;
41        }
42    }
43  }
44
45  //předpokládejme existenci tabulky foo
46  //ve výchozím databázovém spojení obsahující
47  //textový sloupec s názvem 'bar'
48  $table = WDB\Wrapper\TableFactory::fromName('foo');
49  $table->addValidator(new AValidator('bar'));
50  $record = $table->newRecord(
51          array('bar'=>'tohle není a')
52      );
53  try
54  {
55    //neprojde validací, hodnota sloupce 'bar' není validní
56    $record->save();
57  }
58  catch (WDB\Exception\ValidationFailed $e)
59  {
60    $record['bar'] = 'a';
61    $record->save(); //nyní již validací projde
62  }
```

Zdrojový kód 5.9: Implementace vlastního validátoru - PHP část

5.5.2 Javascriptová část ve vrstvě WebUI

Pro použití validátoru webovým rozhraním je potřeba naimplementovat i javascriptovou verzi. Jedinou možností, jak použít validátory napsané v PHP také na straně prohlížeče, by bylo volání PHP validátorů pomocí AJAX[3]. U některých validátorů je sice použití AJAX nutné (např. ověření unikátní hodnoty v databázi), u jiných ovšem zbytečně těžkopádné a nevhodné (např. omezení počtu znaků textového pole). Obzvláště u připojení k internetu s dlouhou odezvou by zbytečné volání AJAX způsobilo dlouhé prodlevy v odezvě uživatelského rozhraní.

Z tohoto důvodu jsme se rozhodli javascriptovou verzi validátorů oddělit od PHP verze. Programátor se tak může použití AJAX vyhnout úplně, nebo jej použít rozumným způsobem tak, aby byly prodlevy volání co nejkratší. Počítáme s tím, že validátory budou psány konfigurovatelné a tedy univerzální pro širokou škálu problémů. Nové validátory se nebudou psát příliš často, zato by měly být napsány kvalitně.

Pomocí `wdb.registerValidator('nazev', funkce)` registrujeme vlastní validační funkci. Argumentem této funkce je objekt formuláře, na kterém je validace zavolána. Z PHP předáme parametry javascriptovému validátoru implementováním metody fetchValidatorData rozhraní iValidator. Ty jsou potom naší funkci přístupné v objektu form.wdbValidation, který odpovídá vstupně/výstupní proměnné

[3] Asynchronous Javascript and XML, technologie pro komunikaci Javascriptu s webovým serverem

$data předané metodě fetchValidatorData převedené do javascriptu pomocí PHP funkce json_encode().

Na příkladu 5.10 a 5.11 si ukážeme, jak k validátoru z příkladu 5.9 přidat javascriptovou validaci. Příklad 5.10 předpokládá, že máme zinicializovanou proměnnou `$table` z příkladu 5.9. Nezapomeňme, že pro fungování javascriptového validačního engine musíme mít načtené javascriptové knihovny a případně css styly, jak bylo popsáno v příkladu 4.2 v kapitole 4.1.

```
WDB\WebUI\TableFactory::fromWrapper($table)->show();
```

Zdrojový kód 5.10: Javascriptový validátor - inicializace WebUI

```
function wdbf_AValidator(form)
{
  if (form.wdbValidation.aValidator == undefined)
  {
    return true;
  }
  var success = true;
  for (var key in form.wdbValidation.aValidator)
  {
    var column = form.wdbValidation
      .columns[form.wdbValidation.aValidator[key]];
    var value = column.element.val();
    if (value != 'a')
    {
      form.wdbValidation.eventDispatcher
        .error(form, column, 'Musí být rovno "a"!');
      success = false;
    }
    else
    {
      form.wdbValidation.eventDispatcher.success(form, column);
    }
  }
  return success;
}
wdb.registerValidator('AValidator', wdbf_AValidator);
```

Zdrojový kód 5.11: Javascriptový validátor

5.6 Jazykové verze

WheelDB nabízí jednoduchou možnost lokalizace textů. Soubory s jazykovými texty se nachází ve složce `lang` v kořenovém adresáři frameworku a jsou pojmenovány podle jazyka. Například cesta k souboru s češtinou je `lang/cs.php`.

Vlastní lokalizaci můžete vytvořit zkopírováním některého z jazykových souborů, pojmenováním dle nového jazyka a umístěním opět do složky `lang`.

Texty z jazykového souboru jsou jednoduše dostupné statickou metodou WDB\Lang::s($ident), kde argument $ident je klíč pole ze souboru jazyka. Tato metoda podporuje tečnovou notaci stejně jako metody třídy WDB\Config, viz kapitola 5.2.5.

Následuje zjednodušená ukázka jazykového souboru 5.12 (demonstruje pouze jeho strukturu, nikoli výčet všech lokalizovaných textů) a ukázka použití lokalizovaných textů 5.13.

```php
return array(
    'validator'=>array(
        'messages'=>array(
            'minlength'=>'%name musí mít minimální počet znaků: %arg',
            'minvalue'=>'%name musí být větší nebo rovno %arg',
        ),
        'excmore'=>'(a %d dalších chyb)',
    ),
    'webui'=>array(
        'schema'=>array(
            'noSelectedTable'=>'Vyberte, prosím, tabulku z nabídky',
        ),
        'newrecord'=>'Nový záznam',
        'notset'=>'Nenastaveno',
    ),
);
```

Zdrojový kód 5.12: Struktura jazykového souboru

```php
use WDB\Lang;
echo Lang::s('webui.newrecord'); //Nový záznam
echo str_replace(
    array('%name', '%arg'),
    array('Uživatelské jméno', 5),
    Lang::s('validator.messages.minlength')
); //Uživatelské jméno musí mít minimální počet znaků: 5
```

Zdrojový kód 5.13: Použití lokalizovaných textů

6. Ukázkové aplikace

V této kapitole popíšeme několik ukázkových aplikací demonstrujících použití WheelDB frameworku. Tyto aplikace jsou umístěny na přiloženém CD ve složce `/WheelDB Framework/demos`.

6.1 Instalace frameworku s ukázkovými aplikacemi

6.1.1 Požadavky

1. Pro spuštění všech ukázek je vyžadován webový server (testováno s Apache 2.2, žádné speciální požadavky na webový server framework nemá) s podporou PHP verze alespoň 5.3.

2. Pro spuštění ukázek s databázovým serverem MySQL je vyžadován MySQL server verze minimálně 5.0.

3. Pro spuštění ukázek s databázovým serverem Oracle je vyžadován databázový server Oracle. Framework je testován s verzí 11g, ovšem nemá žádné zvláštní požadavky, framework by měl fungovat i se staršími verzemi.

6.1.2 Instalace

1. Složku `/WheelDB Framework` zkopírujeme do složky, která je přístupná pod webovým serverem. Do podsložky `var` musí mít PHP oprávnění zapisovat.

2. Můžeme provést instalaci pomocí průvodce v souboru `demo.php`, který otevřeme ve webovém prohlížeči přes lokální webový server. Tento průvodce zkonfiguruje framework včetně přiložených příkladů. Žádná další konfigurace není potřebná. Pokud chceme konfiguraci provést ručně, tento bod vynecháme a pokračujeme dalším bodem.

3. Zkonfigurujeme připojení k databázi dle popisu v kapitole 4.2.2.

4. Pokud chceme používat záznamovou databázi, zavedeme ji dle popisu v kapitole 4.8.1.

5. Framework s příklady je nyní připravený. Použití jednotlivých ukázek je popsáno u každé samostatně.

Některé ukázky vyžadují další instalaci a konfiguraci. Postup je vždy popsán u konkrétního příkladu.

6.2 Web prekazky.cz

WheelDB framework jsme použili při vývoji webu `http://prekazky.cz`, jehož kopie (s pozměněnými přístupovými údaji) je umístěna na přiloženém CD ve

složce `/WheelDB Framework/demos/prekazky.cz/` (všechny cesty k souborům v této kapitole jsou zapsány relativně k této složce, pokud není uvedeno jinak). Jde o jednoduchý internetový obchod bez nákupního košíku, tedy spíše o galerii zboží, které si uživatel může objednat pomocí kontaktního objednávkového formuláře.

Jádro a webové rozhraní uživatelské části webu je vytvořeno pomocí Nette frameworku. Grafické zpracování, texty a obrázky produktů byly dodány třetí stranou a **nejsou součástí této práce**.

Tento web je dobrým příkladem webu, pro jaký je WheelDB především určen. Je velmi jednoduchý (tři SQL tabulky) a požadavek byl především na levný a jednoduchý vývoj. WheelDB se stará o dvě části webu:

- **Výpis informací o produktech** na některých stránkách, konkrétně *skokový materiál/tréninkový materiál* a všechny podstránky sekce *drezurní materiál*; zde je použita vrstva Wrapper a dotazovací rozhraní WheelDB, které předávají údaje šablonám Nette skrz *presentery*[1].
- **Administrační rozhraní** vygenerované pomocí vrstvy WebUI WheelDB frameworku.

Web demonstruje několik zajímavých funkcí a možností WheelDB frameworku, které popíšeme v podkapitolách 6.2.2 až 6.2.5.

6.2.1 Instalace a spuštění

1. Tento příklad vyžaduje nainstalovaný databázový server MySQL. Do něj importujeme data pomocí skriptu v souboru `!/prekazky.mysql.sql`.
2. Připojení k databázi pro tento příklad zkonfigurujeme v souboru `!/app/config/wdb-config.php`. Připojení k databázi obchodu umístíme pod klíč default, připojení k záznamové databázi pod klíč log. Pokud nechceme záznam používat, klíč log ze souboru odstraníme.
3. Ověříme, že do složky `!/temp` má PHP povolený zápis.
4. Uživatelskou část webu otevřeme zadáním adresy kořenového adresáře webu do webového prohlížeče.
5. Pro otevření administračního rozhraní přidáme na konec URL adresy kořenového adresáře `/admin/`. Pro přístup je potřeba uživatelské jméno a heslo. V této ukázce je nastavené uživatelské jméno admin a heslo nimdA. Tyto údaje lze změnit v souboru `!/app/config/app.neon`.

6.2.2 Propojení s Nette frameworkem

Tento web ukazuje, jak snadno lze WheelDB začlenit do jiného frameworku. WheelDB dodává Nette frameworku modelovou vrstvu, která nahrazuje vestavěnou Nette\Database postavenou na knihovně NotORM (srovnání s WheelDB v kapitole 3.1). Změnili jsme pouze nastavení některých adresářů — není to sice nutné,

[1] *Presenter* je element prezentační vrstvy Nette frameworku; více informací o presenterech lze nalézt na adrese `http://doc.nette.org/cs/presenters`, ale pro pochopení tohoto příkladu je není potřeba znát.

ale vyhneme se tak duplikování dočasných adresářů a adresářů s konfiguračními soubory.

WheelDB framework je do aplikace zaveden na řádku 13 souboru `!/app/bootstrap.php`. Na dalším řádku je nastavena cesta ke konfiguračnímu souboru, který je umístěn do `!/app/config/wdb-config.php` spolu s konfiguračními soubory Nette. V tomto konfiguračním souboru jsou také nastaveny cesty k některým složkám WheelDB.

Ukázku integrace vrstvy Wrapper do Nette *presenteru* nalezneme v souboru `!/app/presenters/ProductsPresenter.php`. Zde jednoduchým způsobem pomocí WheelDB získáme název aktuální kategorie a seznam produktů pro další použití v Nette šablonách.

Integrace vrstvy WebUI do administračního rozhraní je také jednoduchá.

- V souboru `!/app/controls/wdbSchema.php` je vytvořena Nette komponenta wdbSchema, která slouží jako adaptér mezi rozhraním vygenerovaným vrstvou WebUI a šablonovým systémem Nette frameworku.
- Jelikož se Nette komponenty generují ze šablon, potřebujeme šablonu `!/app/controls/wdbSchema.latte`. Protože se o vygenerování HTML kódu této komponenty stará WheelDB framework, šablona pouze vypisuje proměnnou `$control`, kterou přijme od skriptu v souboru `wdbSchema.php`.
- Metoda `AdminPresenter::createComponentWdbAdmin` v souboru `!/app/presenters/AdminPresenter.php` zprostředkuje komponentu wdbSchema Nette šabloně zápisem `{control wdbAdmin}`, jak můžeme vidět v souboru šablony `!/app/templates/Admin/default.latte`.

6.2.3 Vlastní typ sloupce bez protějšku v databázi

Ve zdrojovém kódu tohoto webu můžeme nalézt ukázku, jak lze do WheelDB přidat vlastní typ sloupce, v tomto případě jde o sloupec obsahující obrázky. Jelikož obrázky nejsou uloženy v databázi, ale v souborech, jde navíc o ten zvláštní případ, kdy sloupec vrstvy Wrapper nemá protějšek v databázi a je tedy naimplementován bez vazby na vrstvu Analyzer. Tento sloupec je implementován pomocí PHP skriptů ve složce `!/app/wdb/`:

- `ImageColumnWrapper.php` je implementací sloupce typu ImageColumn vrstvy Wrapper, který slouží pro ukládání obrázků. Obsahuje logiku čtení a ukládání obrázku do souboru. Soubor je identifikován pomocí jiného sloupce vrstvy Wrapper (na tomto webu vždy číselný primární klíč), jehož hodnota je vložena do názvu souboru.
- `ImageColumnWebUI.php` je generátor webového rozhraní — HTML kódu pro nahrání a zobrazení obrázku.
- `ImageTarget.php` obsahuje pomocnou strukturu pro ImageColumn, která určuje, kam má být obrázek uložen, pod jakou URL je dostupný a jakou má mít velikost. Konstruktor ImageColumnWrapper může přijmout pole těchto struktur a tak při nahrání jednoho obrázku vytvořit více různých velikostí, například malý náhled a plnou velikost.

- `GalleryTable.php` a `ProductAdminTable.php` jsou implementace tabulek vrstvy Wrapper využívajících sloupec ImageColumn.

6.2.4 Konfigurace pomocí anotací

Pomocí anotací přímo v databázových komentářích jsou nastaveny popisky tabulek a sloupců, zobrazení sloupců v jednotlivých režimech a je skryta tabulka product_categories. Tato tabulka obsahuje seznam kategorií produktů, který ale bohužel uživatel nemůže měnit, protože je pevně dán designem webu. Tabulka products se na tabulku product_categories odkazuje, tudíž v administraci produktů můžeme vidět a měnit kategorie jednotlivých produktů.

Dále je pomocí anotací potomka třídy WDB\Structure\Structure vytvořena zmíněná struktura ImageTarget.

6.2.5 Přenositelnost na databázi Oracle

Připravili jsme skript pro zavedení databáze Oracle se strukturou a daty shodnými s MySQL verzí tohoto příkladu. Tím demonstrujeme, jak snadno lze s WheelDB používat různé databáze a případně je měnit i pod již hotovou webovou aplikací.

V případě, že by webová aplikace využívala funkce dostupné pouze na některých ovladačích databáze, nebylo by možné ji používat v kombinaci s databází, jejíž ovladač tyto funkce nepodporuje. Tento příklad ale žádné takové funkce nevyužívá, takže přechod na databázi Oracle je možný.

Instalace

1. Do databáze Oracle importujeme soubor `!/prekazky.oracle.sql`.
2. V souboru `!/app/config/wdb-config.php` změníme název připojení default nebo jej celé smažeme/zakomentujeme a připravenou konfiguraci připojení s názevem oci přejmenujeme na default.

6.3 Databáze půjčovny

Tuto aplikaci jsme vytvořili jako ukázku některých dalších funkcí, které web *prekazky.cz* nepokryl. Jde o jednoduchou databázi půjčovny médií, která eviduje tituly v půjčovně, zákazníky a výpůjčky. Pomocí *SQL pohledů*[2] je možné zobrazit, kolik exemplářů kterého titulu je volných, případně exempláře kterých titulů jsou všechny rozpůjčované.

Aplikace je ukázkou zejména těchto funkcí WheelDB frameworku:

- **Konfigurace modelu pomocí NeonWheel**: protože *SQL pohledy* v MySQL databázi nemohou mít vlastní komentáře, varianta konfigurace modelu této aplikace pomocí SQL komentářů by byla nedostačující. Proto jsme vytvořili ukázku konfigurace modelu pomocí NeonWheel, viz kapitola 4.7.4. Konfigurace modelu se nachází v souboru `models/pujcovna.nw` v adresáři WheelDB frameworku.

[2] *SQL pohled* je předpřipravený SELECT dotaz v databázi, k němuž lze přistupovat jako k tabulce pro čtení.

- **Vlastní validátor**: půjčovna obsahuje vlastní validátor záznamů tabulky výpůjček: při zadání výpůjčky kontroluje, zda je vybraný titul volný k půjčení a zda zákazník dosáhl potřebného věku pro vypůjčení věkově omezeného titulu. Třída tohoto validátoru CanBorrowValidator se nachází v souboru `pujcovna.php` umístěném ve složce tohoto příkladu. K tabulce výpůjček je připojen pomocí wrapper třídy VypujckyTW, která se nachází v tomtéž souboru.

- **Datové typy sloupců**: použili jsme tyto další datové typy, které se v předchozím příkladu *prekazky.cz* nevyskytovaly. Datové typy enum a set se dají nahradit a v rozsáhlejších aplikacích se obvykle nahrazují cizími klíči, ovšem v jednoduchých aplikacích jako je tato ukázka mohou usnadnit práci.
  - **zakaznici.mail**: sloupec validovaný jako e-mail v platném formátu (validační pravidlo email)
  - **zakaznici.telefon**: sloupec validovaný dle regulárního výrazu jako platné telefonní číslo (validační pravidlo pattern)
  - **zakaznici.datum_narozeni**: sloupec typu datum
  - **vypujcky.vratit**: sloupec typu datum, který může obsahovat hodnotu NULL — v uživatelském rozhraní reprezentováno zaškrtávacím políčkem před polem pro zadání data
  - **vypujcky.vracena**: sloupec typu boolean; jelikož většina SQL databází nepodporuje nativně sloupce boolean, ale používá místo nich sloupec typu integer s hodnotou 1 nebo 0, není možné typ boolean odvodit ze struktury databáze a musí být zvolen v konfiguraci modelu zadáním `webui WDB\WebUI\ColumnBoolean`.
  - **tituly.zanry**: seznam žánrů daného titulu reprezentovaný datovým typem set (specifický pro MySQL databázi, Oraclem není podporován)
  - **tituly.vek**: věkové omezení pro vypůjčení daného titulu reprezentované datovým typem enum (opět specifický pro MySQL databázi, Oraclem není podporován)

- **Validátor unikátních klíčů**: V tabulce zákazníků musí být unikátní telefon, e-mail a dvojice (jméno, datum narození). Tato podmínka je zadána pouze unikátními klíči na úrovni databáze, WheelDB podle ní automaticky přiřadí záznamům tabulky zákazníků třídu validátoru WDB\Validation\UniqueConstraint, která před pokusem o uložení do databáze unikátnost ověří.

6.3.1 Instalace a spuštění

1. Zdrojový kód tohoto příkladu se nachází v podsložce `demos/pujcovna` složky WheelDB frameworku na přiloženém CD.

2. Příklad vyžaduje nainstalovaný databázový server MySQL. Do něj importujeme data pomocí skriptu v souboru `pujcovna.mysql.sql`.

3. Připojení k databázi pro tento příklad nastavíme v souboru `config.php` ve složce příkladu pod klíč default, připojení k záznamové databázi pod klíč log. Pokud nechceme záznam používat, klíč log ze souboru odstraníme.

4. Příklad otevřeme zadáním adresy kořenového adresáře webu do webového prohlížeče.

6.4 Utilita pro opravu znakové sady databáze

Tato utilita vznikla z praktické potřeby a demonstruje užitečnost WheelDB frameworku i pro zcela jiný typ aplikací, než jsou webová rozhraní pro procházení a úpravy databáze.

Začínajícím PHP programátorům se často stává to, že neví o nutnosti nastavit správnou znakovou sadu pro připojení k databázi. Na první pohled žádný problém nevznikne, protože data obvykle zapisuje i čte stejná PHP aplikace. Ta databázi sdělí nějakou výchozí znakovou sadu pro připojení, která nemusí odpovídat skutečnosti. Když aplikace čte data, která předtím s touto špatnou konfigurací zapsala, kombinace znakové sady pro spojení s databází a skutečné znakové sady čtených dat zůstává stejná, text je tedy načten ve stejné podobě, jako byl uložen.

Problém vyvstane například ve chvíli, kdy je k databázi přistoupeno jinou aplikací, třeba utilitou *phpMyAdmin* sloužící pro správu databáze nebo podobnou. Ta nemá informaci o chybně uvedené znakové sadě, proto zobrazí texty z databáze tak, jak jsou v ní skutečně uložené, tedy chybně. Takovou databázi je pak často obtížné exportovat, protože nástroj pro export může na některých chybách v znakové sadě zhavarovat (například ne všechny sekvence bytů jsou platné utf8 řetězce) nebo při pokusu o převod znakové sady text nevratně poškodit. Mohou ale nastat i jiné problémy, například nebude správně fungovat národní abecední řazení na úrovni databáze a další.

Utilita *encorep* z ukázkových programů WheelDB slouží k vyřešení tohoto problému. Stačí zadat údaje pro připojení k databázi, dosud chybně uvedenou a skutečnou znakovou sadu a utilita postupně všechny záznamy převede do správné znakové sady. Utilita je napsána pro databázi MySQL, ve které tento problém nastává často, bylo by ji ale možné jednoduše upravit i pro ostatní platformy.

Vzhledem k použitému řešení si utilita neporadí s tabulkami, které nemají primární klíč nebo jsou jeho součástí právě špatně kódované texty. U takových tabulek se totiž změnou znakové sady změní klíče jednotlivých řádků. I tento problém by bylo možné řešit, pokud bychom uchovávali klíč k těmto tabulkám ve správné i chybné znakové sadě a buďto napsali potomka třídy WDB\Wrapper\Record s úpravou identifikace řádků, nebo data ukládali vytvořením UPDATE dotazu mimo tuto třídu. Jelikož jsou v drtivé většině MySQL databází tabulky identifikovány číselným klíčem nebo znakovým identifikátorem bez diakritiky, rozhodli jsme se, že tuto ukázku pro jednoduchost nebudeme komplikovat řešením zmíněného problému se znakovou sadou primárního klíče.

6.4.1 Instalace a spuštění

Pro nasimulování problému slouží skript `demos/encorep/create-bad-encoding.php` ve složce WheelDB frameworku na přiloženém CD. V tomto souboru je

nastaveno připojení k MySQL databázi na server localhost s uživatelským jménem i heslem *root*. Tyto údaje můžeme upravit přímo v tomto souboru. Skript vytvoří schéma *badencoding* a v něm tabulku *foo* s textem v chybné znakové sadě. Nyní můžeme tabulku zobrazit například v *phpMyAdmin*, který zobrazí text „žluťoučký kůň pěl ďábelské ódy“, ovšem s chybými znaky místo diakritiky.

Následně otevřeme ve webovém prohlížeči skript `demos/encorep/encorep.php` (skrz webový server, nikoli pouze otevřít soubor). Objeví se formulář, ve kterém — pokud je to potřeba — upravíme údaje pro připojení k databázi, zadáme jméno schématu *badencoding* a klikneme na *opravit kódování*. Nyní se můžeme v *phpMyAdmin* přesvědčit, že je text již zobrazen správně.

7. Další možnosti rozšiřování projektu

Cílem této práce je představit WheelDB framework jako nový pohled na vytváření jednodušších aplikací, které je potřeba vytvořit rychle a levně v přijatelné kvalitě. Již v této verzi je pro jednoduché projekty dobře použitelný a šetří spoustu času při vývoji databázové části aplikací, víme ale o spoustě možných vylepšení, která rozšíří spektrum aplikací, pro než je framework vhodný, a omezí případy, kdy je určitý úkon s použitím WheelDB složitější než bez něj[1]. Uvedeme zde ta nejpodstatnější vylepšení, která by se měla implementovat, pokud se framework v praxi osvědčí.

Podkapitoly jsou seřazeny v přibližném pořadí, v jakém by je bylo vhodné implementovat. Rozhodující pro seřazení je jejich důležitost pro framework a náročnost na implementaci.

7.1 Generátor minifikované verze

Webové i jiné aplikace psané v interpretovaných programovacích jazycích často procházejí před nasazením do produkčního prostředí procesem *minifikace*. Ten zdrojové kódy upraví například těmito způsoby:

1. Odstraní ze zdrojového kódu komentáře.
2. Kde je to možné, odstraní mezery a další bílé znaky.
3. Kde bílé znaky musí být, nechá jich nejmenší možný počet.
4. Pokud je zdrojový kód rozdělen do více souborů a je-li to možné, spojí je.
5. Zkrátí názvy proměnných, funkcí/metod a další identifikátory, které jsou viditelné pouze v rozsahu minifikovaného zdrojového kódu a tudíž se změna jejich názvu neprojeví navenek.

Ztráta komentářů a formátování zdrojového kódu v produkčním prostředí nebo při používání klientem není žádný problém, tyto informace jsou užitečné pouze při vývoji. Zdrojový kód zabírá při distribuci zákazníkovi nebo nasazení do produkce méně místa a je částečně *obfuskovaný* (vysvětlíme dále). V případě PHP aplikací, a tedy i WheelDB frameworku, je ale především výhoda v tom, že jsou zdrojové kódu **spojeny do jednoho souboru**. Načítání jednoho většího souboru je rychlejší než načítání velkého množství malých souborů po jednotlivých třídách. Toto urychlení je významné především na PHP serverech běžících pod systémem Windows, které jsou ve zpracování PHP někdy znatelně pomalejší než linuxové stroje se srovnatelným výkonem hardwaru.

[1] Množství takových případů se pochopitelně vždy snažíme minimalizovat. Avšak každý framework si s sebou nese určitá pravidla a programátorské postupy, podle kterých je navržen. V některých situacích, které nebyly při návrhu frameworku zohledněny, je potom nutné buď framework obejít, nebo složitým kódem docílit toho, aby byla situace vyřešena v rámci koncepce frameworku.

Plánovaným rozšířením frameworku je vyvinout pomocný PHP skript, který z vývojářských zdrojových kódů automaticky vygeneruje minifikovanou verzi. Náš skript provede výše zmíněné kroky č. 1–4. Díky tomu, že má PHP vestavěné fukce pro rozdělení zdrojového kódu PHP načteného do řetězce na tokeny, kroky 1–3 nepředstavují problém, musíme pouze zachovat komentáře s anotacemi, které mají ve WheelDB vliv na běh programu.

Nejsložitější a nejdůležitější je krok č.4, spojení do jednoho souboru. Dokud máme každou třídu a třídní rozhraní uložené v samostatném souboru a ty načítá PHP speciální funkcí automaticky ve chvíli, kdy jsou potřeba, není problém se závislostí jedné entity na druhé (tj. dědění/implementace třídy/rozhraní). Pokud ale třídy a rozhraní spojíme do jednoho souboru, musí v něm být uvedeny tak, aby se třída závislá na jiné třídě nebo třídním rozhraní dostala do kompilátoru až po nich. Je tedy potřeba určit závislosti a na jejich základě definice vhodně seřadit. To je vždy možné, protože nemůže existovat cyklická závislost mezi třídami a třídními rozhraními — znamenalo by to, že třídy nebo rozhraní tranzitivně dědí samy sebe, což v objektovém programování nedává smysl.

Procesem blízkým minifikaci je také „obfuskace“, která má za cíl zhoršit čitelnost publikovaného zdrojového kódu, pokud není žádoucí, aby v něm kdokoli, komu se dostane do ruky, mohl snadno dělat změny, nebo třeba jen vyčetl, jak funguje.

Narozdíl od minifikace jsou nejdůležitějšími kroky obfuskace přejmenování identifikátorů a odstranění komentářů, čímž se z kódu vytratí lidské názvy toho, co která část dělá. Smazání bílých znaků sice také zhorší čitelnost zdrojového kódu, ale existují nástroje, které provedou opačný proces — zformátují kód podle zadaných pravidel tak, aby byl dobře čitelný.

Žádná obfuskace nemůže zajistit stoprocentní ochranu kódu, pouze jeho čtení a modifikaci o určitou úroveň ztíží.

Jelikož WheelDB framework není produkt pro běžného uživatele počítače, ale nástroj pro programátory, jehož originální zdrojové kódy jim budou dostupné, zaměřovat se na obfuskaci jeho kódu nemá význam.

7.2 Zjednodušení zadávání dotazů

V kapitole 4.3.1 jsme popsali, proč jsme se ve WheelDB rozhodli pro objektovou reprezentaci dotazů a její výhody a nevýhody. Dalším důležitým vylepšením frameworku je zjednodušení tvorby dotazů, aby zápis prostého porovnání nezabral tři řádky kódu jako v příkladu 4.4.

První jednoduchou možností, jak tvorbu dotazů zjednodušit, je vytvoření zkrácených aliasů a funkcí pro současné třídy reprezentující dotazy. Po takovém vylepšení by mohl zmíněný kód vypadat například takto:

```
1 use WDB\Query\B;
2 function uprav(WDB\Query\Select $select)
3 {
4    $select->cond(B::eq(B::col('foo'),B::str('bar')));
5    $select->removeField('baz');
6
7 }
```

Zdrojový kód 7.1: Možná podoba práce se SELECT dotazem

To je jednoduché a mnohem přehlednější řešení, stále je to ale méně pohodlné, než zápis v jazyce SQL. Jako optimální řešení plánujeme připravit pro WheelDB nástroj pro vybudování objektové reprezentace dotazu nebo jeho části z řetězce v jazyce SQL nebo jeho dialektu, takže programátor bude moci dotaz pohodlně zapsat stejně jako v jiných databázových frameworcích a dále s ním pracovat objektově a využít tak tuto výhodu WheelDB.

7.3 Širší možnosti konfigurace

WheelDB framework lze vylepšit o mnoho možností konfigurace, aby bylo možné provádět více obvyklých úprav tím, že pouze zkonfigurujeme datový model a nebudeme muset požadovanou funkcionalitu programovat. Uvedeme několik příkladů:

- u pohledů (VIEW), které nemají primární klíč, definovat klíč (tedy množinu sloupců, jejichž kombinace hodnot je vždy unikátní) virtuálně na úrovni frameworku, aby bylo možné snadno přistupovat k jednotlivým záznamům
- možnost definovat sloupce pro výchozí řazení záznamů v tabulce při výpisu záznamů a různé další zásady výchozího chování
- umožnit nastavení snadno dynamicky měnit, například pro jednu tabulku mít jedno nastavení pro přístup z administračního rozhraní a druhé pro veřejný přístup a přepínat mezi těmito nastaveními; V současnosti lze jednotlivá nastavení měnit pomocí metod objektů reprezentujících databázové entity, ale není zatím implementován postup, který by umožnil načíst jednoduše zapsané kompletní nastavení.

7.4 Rozlišení oprávnění dle uživatelů

V kapitole 4.7.2 jsme popsali, jak zkonfigurovat přístupové oprávnění do jednotlivých tabulek z webového rozhraní WheelDB. V praxi je ale často jednotná konfigurace pro prvky datového modelu nedostačující, například zatímco běžný uživatel může určitá data pouze zobrazovat, administrátor je může i měnit. To je nyní možné, můžeme po načtení tabulky změnit přístupová oprávnění ručně na základě přihlášeného uživatele, není to ale příliš pohodlné.

WheelDB bude poskytovat třídní rozhraní, které programátor implementuje jako třídu konfigurace oprávnění, a tu potom předá frameworku. Klíčová metoda tohoto rozhraní bude přijímat jako parametr název tabulky nebo sloupce, pro které vrátí strukturu oprávnění. Tato třída může dynamicky rozhodovat, jaká oprávnění aplikovat na danou tabulku, například dle aktuálně přihlášeného uživatele.

WheelDB zároveň poskytne jednoduchou vlastní implementaci tohoto třídního rozhraní načítající konfiguraci z nějakého standardního datového formátu, např. XML, YML nebo PHP pole.

7.5 Vylepšení vrstvy WebUI

Současná verze WheelDB obsahuje pouze jednoduchou implementaci HTML prvků pro vkládání různých datových typů. V budoucnu by framework měl obsahovat prvky pro pohodlné zadávání různých datových typů, zejména kalendář pro datum a čas.

Dále bychom chtěli implementovat volitelné filtrování podle určité hodnoty sloupce.

Uživatelsky přívětivé vylepšení by byl speciální typ výpisu záznamů, který by umožnil záznamy editovat přímo bez nutnosti přechodu na stránku editace. Práce s takovým prvkem by mohla vypadat jako práce s jednoduchými daty v tabulkovém procesoru, zpočátku bez možnosti používání rozličných funkcí (součet buněk apod.). Ty je samozřejmě také možné později implementovat, ale vzhledem k účelu tohoto prvku — jednoduché úpravě záznamů v tabulce — je nepovažujeme za nezbytné. Naopak užitečný by mohl být nástroj umožňující jednoduše změnit hodnotu více záznamů najednou — například si výše zmíněným filtrováním vypsat pouze určitou množinu záznamů a pak všem nastavit určitou hodnotu v nějakém sloupci.

7.6 Inteligentní podpora vztahů *m:n*

V rozsahu tohoto projektu jsme nepodchytili žádné speciální zjednodušení práce s databázovými entitami ve vztahu *m:n*, tedy vztah mezi dvěma tabulkami, ve kterém se může jeden záznam z obou tabulek vyskytovat vícekrát. V relačních databázích se tento vztah řeší pomocnou tabulkou vyjadřující tuto relaci. Jako příklad uvedeme studenty, kteří se mohou hlásit na více zkoušek, a předměty, na jejichž zkoušky může být přihlášeno více studentů (zjednodušeno, neřešíme zde např. klíče):

```
CREATE TABLE studenti (
    id_student INT,
    jmeno VARCHAR(50)
);
CREATE TABLE predmety (
    id_predmet INT,
    nazev VARCHAR(50)
)
CREATE TABLE prihlaseni_zkousky (
    id_student INT,
    id_predmet INT
)
```

Zdrojový kód 7.2: Vztah *m:n*

V SQL databázích neexistuje přímá implementace vztahu *m:n* přímo mezi dvěma tabulkami, a vzhledem k tomu, že se řeší takto třetí tabulkou, je možné ve WheelDB včetně WebUI rozhraní s relací přesně takto pracovat — záznamy první a druhé tabulky spravujeme odděleně a vztah mezi nimi definujeme ve třetí tabulce.

Možné vylepšení by bylo implementovat speciální podporu pro vztahy typu *m:n*, díky které bychom mohli pomocí speciálního prvku uživatelského rozhraní například naklikat, na které zkoušky je student přihlášen. Problémem zde je, jak

strojově rozpoznat, zda je relační tabulka pouze reprezentací vztahu *m:n*, nebo se jedná o skutečnou entitu se vztahy *1:n* do dvou dalších tabulek. Například známkování zkoušek by mohlo vypadat takto:

```
CREATE TABLE znamkovani (
  id_student INT,
  id_predmet INT,
  znamka INT DEFAULT NULL
);
```

Zdrojový kód 7.3: Tabulka vztahu *m:n* spolu s dalšími daty

Zde dokonce tabulka může vystupovat v obou rolích: pohodlné přihlášení studentů na zkoušky bychom mohli provést z editace záznamu studenta, ale při doplňování známek by bylo vhodné přistupovat k této tabulce jako samostatné (předpokládejme, že vyučující doplňuje známky z více předmětů zároveň, takže doplňování známek z editace záznamu předmětu by také nebylo vhodné řešení).

Nejlepší řešení se nám jeví vytvořit rozhraní tak, aby bylo možné v takovémto případě v programu nastavit, jakým způsobem se má k relaci přistupovat. Pro zachování principu *konvence před konfigurací* určíme vhodné výchozí chování.

To by bylo pravděpodobně takové, že k tabulce, která se skládá pouze ze dvou cizích klíčů a žádného jiného sloupce, se bude přistupovat jako ke vztahu *m:n* a nebude zobrazována samostatně. Tabulka, která má sloupců více, nebude implicitně chápána jako vztah *m:n* a bude zobrazována pouze samostatně.

7.7 Propojení s existujícími frameworky

Na začátku projektu jsme zvažovali, zda vyvíjet vlastní řešení různých částí, např. generování webového rozhraní v HTML, nebo použít nějaké existující frameworky. Rozhodli jsme se, že základní verze frameworku bude obsahovat pro všechny důležité části vlastní řešení. Sice nebudou propracované v takovém rozsahu, ale umožní, aby byl WheelDB framework co nejméně závislý na prostředí a nevyužíval ke svému běhu další frameworky. Kromě základních rozšíření PHP je jedinou použitou externí knihovnou jQuery, která je dnes již téměř standardem webových aplikací používajících javascript.

Pro budoucí verze frameworku by ale mohla existovat alternativa, která umožní framework integrovat s jinými knihovnami a frameworky. Například pro generování webového rozhraní by byl dobře použitelný Nette framework (`http://nette.org`), který má propracované objektové rozhraní pro generování a obsluhu formulářů včetně validace a pěkný šablonovací systém pro generování HTML kódu.

Dále jsme zvažovali, zda je lepší takovýto produkt postavit na vlastní DBAL vrstvě, nebo použít například Doctrine (srovnávaný v kapitole 3.2). U této ani žádné jiné knihovny jsme ale nenalezli podporu pro čtení a analýzu struktury a metadat databáze, na jejichž základě generujeme webové uživatelské rozhraní. Proto jsme vytvořili DBAL vrstvu v této podobě, která mimo to nabízí i další výhody, jako je objektová reprezentace dotazů. WheelDB by měl sloužit jako plná alternativa přístupu k databázi a pokud bychom psali databázový ovladač přistupující k databázi skrze například Doctrine, byla by to zcela zbytečná mezivrstva. Pokud programátor preferuje psaní dotazů například v Doctrine a zároveň chce používat nějakou funkcionalitu WheelDB, je možné oba frameworky používat vedle sebe, pro složitější projekty ale považujeme za lepší si jeden vybrat.

7.8 Sledování změn v databázi

Jak jsme popsali v kapitole 4.8, WheelDB podporuje záznam provedených dotazů do speciální databáze. V současné verzi se záznamovými informacemi WheelDB dále nepracuje, pouze je ukládá.

Pokud bychom rozšířili zaznamenávané informace a integrovali zpětně s aktivní částí programu, bylo by možné vytvořit například interaktivní sledování změn v databázi. Programy napsané pomocí WheelDB by pak mohly procházet a případně anulovat provedené změny. K tomuto by samozřejmě WheelDB mohl také poskytnout generátor uživatelského rozhraní.

7.9 Podpora perzistence smazaných záznamů

V různých databázových aplikacích existují data, která se nemohou reálně smazat kvůli uchování historie, přesto je ale potřeba je nějak označit za smazané. Například mějme IT firmu, která má v databázi tabulku zaměstnanců a tabulku úkolů, které mají splnit. Pokud zaměstnanec firmu opustí, je vhodné jej nějak označit za smazaného, aby mu dále nešly přiřazovat úkoly ke splnění. V databázi ale jeho údaje musí zůstat, protože uchovává úkoly, na kterých měl během svého působení pracovat.

Tento problém se dá řešit mnoha způsoby, například těmito:

1. Zaměstnance z tabulky mažeme a historii uchováváme jiným způsobem, například export do souborů nebo do jiné tabulky. Tento způsob je sice jednoduchý, ale není moc šikovný. Nutí programátora pracovat s historickými daty jinak než s přítomností, což vede buďto k duplicitnímu, nebo ve výsledku zbytečně komplikovanému kódu.

2. Tabulce zaměstnanců přidáme nový sloupec — příznak typu boolean určující, zda je zaměstnanec smazán. Tento způsob je již mnohem šikovnější a v praxi použitelný, historii i současnost uchovává jednotně a jednoduchou podmínkou v SQL dotazu vybereme, se kterými zaměstnanci chceme pracovat - aktivními, smazanými či obojími.

3. Podobně jako v předchozím bodě přidáme tabulce zaměstnanců příznak, ale nikoli typu boolean, ale časové razítko s možností nabývat hodnoty NULL. V principu jde o stejný přístup, ale informace časového razítka nabízí více možností. Časové razítko pomůže rozeznat tyto případy:

   - NULL znamená, že zaměstnanec **je aktivní** a není známo, kdy svou účast ve firmě ukončí.
   - časové razítko **menší** než přítomnost značí, že zaměstnanec v současnosti **již není aktivní** a není možné mu přidělovat žádné úkoly.
   - časové razítko **větší** než přítomnost značí, že zaměstnanec **je stále aktivní**, ale již má určen čas ukončení svého působení ve firmě a není možné mu přiřazovat úkoly, na kterých se má pracovat až po tomto čase.

- Časové razítko **menší než přítomnost**, ale **větší než určité období** v historii značí, že je zaměstnanec pro dané období stále aktivní, ačkoli v přítomnosti již není. Pokud například administrátor opravuje chybně zadanou databázi úkolů z minulého měsíce, po jehož skončení zaměstnanec odešel, má možnost mu na daný měsíc dodatečně přiřadit úkol.

4. Předchozí případ ještě můžeme vylepšit o další příznak - časové razítko vzniku záznamu (kdy zaměstnanec do firmy nastoupil). Bude fungovat analogicky jako ukončení, určí, od jakého času je možné zaměstnanci nejdříve přiřazovat úkoly.

První způsob je nešikovný a druhý plně nahraditelný třetím. WheelDB by mohl mít vestavěnou podporu v podobě třetího nebo čtvrtého způsobu. Pomocí anotací by bylo možné označit, které tabulky mají být takto perzistentní a který sloupec slouží jako příznak smazání či vytvoření.

Tlačítko „smazat“ u záznamů takové tabulky by pak záznam fyzicky nesmazalo, ale nastavilo mu příznak ukončení na aktuální časové razítko. Při procházení záznamů tabulky by se dalo volit, zda zobrazovat aktivní či smazané záznamy, nebo ještě lépe zobrazit záznamy aktivní v zadaném časovém rozsahu. Tabulky vázané na perzistentní tabulku přes cizí klíč by mohly brát tyto příznaky v úvahu a nezobrazovat ve výběru záznamu připojeného cizím klíčem položky, které v současnosti nejsou aktivní.

7.10 Optimalizace dotazů

V kapitole 3.1 jsme popsali srovnání WheelDB frameworku s knihovnou NotORM a za největší výhodu NotORM jsme označili optimalizované dotazy. V současné verzi WheelDB jsou dotazy generované vrstvami Wrapper a WebUI vytvářeny přímou cestou bez optimalizací. Dalším možným vylepšením je tedy implementace podobného optimalizačního mechanismu do WheelDB.

Závěr

Naším cílem v bakalářské práci bylo představit nový pohled na vývoj jednodušších aplikací, které je potřeba vytvořit rychle za nízkou cenu, ale přesto se od nich očekává určitá minimální úroveň kvality. WheelDB framework tento problém řeší pro případ webových databázových aplikací. Na ty jsme se zaměřili, protože s nimi již máme bohaté zkušenosti a měli jsme už na začátku práce dobrou představu, jak k problematice přistupovat a na co si především dát pozor.

V textu práce jsme kromě dokumentace programu řešili zejména srovnání s frameworky zabývající se podobnou problematikou a další možnosti rozvoje frameworku.

Na základě poznatků nabytých při psaní této práce můžeme zhodnotit současnou situaci takovýchto frameworků. Podařilo se nám vytvořit framework, který opravdu umožní vytvoření prototypu určitého typu aplikace pouhými třemi řádky zdrojového kódu. Přitom ale programátor není frameworkem nijak výrazně omezen, chování frameworku může libovolně upravovat a rozšiřovat až do požadované podoby výsledné aplikace. V jednoduchých případech, kterých je ostatně v případě databázových aplikací mnoho, dokonce téměř žádné další úpravy nemusí být vůbec potřeba.

Ačkoli mají různé konkurenční produkty propracovanější některé části, žádný dosavadní framework v PHP nenabízí tak komplexní propojení vrstev aplikace od datového modelu až po uživatelské rozhraní. Většina z nich se zaměřuje pouze na omezenou problematiku, a tak musí programátor obvykle spojovat více frameworků a knihoven, které jsou spolu více či méně kompatibilní, pro vytvoření koncové aplikace. Záležitosti, které je potřeba řešit na více úrovních aplikace, například datové typy a validace, pak často vedou k duplicitnímu řešení na více místech, což nebývá programátory považováno za dobré.

Na začátku psaní práce jsme se rozhodli ve frameworku implementovat vlastní řešení většiny problémů souvisejících s databázovými aplikacemi jako například dotazovací rozhraní nebo generování HTML formulářů. Díky tomu WheelDB framework nevyžaduje pro použití žádné další knihovny ani frameworky vyjma javascriptového jQuery, který je dnes prakticky standardem webových aplikací.

U dotazovacího rozhraní framework nabízí formu konstrukce dotazů pomocí objektů, jakou jsme u jiných frameworků nenalezli, problematika efektivního dotazovacího rozhraní se ale ukázala být velmi rozsáhlá a v rámci WheelDB frameworku zdaleka nedosáhla možného potenciálu.

WheelDB ukázal, že v případě frameworků je stále velký prostor pro rozvoj a programátorský čas potřebný k vývoji aplikace s pomocí frameworků je možné až překvapivě minimalizovat. V současné verzi je WheelDB funkčním celkem, který tato očekávání splnil, pro opravdu pohodlné používání programátory by ale ještě bylo vhodné implementovat většinu vylepšení popsaných v kapitole 7.

Seznam použité literatury

Convention over Configuration [online]. Wikipedia, the free encyclopedia, Únor 2013. [cit. 25.2.2013]. Dostupné z: `http://en.wikipedia.org/wiki/Convention_over_configuration`.

Programming Language Popularity [online]. DedaSys LLC, 2013. [cit. 25.6.2012]. Dostupné z: `http://langpop.com/`.

RIEHLE, D. *Framework Design: A Role Modeling Approach*. PhD thesis, ETH Zürich, 2000. Dostupné z: `http://www.riehle.org/computer-science/research/dissertation/`. Dissertation No. 13509.

TRUJILLO, G. *MySQL versus Oracle Features/Functionality* [online]. 2008. [cit. 15.7.2012]. Dostupné z: `https://blogs.oracle.com/GeorgeTrujillo/entry/mysql_versus_oracle_features_functionality`.

Seznam tabulek

Sezam zdrojových kódů

Seznam obrázků

Seznam použitých zkratek

- **ANSI** — American National Standards Institute; americká standardizační organizace.
- **AJAX** — Asynchronous Javascript and XML; technologie pro komunikaci javascriptu ve webovém prohlížeči s webovým serverem bez nutnosti znovunačtení webové stránky v okně prohlížeče.
- **API** — Application Programming Interface (aplikační programové rozhraní); rozhraní, které aplikace, knihovna nebo framework poskytuje programátorům jiných aplikací.
- **DBAL** — Database Abstraction Layer; vrstva poskytující jednotné aplikační rozhraní pro různé databázové systémy.
- **CRUD** — Create, Read, Update, Delete; množina operací „vytvořit, přečíst, aktualizovat, smazat“ prováděná nad záznamy v databázi.
- **HTML** — HyperText Markup Language; značkovací jazyk sloužící ke strukturování hypertextových dokumentů.
- **MVC** — Model,View,Controller; návrhový vzor popisující třívrstvou architekturu (často webové) aplikace.
- **MVP** — Model,View,Presenter; návrhový vzor podobný MVC s drobnými odlišnostmi v pohledu na vrstvu aplikační logiky.
- **ORM** — Object-relational mapper; návrhový vzor pro ukládání objektů do relačních databází.
- **ODM** — Object-document mapper; návrhový vzor sloužící k ukládání objektů mapovaných na dokumenty; alternativa k ukládání do relační databáze.
- **PCRE** — Perl compatible regular expressions (regulární výrazy kompatibilní s jazykem Perl). PHP využívá dva různé dialekty zápisu regulárních výrazů - POSIX (názvy funkcí začínající `ereg`) a Perl (názvy funkcí začínající `preg`). Jelikož se varianta PCRE ukázala v praxi lepší, v PHP od verze 5.3.0 je POSIX varianta označená jako *deprecated*, tj. že by se dále neměla používat a v blízké budoucnosti může být z nových verzí PHP úplně odstraněna.
- **PHP** — PHP Hypertext Preprocessor, původně Personal Home Page; programovací jazyk pro webové aplikace běžící na straně serveru použitý pro vytvoření WheelDB Frameworku.
- **SQL** — Standard Query Language; dotazovací jazyk sloužící k práci se zejména relačními databázemi.
- **URL** — Unique Resource Locator; řetězec jednoznačně lokalizující zdroj nějakého obsahu (dokument, multimédia...) v internetu.

- **XML** — Extensible Markup Language; obecný značkovací jazyk sloužící pro zápis stromových datových struktur v plaintextu.

Slovníček pojmů

- *Autoloading* je mechanismus zajišťující automatické načtení třídy ve chvíli, kdy ji aplikace potřebuje.
- *Deprecated* (nejvhodnější český překlad je asi „zavržený“) se označují třídy, funkce, metody a další entity, které jsou pozůstatkem dřívějších verzí jazyka/knihoven. Jsou zachovány pouze z důvodu zpětné kompatibility a nedoporučuje se je nadále používat. Obvykle se entita takto označí poté, co je do jazyka/knihovny přidána jiná entita plnící stejnou funkci lépe (například univerzálněji, srozumitelněji).
- *Escapování* znamená nahrazení řídících znaků v řetězci speciální sekvencí, která je chápána jako původní tisknutelný znak bez řídícího významu. Escapovací sekvence se liší podle toho, jak se bude řetězec znaků interpretovat. Například řetězec `''a'' < ''b''` se pro zpracování HTML *escapuje* jako `&quot;a&quot; &lt; &quot;b&quot;`. do PHP řetězce se zapíše jako `''\''a \'' < \''b\''''`.
- *Konvence před konfigurací* (convention over configuration) je vývojový přístup blíže popsaný v kapitole 2.1.
- *Magické metody* (magic methods) jsou metody mající v daném programovacím jazyce speciální význam. V PHP jsou to například metody `__get()` a `__set()` definující chování při přístupu k atributu objektu, který není definován.
- *Méně kódu, více bezpečnosti* (less code, more security) je vývojový přístup blíže popsaný v kapitole 2.2.
- *Minifikace* zdrojového kódu je proces, kdy se různými technikami co nejvíce zkrátí délka zdrojového kódu pro nasazení do produkčního prostředí.
- *Obfuskace* je varianta minifikace, kde je kromě zkrácení kódu věnována snaha především jeho znepřehlednění, což jej částečně ochrání před dekompilací, pochopením a úpravami.
- *PhpDoc* je standard dokumentačních komentářů elementů programu (tříd, metod atd.) pro PHP vycházející z podobného standardu JavaDoc pro jazyk Java. Tyto komentáře jsou určeny pro strojové zpracování a umožňují zaprvé vývojovým prostředím poskytovat kontextovou nápovědu k elementům a zadruhé strojově vygenerovat dokumentaci v podobě reference aplikačního rozhraní (API) frameworku. *PhpDoc* komentáře začínají sekvencí `/**` a končí `*/`.
- *Presenter* je element prezentační vrstvy Nette frameworku. Více informací o presenterech lze nalézt na adrese `http://doc.nette.org/cs/presenters`
- *Reflection* je rozhraní nabízené některými programovacími jazyky, které umožňuje programu číst informace z vlastního zdrojového kódu - například komentáře, seznam metod a atributů třídy a další.

- *Rozhraní tříd, třídní rozhraní* (object interface) je v této práci označován mechanizmus rozhraní objektového programování, tj. definice typu obsahující deklarace objektového rozhraní, které poskytují instance třídy implementující toto rozhraní. Pojem *třídní rozhraní* používáme pro odlišení od obecného pojmu rozhraní, který může označovat mnoho jiných jevů, např. aplikační rozhraní.

- *SQL Pohled* je předpřipravený SELECT dotaz v databázi, k němuž lze přistupovat jako k tabulce pro čtení.

- *Syntaktický cukr* (syntactic sugar) je pomůcka umožňující zapsat přehledně a jednoduše úseky zdrojového kódu, jejichž zápis je v daném programovacím jazyce složitý nebo nepraktický, přičemž funkcionalita kódu zůstává stejná. Prostým příkladem syntaktického cukru je infixový zápis binárních aritmetických operátorů: $a + b$ místo $+(a, b)$.

- *Tovární metoda* (factory method) je metoda realizující návrhový vzor *Object Factory*, zjednodušeně řečeno statická metoda vytvářející nový objekt. Výhodou oproti klasickému konstruktoru je například to, že továrních metod může být více a tím se objekty mohou přehledněji vytvářet různými způsoby.

- *Záměna řízení aplikace* (inversion of control) je charakteristika odlišující framework od běžné knihovny popsaná v kapitole 2.3.

Přílohy

1. CD se zdrojovým kódem WheelDB frameworku, dokumentací API a ukázkovými aplikacemi