Univerzita Pardubice

Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2009 PIRKL LUKÁŠ

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Systém pro optimalizaci využití zásob

Lukáš Pirkl

Bakalářská práce

2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra informačních technologií
Akademický rok: **2008/2009**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Lukáš PIRKL**

Studijní program: **B2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **Systém pro optimalizaci využití zásob**

Zásady pro vypracování:

Tato práce si bere za úkol vytvoření systému, který by na základě známých zásob v kuchyni dokázal vybrat vhodný recept. Systém má za úkol vybrat ekonomicky (nákup co nejméně chybějících surovin), ovšem s přihlédnutím k požadavkům uživatele (typ pokrmu, vynechání některých ingrediencí apod.). Systém bude vytvořený jako internetová aplikace (PHP) a bude využívat databáze (My SQL).
Teoretická část: PHP; Návrh databázové struktury

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

GUTMANS, Andi, BAKKEN, Stig, RETHANS, Deric. Mistrovství v PHP 5. Bogdan Kiszka. Brno : CP Books, a.s., 2005. 656 s. ISBN 80-251-0799-X.

DUBOIS, Paul. MySQL profesionálně : komplexní průvodce použitím, programováním a správou MySQL. 1. vyd. Brno : Mobil Media, 2003. 1072 s. IDnes internet i knihy. ISBN 80-86593-41-X.

Vedoucí bakalářské práce: **Ing. Pavel Škrabánek**
Katedra řízení procesů

Datum zadání bakalářské práce: **15. ledna 2009**

Termín odevzdání bakalářské práce: **15. května 2009**

doc. Ing. Simeon Karamazov, Dr.
děkan

L.S.

Ing. Lukáš Čegan
vedoucí katedry

V Pardubicích dne 31. března 2009

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Ústí nad Orlicí dne 13. 4. 2009

Lukáš Pirkl

Anotace

V teoretické části je popsána historie a vlastnosti databázového systému MySQL včetně vysvětlení základních příkazů. Dále je popsán vznik skriptovacího jazyka PHP a jeho použití v rámci frameworku CakePHP. Tyto technologie jsou použity při realizaci praktické části, která má za cíl vytvořit systém pro optimalizaci využití zásob.

Klíčová slova

Optimalizace využití zásob, MySQL, PHP, CakePHP, MVC

Title

The system for optimizing the use of stock

Annotation

The theoretical part describes the history and features of MySQL database system, including an explanation of basic commands. It is also described the emergence of scripting language PHP and its usage in the CakePHP framework. These technologies are used in the implementation of the practical part, whose aim is created a system for optimizing the use of stock.

Keywords

Optimizing the use of stock, MySQL, PHP, CakePHP, MVC

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

ACL	Access Control List
AJAX	Asynchronous JavaScript and XML
BDB	Berkeley Database
CSS	Cascade Style Sheets
CSV	Comma-separated values
E-R diagram	Entity Relationship Diagram
HTML	Hypertext Markup Language
ID	Identification
ISAM	Indexed Sequential Access Method
MVC	Model View Controller
PHP	PHP: Hypertext Preprocessor
RDBMS	Relational Database Management System
SQL	Structured Query Language
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
XML	eXtensible Markup Language

1 Úvod

V dnešní době o úspěchu firmy na trhu rozhoduje řada více či méně důležitých aspektů. Jedním z nich je i schopnost firmy optimálně využívat své skladovací kapacity a suroviny. Toto bylo impulzem k zadání bakalářské práce, zaměřené na vytvoření systému, který by dokázal optimálně spravovat zásoby. Požadavkem při jeho tvorbě bylo využití volně dostupných prostředků. Jako nejvhodnější se jeví databázový systém MySQL a skriptovací jazyk PHP.

Jelikož návrh systému pro správu skladu výrobního podniku je značně závislý na velikosti firmy a jejím zaměření a rovněž by byl značně komplikovaný, je výstupem této práce systém, který podle požadavků uživatele a aktuálního stavu zásob zvolí nejvhodnější recept pro přípravu jídla. Tato aplikace dostatečně demonstruje možnosti PHP a MySQL při tvorbě podobných systému pro potřeby firem.

V první části této práce je popsán systém pro správu relační databáze MySQL, jeho historie a vlastnosti. Dále jsou rozebrány důležité příkazy pro tvorbu a úpravu databázových tabulek, indexů, pohledů a také základní SQL dotazy pro získávání dat. Rovněž je zmíněno, jak navrhovat optimální databázové struktury a SQL dotazy.

Další část je věnována skriptovacímu jazyku PHP, jeho historii a vlastnostem. Protože je PHP svou syntaxí velice podobné jazyku C, základní příkazy a programové konstrukce popsány nejsou. Místo toho je prostor věnován frameworku CakePHP. Je vysvětlen princip návrhového vzoru MVC a jeho implementace ve frameworku. Také je vysvětlena konvence pro tvorbu názvů jednotlivých částí aplikace v něm tvořené.

V poslední části je popsána samotná aplikace tvořící praktickou část této práce. Jejím cílem je navrhnout systém pro optimální využití zásob v kuchyni. Tedy podle aktuálního stavu surovin vybrat takový recept, který po uvaření způsobí, že zásoby na skladě budou stále optimální.

V závěru se nachází celkové hodnocení a shrnutí poznatků z vývoje aplikace. Rovněž nechybí návrh, jak se může aplikace v budoucnu zdokonalit.

2 MySQL

2.1 RDBMS

MySQL je systém pro správu relační databáze (RDBMS). Znamená to, že data jsou ukládány ve formě tabulek. Sloupce tabulky mohou být různých datových typů a mohou mít různé podmínky pro povolená data (např. pouze celá čísla od 1 do 10). Sloupce mohou mít také speciální vlastnosti, jakými jsou primární a cizí klíče. Díky těmto klíčům lze definovat vztahy (relace) mezi tabulkami a tím vytvořit relační databázi v pravém slova smyslu. MySQL je systém typu klient/server. Pro práci s daty klient odesílá serveru SQL dotazy a server mu odpovídá.

2.2 Historie

MySQL byl vyvinut v roce 1994, když firma TcX hledala databázoví server pro webové aplikace a všechny dostupné komerční řešení se pro jejich rozsáhlé tabulky zdály příliš pomalé. Vyšli ze systému mSQL, pro který existovalo velké množství volně šiřitelných nástrojů a vytvořili MySQL, které mělo programovací rozhraní velice podobné. Nebylo tedy potřeba vyvíjet další nástroje pro správu, nový systém byl kompatibilní s mSQL.

V roce 1996 byl MySQL puštěn do světa jako binární distribuce pro Linux a Solaris. Za účelem distribuce a pořádání různých školení byla založena firma MySQL AB. Systém byl uvolněn pro mnoho dalších platforem a rovněž byly uvolněny zdrojové kódy. MySQL je jako jeden z mála databázových systémů Open Source projekt.

Počátkem roku 2008 proběhla akvizice[1] MySQL AB společností Sun Microsystems za přibližně 1 miliardu dolarů. O rok později byla oznámena akvizice Sun Microsystems firmou Oracle.

[1] Akvizice je proces, při němž dochází k právnímu nebo ekonomickému spojování podniků.

2.3 Výhody

- Rychlost – podle tvrzení vývojářů, nejrychlejší dostupný RDBMS
- Snadné používání – jednodušší správa oproti složitějším systémům
- Podpora dotazovacího jazyka – použití SQL stejně jako většina jiných systémů
- Univerzálnost – k databázi se může současně připojit více různých klientů (nástroje pro správu, webové aplikace, …)
- Zabezpečení – umožňuje řízení přístupu a šifrování pomocí protokolu SSL
- Rozšířenost – dostupný prakticky u všech poskytovatelů webhostingu
- Přenositelnost – verze pro různé varianty Unixu, Windows, OS/2
- Malá velikost – desítky MB (samotný MySQL)
- Dostupnost a cena – pro nekomerční účely zdarma

2.4 Nevýhody

- Horší výkon při zátěžových aplikacích
- U nejrozšířenějšího zpracovatele tabulek MyISAM není podpora transakcí a cizích klíčů
- Nepodporuje složitější programátorské konstrukce

2.5 Zpracovatelé tabulek

MySQL obsahuje více zpracovatelů tabulek (storage engines). Každý má vlastní typ tabulek se specifickými charakteristikami a vlastnostmi, a proto se každý hodí pro jiné aplikace. V konfiguraci MySQL lze určité typy zakázat anebo povolit. Takže se může stát, že na levném webhostingu bude k dispozici pouze výchozí typ tabulek (MyISAM) a nic víc.

2.5.1 ISAM

ISAM (Indexed Sequential Access Method) byl prvním vyvinutým typem a jediným dostupným ve verzích MySQL starších než 3.23. Byl překonán typem MyISAM, který je méně omezující a ihned se stal typem výchozím. Od verze 5.0 již typ ISAM není podporovaný.

2.5.2 MyISAM

MyISAM se objevil jako nástupce typu ISAM ve verzi 3.23 a pokud server není nastaven jinak, tak je typem výchozím. Umožňuje ukládat tabulky velkých velikostí, omezených pouze maximální podporovanou velikostí souboru operačního systému. Tabulky jsou ukládány ve formátu nezávislém na stroji, takže jsou přenositelné mezi různými platformami. MyISAM poskytuje největší podporu pro *AUTO_INCREMENT* na rozdíl od ostatních typů tabulek. Tabulky mají indikátor správného uzavření. Pokud je server nekorektně ukončen, při jeho dalším spuštění lze snadno najít tabulky, u kterých je šance, že byly poškozeny a ještě během startu se je pokusit opravit. Největší výhodou tohoto typu je poskytování *FULLTEXT* indexu, pomocí kterého lze provádět fulltextové vyhledávání.

2.5.3 Merge

Tabulka typu merge je v podstatě jenom spojení více stejných tabulek MyISAM. Tímto způsobem se dá obejít omezení maximální velikosti tabulky. Fyzicky je tabulka uložena ve více souborech, a proto může být ve výsledku větší než je maximální velikost souboru v operačním systému. Tabulky, které tvoří tabulku merge musí mít stejnou strukturu (stejné názvy, datové typy i pořadí sloupců).

2.5.4 HEAP

Tabulky HEAP jsou velice rychlé, protože jsou vytvořeny pouze v paměti. Navíc všechny jejich řádky mají pevnou délku, což ještě více urychluje prohledávání. Z těchto faktů vyplívají omezení. Tabulky jsou pouze dočasné. Po vypnutí serveru zmizí. Nelze v nich používat datové typy *TEXT* a *BLOB*, protože nemají přesně definovanou délku. I typ *VARCHAR* se interně považuje za *CHAR* s pevnou délkou.

2.5.5 BDB

Tento typ tabulky je zpracován zpracovatelem Berkeley DB, který vyvinula Sleepycat. Podporuje transakce s akcemi *commit* a *rollback*, data se automaticky po havárii sama obnovují a umožňuje zamykání na úrovni stránek.

2.5.6 InnoDB

Tento typ tabulky je zpracován zpracovatelem InnoDB, který vyvinula Innobase Oy. Podporuje transakce s akcemi *commit* a *rollback*, data se automaticky po havárii sama obnovují a umožňuje zamykání na úrovni řádků. Dále umožňuje správu nejen primárních, ale navíc také cizích klíčů, včetně kaskádového vymazávání. InnoDB

umožňuje obejití maximální velikosti souboru v operačním systému tím, že soubor s daty rozdělí do více menších podobně jako typ Merge. Děje se to ovšem automaticky.

2.5.7 Archive

Typ Archive vznikl z důvodu stále rostoucího objemu uchovávaných historických dat různých organizací. Archive je stejný typ jako MyISAM s tím, že všechna data komprimuje pro jejich efektivnější skladování. K archivním datům se přistupuje jenom zřídka, a proto nepodporuje indexy. Bez indexů se navíc ušetří další datový prostor.

2.5.8 CVS

Tento typ používá pro ukládání dat textové soubory, ve kterých jsou data oddělena čárkou. Využití je pro snadnou editaci tabulek ve chvílích, kdy je databáze offline a snadný import a export dat do tabulkových procesorů a editorů. Používají se pro aplikace, které potřebují hlavně ukládat velké množství dat.

2.6 Vztahy mezi tabulkami

Pokud spolu nějaké dvě tabulky souvisí, je mezi nimi vytvořen vztah (relace).

2.6.1 1:1

Jeden záznam v první tabulce souvisí s právě jedním záznamem v tabulce druhé. Tento vztah se v praxi moc nepoužívá, protože většinou se takovéto záznamy umístí do jedné tabulky rovnou.

2.6.2 1:N

S jedním záznamem v první tabulce souvisí více záznamů z tabulky druhé. Jedná se o nejčastější vztah. Například jedna surovina má více jednotek a jedna jednotka je ve více receptech.

2.6.3 M:N

S mnoha záznamy v první tabulce souvisí mnoho záznamů z tabulky druhé. V praxi se realizuje pomocí propojovací tabulky, takže se jedná o dva vztahy, jeden typu 1:N a druhý typu 1:M. Například jedna charakteristika může být přidělena u mnoha receptů a jeden recept zase může mít více charakteristik.

2.7 Základní dotazy pro práci s MySQL

2.7.1 Práce s databází

V jednom systému MySQL může existovat mnoho různých databází. Lze tak snadno oddělit data patřící rozdílným aplikacím, případně uživatelům. Nová databáze se vytvoří jednoduchým příkazem:

```
CREATE DATABASE bakalarska_prace;
```

Abychom mohli zadávat další dotazy snadněji a nemuseli před jménem každé tabulky uvádět, do které patří databáze, lze nastavit vytvořenou databázi jako výchozí.

```
USE bakalarska_prace;
```

Pokud už databázi nepotřebujeme, lze ji smazat příkazem:

```
DROP bakalarska_prace;
```

2.7.2 Práce s tabulkou

Abychom mohli data kam ukládat, je potřeba do databáze vytvořit tabulku.

```
CREATE TABLE recepty (
  id                INT UNSIGNED NOT NULL AUTO_INCREMENT,
  nazev             VARCHAR(255) NOT NULL,
  postup            TEXT,
  casova_narocnost  INT NOT NULL,
  PRIMARY KEY (id)
);
```

Dotaz vytvoří tabulku s názvem *recepty*, která bude obsahovat čtyři sloupce. Sloupec s názvem *id* bude datového typu *INT* (celá čísla), bez záporných čísel, nesmí obsahovat hodnotu *NULL* a automaticky bude svou hodnotu s každým dalším záznamem zvyšovat, pokud mu nepřiřadíme jinou. Sloupec se jménem *nazev* bude datového typu *VARCHAR* (řetězec znaků) s maximální délkou 255 a nesmí obsahovat hodnotu *NULL*. Sloupec *postup* je datového typu *TEXT* (posloupnost znaků libovolné délky). Poslední sloupec *casova_narocnost* je typu *INT* a nesmí být *NULL*. Poslední řádek s klíčovým slovem *PRIMARY KEY* určuje primární klíč tabulky. Podle takto označeného sloupce (může jich být i více) se jednoznačně určí právě jeden řádek tabulky. Automaticky je na takovýto sloupec aplikována podmínka, že data v něm musí být jedinečná.

K již vytvořené tabulce lze přidat další sloupec (např. s názvem *pocet*) příkazem:

```
ALTER TABLE recepty ADD COLUMN pocet INT;
```

Pomocí *ALTER TABLE* lze také upravovat existující sloupec a to nejenom jeho datový typ, ale i jeho název. Následující příklad přejmenuje sloupec *pocet* na *jmeno* a jeho typ změní na *INT*:

```
ALTER TABLE recepty CHANGE COLUMN pocet jmeno TEXT;
```

Nechtěně přidané nebo vytvořené sloupce lze rovněž z tabulky odstranit:

```
ALTER TABLE recepty DELETE COLUMN jmeno;
```

Pokud bychom uznali za vhodnější celou tabulku odstranit a vytvořit novou, použijeme příkaz:

```
DROP TABLE recepty;
```

Může ale nastat situace, že nechceme smazat celou tabulku, ale hodilo by se ji pouze vyprázdnit. V takovéto situaci se hodí příkaz:

```
TRUNCATE TABLE recepty;
```

MySQL ve skutečnosti tabulku smaže a podle její definice ji znovu vytvoří. Z toho vyplívá skutečnost, že hodnoty generované pomocí *AUTO_INCREMENT* se začnou počítat znovu od začátku.

2.7.3 AUTO_INCREMENT

Prakticky pro každou vytvářenou tabulku se hodí mít jednoznačný primární klíč pro určení záznamu. V tabulce uchovávající informace o uživatelích systému by jím mohlo být například jejich uživatelské jméno. V tabulce se seznamem charakteristik by se dal použít název této charakteristiky. Jenomže může nastat situace, kdy bude potřeba upravit právě tento název. Pak by se musel tento název upravit nejen v tabulce, kde je uveden jako primární klíč, ale také ve všech ostatních tabulkách, které jsou s touto v nějakém vztahu (název je pro ně cizím klíčem). V těchto případech je snazší mít jako identifikátor nějakou jinou informaci, která se určitě nebude nikdy měnit.

Nejčastěji se pro identifikátor (ID) používá kladné celé číslo. MySQL nám jeho generování ulehčuje právě atributem *AUTO_INCREMENT*. Při přidávání nového záznamu stačí, aby se hodnota pro sloupec s tímto atributem neuvedla, a server sám

dosadí další hodnotu. V případě, že je hodnota ID uvedena a ještě se ve sloupci nenachází, použije se. Pokud je tato vložená hodnota ve sloupci největší, generování posloupnosti pokračuje právě od ní. Tím v posloupnosti vznikne díra. Pokud potřebujeme v dalším dotazu zjistit, jaké číslo bylo vygenerováno, lze použít funkci *LAST_INSERT_ID()*.

Každý typ tabulky přistupuje k *AUTO_INCREMENT* trochu jinak. MyISAM přišla oproti starému ISAM s řadou úprav. U typu ISAM se posloupnost generovala o jedna větší než je největší hodnota v tabulce. Takže pokud se smazal řádek s nejvyšší hodnotou, další vložený záznam dostal stejnou hodnotu, jako měl ten minulý. To je při generování například členských čísel nevhodné. Stejně tak u ISAM nešlo nastavit počáteční hodnotu generování a všechny museli začínat jedničkou. U MyISAM je generovaná posloupnost ryze rostoucí a hodnoty se opětovně nepoužívají. Při vytváření tabulky lze dokonce určit, kterou hodnotou se má posloupnost začít.

Od verze MySQL 3.23.5 MyISAM navíc podporuje složené primární klíče. Takže můžeme v jedné tabulce vytvořit více na sobě nezávislých posloupností. Takovou tabulku vytvoříme následujícím příkazem:

```
CREATE TABLE bugy (
  projekt  varchar(20) NOT NULL,
  bug_id   INT UNSIGNED AUTO_INCREMENT NOT NULL,
  popis    TEXT,
  PRIMARY KEY (projekt, bug_id)
) TYPE = MyISAM;
```

Do tabulky přidáme data:

```
INSERT INTO bugy SET projekt="prvni", popis="a";
INSERT INTO bugy SET projekt="druhy", popis="b";
INSERT INTO bugy SET projekt="druhy", popis="c";
INSERT INTO bugy SET projekt="prvni", popis="d";
INSERT INTO bugy SET projekt="druhy", popis="e";
```

Sloupec *bug_id* se vygeneruje automaticky pomocí atributu *AUTO_INCREMENT*. Výsledek pak vypadá následovně:

```
---------------------------
| projekt | bug_id | popis |
---------------------------
| prvni   |      1 |     a |
| prvni   |      2 |     d |
| druhy   |      1 |     b |
| druhy   |      2 |     c |
| druhy   |      3 |     e |
---------------------------
```

Na rozdíl od standardního chování *AUTO_INCREMENT* u tabulek MyISAM se při smazání nejvyššího ID použije při dalším vkládání znovu.

Pro typ tabulky HEAP lze *AUTO_INCREMENT* použít až od verze MySQL 4.1. Počáteční hodnotu posloupnosti lze při vytváření tabulky předem nastavit, odstraněná nejvyšší hodnota ID se znovu nepoužije a složený primární klíč pro používání více nezávislých posloupností použít nelze. U tabulek typu BDB nelze určit začátek posloupnosti, odstraněná nejvyšší ID se používají znovu a složený primární klíč lze použít pro více nezávislých posloupností. U tabulek typu InnoDB nelze nastavit začátek posloupnosti, odstraněná nejvyšší ID se znovu nepoužijí a složený primární klíč pro vytvoření více posloupností nelze.

2.7.4 Zobrazování dat

Pro zobrazení dat slouží příkaz začínající slovem *SELECT*. Pro zobrazení celé tabulky *suroviny* se použije následující příkaz:

```
SELECT * FROM suroviny;
```

Za slovem *SELECT* následuje seznam sloupců tabulky, které se mají zobrazit. V tomto případě je místo něho hvězdička, která znamená, že se zobrazí všechny sloupce. Za slovem *FROM* následuje seznam tabulek, ze kterých se budou zobrazovat sloupce.

Pokud by nás zajímala pouze jedna konkrétní surovina s id např. 1 a u této suroviny pouze její název, příkaz upravíme do této podoby:

```
SELECT nazev FROM suroviny WHERE id=1;
```

V příkazu se nyní objevilo slovo *WHERE*, za kterým následují podmínky. Podmínek může být více a mohou se spojovat logickými operátory *AND* nebo *OR*.

Lze provádět také složitější operace. Například spočítat kolik surovin má na skladě stejné zásoby.

```
SELECT count(skladem_gramu) FROM suroviny
GROUP BY skladem_gramu;
```

GROUP BY sloučí všechny řádky, které obsahují ve sloupci *skladem_gramu* stejné hodnoty. Funkce *count()* pak dokáže spočítat, kolik řádků bylo takto sloučeno. Exis-

tuje mnoho další funkcí, které pracují se sloučenými řádky. Např. *sum()* spočítá sumu čísel, *avg()* zase jejich průměr.

Na sloučené řádky lze aplikovat podmínky stejně snadno jako na normální řádky slovem *WHERE*.

```
SELECT count(skladem_gramu) as pocet FROM suroviny
GROUP BY skladem_gramu
HAVING pocet > 1
```

Příkaz zobrazí pouze řádky, které vyšly větší než jedna. Pro tyto sekundární podmínky se používá slovo *HAVING*. Abychom však v této podmínce mohli s nově vzniklým sloupcem pracovat, je pro něj vytvořen alias *pocet*. Tento alias se bohužel nedá využít v primárních podmínkách za slovem *WHERE,* protože v době jejich aplikace, o něm MySQL ještě neví.

Dále by bylo dobré vědět, jaké má tato konkrétní surovina jednotky. Ty jsou ovšem uloženy v jiné tabulce, protože jedna surovina může mít více jednotek.

```
SELECT suroviny.nazev, jednotky.nazev
FROM suroviny, jednotky
WHERE suroviny.id=jednotky.surovina_id;
```

Když data zobrazujeme z více tabulek, je potřeba uvádět před názvem sloupců, ze které tabulky jsou. Pokud by byly názvy v každé tabulce různé, server by si správnou hodnotu našel, ale v případě sloupce *nazev* by nepoznal, který název máme na mysli.

V případě že budeme mít surovinu, která nemá žádnou jednotku, a my ji přesto budeme chtít zobrazit, musíme použít pro spojení slovo *JOIN*.

```
SELECT suroviny.nazev, jednotky.nazev
FROM suroviny LEFT JOIN jednotky
ON suroviny.id=jednotky.surovina_id;
```

Před slovem *JOIN* je ještě slovo *LEFT*, které upřesňuje, na kterou stranu se má spojovat. Tento příkaz vypíše všechny suroviny a u těch, které nemají žádnou jednotku, místo ní vypíše *NULL*. Když bychom místo *LEFT* napsali *RIGHT*, vypíší se stejným způsobem naopak jednotky, které nemají žádnou surovinu. Neuvedeme-li ani jedno, výsledek bude stejný jako v předchozím případě.

2.7.5 Pohledy

Pokud napíšeme nějaký komplikovaný příkaz *SELECT,* který budeme často používat a využít ho při psaní dalších dotazů, lze z něho vytvořit pohled.

```
CREATE VIEW suroviny_jednotky AS (
SELECT suroviny.nazev as suroviny,
       jednotky.nazev as jednotky
FROM suroviny LEFT JOIN jednotky
ON suroviny.id=jednotky.surovina_id
);
```

Pohled je vlastně náhled na data, se kterým se dá pracovat jako s další tabulkou. Takže pro jeho zobrazení stačí napsat:

```
SELECT * FROM suroviny_jednotky;
```

Díky pohledům mohou být velice složité dotazy přehlednější, ovšem všeho moc škodí. Pokud bude v jedné databázi vytvořeno hodně pohledů, kde navíc jeden bude zobrazovat data z druhého, nakonec ani nemusíme poznat, ze které tabulky se vlastně data berou a vše bude daleko nepřehlednější než bez pohledů.

Pro odstranění nepotřebného pohledu lze použít příkaz:

```
DROP VIEW suroviny_jednotky;
```

2.7.6 Indexy

Index urychluje procházení a vyhledávání v tabulce. Je to pomocná struktura, která obsahuje setříděná data z indexovaného sloupce tabulky. Během vyhledávání například všech řádků obsahujících číslo 5, se použije index, umožňující použít např. binární vyhledávání. Bez indexu by se musela tabulka procházet celá, záznam po záznamu a to by trvalo o poznání déle.

MySQL obsahuje čtyři druhy indexů:

1. Klasický index – bez žádných omezení, použitelný pro cokoli
2. Jedinečný index – navíc hlídá, aby data byla jedinečná a v případě vícesloupcového indexu, aby byla jedinečná kombinace hodnot ve sloupcích
3. Primární klíč – stejný jako jedinečný, ale navíc hlídá, že hodnota není neznámá (*NULL*) a může být použitý v tabulce jenom jednou
4. Fulltextový index – umožňuje textová data prohledávat fulltextově

Indexy lze přidávat k existujícím tabulkám. Například chceme přidat index ke sloupci *nazev* tabulky *recepty*:

```
ALTER TABLE recepty ADD INDEX(nazev);
```

V případě, že bychom považovali za vhodnější, aby název byl jedinečný pro celou tabulku, použijeme jedinečný index.

```
ALTER TABLE recepty ADD UNIQUE(nazev);
```

Postup receptu by se dal použít pro fulltextové vyhledávání, které ovšem musí mít nejprve vytvořený potřebný index.

```
ALTER TABLE recepty ADD FULLTEXT(postup);
```

Stejně tak lze vytvořit i index pro primární klíč slovem *PRIMARY KEY*.

Indexy lze rovněž z tabulky odstranit, aniž by se jakkoliv poškodila uložená data. Pro všechny typy indexů je příkaz stejný.

```
ALTER TABLE recepty DROP INDEX postup;
```

2.8 Optimalizace databáze

Pro co nejrychlejší běh aplikací vyžívajících databázi je potřeba, aby i databáze samotná odpovídala na dotazy co nejrychleji. Obzvláště při velkém objemu dat se projeví i ty nejdrobnější chyby v návrhu tabulek, dotazů, pohledů a indexů.

2.8.1 Návrh tabulky

I když se tabulky dají snadno upravovat i za běhu, jsou kvůli jejich provázanosti s další aplikací větší úpravy často velice obtížné a pracné. Proto je potřeba už při jejich návrhu myslet na to, aby byly co nejoptimálněji navržené.

Nejprve je potřeba vybrat vhodný typ tabulky. Poté podle jeho vlastností zvolit správné datové typy pro sloupce. U tabulek typu ISAM a MyISAM je lepší volit typy s pevně danou velikostí. Lépe s nimi pracuje a nevzniká fragmentace. Takže je nejlepší pro všechny textové řetězce používat datový typ *CHAR*, což ale není možné ve všech případech. Typ tabulek InnoDB uchovává záznamy jiným způsobem, a proto je mu jedno, jestli jsou záznamy stejně dlouhé nebo ne. Zde se naopak vyplatí používat datové typy s proměnlivou délkou. Zaberou totiž pouze tolik místa, kolik je do nich opravdu uloženo. Tím je pak soubor s daty menší, ušetří se kapacita disku a sníží se počet vstupních a výstupních operací pro práci s diskem.

Pro co nejmenší velikost datového souboru je také vhodné používat co nejmenší datové typy. Pokud nám vystačí na uložení čísla typ *MEDIUMINT* nebudeme zbytečně používat *BIGINT*. Index pro sloupec s menším rozsahem navíc pracuje rychleji a stejně jako tabulka samotná i on zabírá méně místa na disku.

Pro určení optimálního datového typu existuje v MySQL od verze 3.23 příkaz *PROCEDURE ANALYSE()*, který dokáže podle dat v tabulce určit vhodný datový typ.

```
SELECT * FROM recepty PROCEDURE ANALYSE();
```

Načítání velkého objemu dat trvá vždy dlouho, proto je velmi výhodné dát objemný sloupec do zvláštní tabulky a přistupovat k němu až v případě, že data opravdu potřebujeme. Například když budeme do databáze ukládat fotografie, budou v datovém typu *BLOB*. Každá fotografie může být velká i několik megabytů. Při každém dotazu *SELECT* * se musí tato velká data načíst. Proto je efektivnější místo hvězdičky psát konkrétní název sloupce. Občas může být výhodnější tento sloupec s fotografiemi odsunout do zvláštní tabulky, například když nám to umožní vytvořit záznamy s pevně danou šířkou řádku. Zamezíme tak fragmentaci. Ale i pokud není možné vytvořit pevně danou šířku řádku, aspoň se nám nebude fragmentovat hlavní tabulka, nad kterou probíhá většina dotazů.

2.8.2 Použití indexů

Indexy zrychlují prohledávání, ale rovněž zpomalují operace vkládání a mazání dat a také úpravy indexovaných hodnot. Pokaždé se totiž musí aktualizovat také index. Ten navíc musí být někde uložen a může se stát, že soubor s indexy dosáhne maximální velikosti souboru podporovaného operačním systémem dříve, než datový soubor tabulky. Proto je potřeba indexy používat s rozvahou a jenom tam, kde nám přinesou užitek.

Pokud máme nějaký sloupec již indexovaný, je potřeba ho efektivně používat. Například následující podmínka *WHERE* index použít nemůže.

```
WHERE sloupec * 2 < 4
```

Každý sloupec se musí vynásobit dvěma a pak se porovnává, zda je menší než čtyři. Stejnou podmínku jde ale napsat i jinak.

```
WHERE sloupec < 4/2
```

Nejprve se spočítá, že výraz 4 / 2 je roven 2 a dále se pro sloupec může použít index a porovná se, zda je menší než 2.

2.8.3 Defragmentace

U některých tabulek vzniká při častém používání příkazů *INSERT* a *DELETE* fragmentace. Tu lze odstranit příkazem:

```
OPTIMIZE TABLE recepty;
```

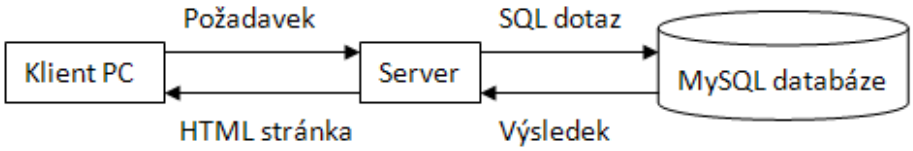
Tento příkaz funguje pouze na tabulky typu MyISAM. U ostatních lze stejného efektu dosáhnout uložením tabulky utilitou *mysqldump*[2], smazáním tabulky z databáze a opětovného vytvoření tabulky z výstupního souboru této utility.

[2] *mysqldump* vytváří soubor s SQL příkazy, které vedou k opětovnému vytvoření tabulky, nebo celé databáze

3 PHP

3.1 Co je PHP

PHP je skriptovací jazyk, který slouží převážně pro generování HTML stránek. Je to serverový jazyk. Znamená to, že všechny potřebné operace vykonává webový server a výsledek odesílá klientovi jako HTML stránku.



Obrázek 1. Komunikace klient/server

Výhodou tohoto řešení je absolutní nezávislost na webovém prohlížeči a platformě klienta. Ovšem nevýhodou je, že po každém požadavku se musí znovu generovat celá stránka. Tato nevýhoda se ale dá obejít pomocí HTML rámců, anebo moderněji, pomocí AJAX[3].

3.2 Historie

V roce 1994 chtěl pan Rasmus Lerdorf zjistit informace o přístupech na své webové stránky. Vytvořil tedy v jazyce Perl sadu skriptů, které tuto činnost zajišťovali. Protože Perl velice zatěžoval server, byl celý projekt přepsán do jazyka C. Mezi vývojáři projekt vzbudil zájem, a proto byl zveřejněn pod názvem Personal Home Page Tools a později Personal Home Page Construction Kit, zkráceně PHP.

V polovině roku 1995 byl PHP rozšířen o podporu SQL dotazů a tedy o propojitelnost s databázovým serverem. Tato novější verze nesla označení PHP/FI 2.0 a stala se oblíbenou po celém světě.

Protože PHP bylo šířeno jako Open Source, mohlo být v roce 1997 přepsáno jinými vývojáři do objektově orientované podoby a označeno jako PHP 3.0. Tím se usnadnil další vývoj a ostatní programátoři mohly vyvíjet pro PHP rozšiřující moduly a přidávat tak podporu dalších specializovanějších funkcí.

[3] *AJAX* je kombinace technologií (JavaScript, DOM, XML, XMLHttpRequest, HTML, CSS), které umožňují podle odezvy serveru aktualizovat pouze část stránky. Lze tak vytvářet uživatelsky přívětivé webové aplikace.

V květnu 2000 byla uvolněna verze PHP 4.0, která měla nové jádro, pojmenované Zend Engine. Ten zrychlil interpretaci PHP skriptů a zjednodušil práci s rozšiřujícími moduly.

PHP 5.0 přišlo na svět v červenci 2004. Jako hlavní novinky byly nový Zend Engine II a hlavně podpora objektově orientovaného programování ve skriptech. Přišel rovněž s vylepšením v podobě lepší práce s databází pomocí objektů a lepší prací s XML soubory pomocí objektu SimpleXML.

3.3 Výhody

- Velice podobná syntaxe jako C
- Open source
- Široká podpora komunity
- Multiplatformní
- Hodně rozšířený
- Podpora široké řady jiných standardů, formátů a technologií

3.4 Nevýhody

- Jazyk je interpretovaný a ne kompilovaný (což může být i výhoda)
- Každý, kdo má přístup k souborům uloženým na serveru, může vidět zdrojový kód aplikace
- Některé funkce mohou v dalších verzích fungovat jinak anebo vůbec

3.5 Framework

Framework je softwarová struktura, která usnadňuje práci vývojáři. Může mu poskytovat podpůrné programy, návrhové vzory, knihovny nebo doporučované postupy při vývoji. Řeší typické a neustále se opakující problémy jednodušeji a tím šetří čas a zefektivňuje vývoj aplikací. Například při vytváření webového systému pro optimalizaci zásob se stačí soustředit na návrh databáze a řešení problémů spojených s efektivním využitím skladu. To, jak zpracovat HTML formuláře, ukládat data do databáze nebo vypisovat chybové hlášky při vložení špatných údajů, zajistí framework.

Pro stránku, na které je pouze jedna kniha návštěv a jinak vše ostatní jsou pouze statické HTML soubory, se samozřejmě použití frameworku nevyplatí. Jednak by s jeho použitím bylo víc práce než naprogramovat takovouto jednoduchou věc přímo, ale

také by to bylo neefektivní z hlediska výkonnosti. Framework je jako jakákoli jiná aplikace také kus kódu, který se musí zpracovávat a každé zpracování trvá určitý čas. Navíc pochopení jeho principů a naučení se s ním efektivně pracovat zabere čas, za který bychom měli knihu návštěv dávno hotovou. Jeho použití je proto vhodné pouze pro vývojáře, kteří často vytvářejí složitější aplikace. Odpadne jim neustálé vytváření stejných nezajímavých a nekreativních kusů kódu a mohou se o to více věnovat tomu, co má zadaný úkol primárně splňovat.

3.6 CakePHP

CakePHP je open source framework napsaný ve skriptovacím jazyku PHP a určený rovněž pro PHP. Je vyvíjen od roku 2005 a má širokou uživatelskou základu, přehlednou dokumentaci a okolo sebe velkou komunitu lidí, která se stará o jeho vývoj a poskytuje technickou podporu prostřednictvím různých diskusních skupin.

3.6.1 Požadavky

- http server (je preferován apache s modulem mod_rewrite)
- PHP 4.3.2 a jakékoliv novější (preferováno je PHP5 a novější)
- Databáze není požadována, ale většina aplikaci ji z principu vyžaduje. Jsou podporovány databáze: MySQL, PostrgeSQL, Firebird DB2, Microsoft SQL Server, Oracle, SQLite

3.6.2 Vlastnosti

- Je kompatibilní s PHP4 i PHP5
- Integruje funkce pro snadné vytváření, čtení, zapisování a mazání databáze
- Obsahuje „dispečera“ pro snadnou správu URL adres jednotlivých částí aplikace
- Šablony pro vzhled jednotlivých stránek jsou standardní HTML soubory s možností používání klasických PHP příkazů
- Obsahuje „pomocníky“ pro vytváření HTML elementů a formulářů, JavaScriptu nebo AJAXu
- Je nezávislý na adresáři. Hotový projekt lze jednoduše přemístit do jiného adresáře a okamžitě všechny odkazy fungují, jak mají.
- Má zabudovanou validaci vstupních dat.
- Umožňuje používat pro správu uživatelů a omezení jejich přístupu k jednotlivým částem aplikace seznam pro řízení přístupu (Access Control List)

- Pro naplnění databáze daty umožňuje jednoduše vygenerovat stránky pouze podle struktury databáze. (Scaffolding)
- Obsahuje komponenty pro správu sessions, zajištění autorizace a autentizace uživatelů a mnoho dalších
- Používá návrhový vzor MVC (Model-View-Controller)
- Poskytuje skripty pro konzoly, které usnadňují práci např. automatickým generováním aplikace podle šablony a navržené databáze.
- Preferuje zásady před konfigurací. To znamená, že pokud správně pojmenujeme tabulky v databázi, model nemusíme prakticky vůbec konfigurovat a všechno bude fungovat.

3.6.3 Návrhový vzor MVC

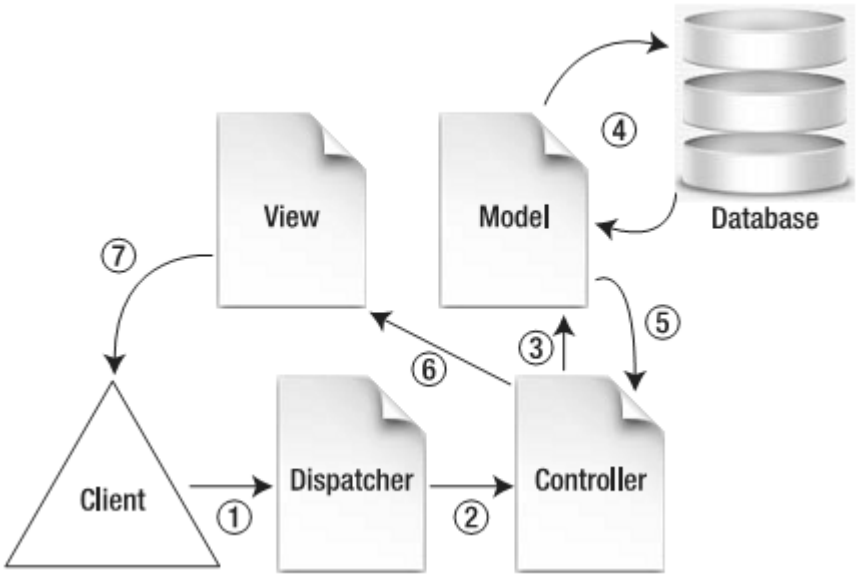
Návrhový vzor je návrh na obecné řešení nějakého problému. Příklad návrhového vzoru může být například architektura. Gotické stavby jsou si všechny vesměs podobné, protože vycházejí ze stejného vzoru. Ve světě objektově orientovaného programování vzory většinou říkají, jaké budou vztahy mezi objekty a třídami, ale žádnou konkrétní třídu neimplementují.

Vzor MVC rozděluje aplikační logiku do tří vrstev.

- Model (model) – zajišťuje komunikaci s databází a práci s daty na nejnižší úrovni
- View (pohled) – zobrazuje data z modelu uživateli
- Controller (řadič) – zpracovává požadavky od uživatele a mění podle nich model nebo pohled

Toto rozdělení zajišťuje flexibilitu aplikace při jejím dalším vývoji. Například při migraci na jiný databázový systém stačí pouze změnit model.

Komunikace mezi vrstvami probíhá podle následujícího schématu.



Obrázek 2. Schéma návrhového vzoru MVC v CakePHP

1. Uživatel klikne na odkaz, který vede do jiné části aplikace. Nebo zadá URL adresu podle ve tvaru:

```
http://www.doména.cz/aplikace/řadič/akce/další parametry
```

2. Dispečer podle URL adresy pozná, který řadič je potřeba zavolat, která akce řadiče se bude provádět a jaké ji předat další parametry.
3. Řadič většinou potřebuje více dat, než kolik jich přijme pomocí parametrů, a proto o ně požádá model.
4. Model přijme požadavek od řadiče, a protože ví, jak komunikovat s databází, řekne si o potřebná data.
5. Když je model hotov a všechna data zapsal nebo přečetl z databáze, pošle výsledek své práce zpět řadiči.
6. Řadič zpracuje získaná data a odešle je pohledu.
7. Pohled ví, jak má vypadat výstupní stránka, pouze do ní vloží data získaná z řadiče. Výsledek se odesílá zpět uživateli jako odpověď na jeho požadavek.

3.6.4 Objektový přístup

CakePHP je naprogramován objektově a to znamená, že model, pohled i řadič jsou ve skutečnosti třídy.

Chceme-li vytvořit např. nový řadič se jménem *recepty*, musíme dodržovat zásady pro tvoření názvů. Třída se musí jmenovat *ReceptyController* a musí být potomkem třídy *AppController*. Třída *AppController* je v celé aplikaci jenom jedna a všechny řadiče jsou její potomky. Proto lze metody a atributy, které budeme potřebovat ve všech řadičích napříč aplikací, umístit právě do ní. Obě tyto třídy se nacházejí v adresáři s naší aplikací a můžeme je libovolně upravovat. Při zkoumání třídy *AppController* zjistíme, že je potomkem třídy *Controller*. To už je vnitřní třída frameworku a zpřístupňuje nám důležité metody jako je například *set*, která odesílá data do pohledu.

Podobnou hierarchii má i model. Třída pro model receptů se musí jmenovat pouze *Recept* a musí být potomkem třídy *AppModel*, ten je zase potomkem třídy *Model*. Za povšimnutí stojí, že název modelu musí být podle zvyklostí v jednotném čísle, zatímco název řadiče i tabulky v databázi v čísle množném. CakePHP totiž obsahuje „inflector“, který umí převádět podstatná jména z jednotného do množného čísla a naopak. U anglických názvů vše funguje samo a takto pojmenované objekty se správně najdou. České názvy se chtějí převádět podle pravidel pro anglické, a proto jim je potřeba pomoct. Naštěstí je to jenom otázka přidání použitých českých slov do pole v konfiguračním souboru aplikace.

Při vytváření pohledu, na rozdíl od řadiče a modelu, nebudeme vytvářet třídu, ale šablonový soubor. Je to klasický HTML soubor s možností používat PHP příkazy. V našem příkladu s receptem musejí jeho pohledy být umístěny v adresáři *recepty* a musejí být pojmenované stejně jako akce řadiče, pro kterou budou zobrazovat výsledky. Lze je pojmenovat i jinak, ale v řadiči pak musí být uvedeno, jaký pohled se má použít. Koncovku musí mít ale vždy **.ctp*[4].

3.6.5 Rozšíření modelu, pohledu a řadiče

Při programování aplikací se často stává, že je potřeba opakovaně řešit stejný problém. Například ukládat data v databázi do stromové struktury, zobrazovat formulá-

[4] Koncovky **.ctp* se používají ve verzích CakePHP 1.2.0 a novější. U starších byly **.thtml*.

řové prvky nebo ověřovat, zda je uživatel přihlášen. CakePHP umožňuje používat rozšíření modelu, pohledu a řadiče o chování, pomocníky a komponenty.

Model lze snadno nastavit, aby se choval jako strom. Stačí nám k tomu jenom využít toto chování (*behavior*). Rozšíříme tak funkce modelu o možnosti přidávat budovat v databázi strom, vytvářet větve, přesouvat je, mazat…

Model lze rozšiřovat ještě pomocí zdrojů dat (*data sources*). Ne vždy budeme potřebovat číst a zapisovat data do databáze. Můžeme například používat XML soubory. Tyto třídy tedy říkají modelu, jak má přistupovat k datům.

Programování řadičů si lze velice usnadnit použitím komponent (*components*). Nemusíme tak pracně vymýšlet, jak zajistit autorizaci uživatelů nebo jak odeslat email.

V pohledech se používají pomocníci (*helpers*), kteří umožňují generovat HTML elementy pomocí PHP příkazů. Výhodou takového generování je například HTML element pro odkaz. Při jeho psaní je potřeba uvést URL adresu, kam odkazuje. Použitím pomocníka můžeme uvést jméno řadiče, akce a další parametry a podle toho se nám URL vygeneruje automaticky. To se hodí v případě, že aplikace využívá směrování adres. CakePHP totiž umožňuje směrovat jakoukoli adresu na konkrétní řadič a akci. Adresa *http://www.domena.cz/recepty* může být přesměrována na řadič surovin s akcí pro jejich přidávání.

CakePHP přichází se spoustou již hotových pomocníků, chování i komponent. Ty se proto musejí před použitím v konkrétním řadiči a modelu povolit. Pomocníci pro pohled se povolují v řadiči, který ho volal. Zrychluje se tím celá aplikace, protože se načítají pouze potřebné části frameworku. Samozřejmě lze vytvářet vlastní komponenty, pomocníky i chování nebo vytvořit potomka s vylepšenými vlastnostmi a používat pak toho.

3.6.6 Struktura souborů

Po stažení a rozbalení archivu s CakePHP se na disku vytvoří adresáře a soubory, jako v následující tabulce.

app	Adresář se soubory naši aplikace.
cake	Adresář se soubory potřebnými pro správnou funkci frameworku.
vendors	Adresář určený pro soubory třetích stran. Budou dostupné ve všech aplikacích. (JavaScriptové frameworky, PHP třídy…)
.htaccess	Konfigurační soubor pro webový server apache, který přesměrovává požadavky na správný skript.
Index.php	Pokud webový server nepodporuje mod_rewrite, provede přesměrování místo souboru .htaccess.
README	Informace o frameworku.

Tabulka 1. Struktura souborů CakePHP

Adresář s aplikací se může jmenovat libovolně. Rovněž může být v adresáři více adresářů s dalšími aplikacemi a všechny budou využívat stejné soubory frameworku v adresáři cake. Adresář s cake může být umístěn kdekoli na serveru, ale aplikace musí mít v konfiguraci uvedenou cestu k němu.

Adresářová struktura samotné aplikace je už složitější.

config	Obsahuje konfigurační soubory aplikace. Informace o připojení k databázi, soubor s pravidly pro převod podstatných jmen z jednotného do množného čísla, směrování URL adres na řadiče a konfiguraci jádra aplikace.
controllers	Obsahuje všechny řadiče a adresář *components* pro jejich komponenty.
locale	Obsahuje soubory s překladem textových řetězců použitých v aplikaci pro jiné jazykové mutace aplikace.
models	Obsahuje všechny modely a také adresář *behaviors* pro určení jejich chování a ještě adresář *datasources* se soubory pro jiné zdroje dat.
plugins	Obsahuje zásuvné moduly aplikace.
tmp	Do tohoto adresáře si CakePHP ukládá dočasná data. Je proto potřeba, aby byl nastaven s právem pro zápis.
vendors	Adresář určený pro soubory třetích stran. Budou dostupné pouze v této aplikaci. (JavaScriptové frameworky, PHP třídy…)
views	Adresář pro všechny pohledy a soubory jiných šablon. Např. jak má vypadat celá stránka (layout)
webroot	Adresář, který je považován za kořenový adresář aplikace. Obsahuje obrázky, soubory CSS…
app_controller.php	Soubor s rodičovskou třídou všech řadičů aplikace.
app_helper.php	Soubor s rodičovskou třídou všech pomocníků aplikace.
app_model.php	Soubor s rodičovskou třídou všech modelů aplikace.
.htaccess	Konfigurační soubor serveru apache, který provádí přesměrování požadavků na správný soubor.
index.php	Pokud server nepodporuje mod_rewrite, pro přesměrování se použije tento soubor.

Tabulka 2. Struktura souborů aplikace

4 Popis vytvořené aplikace

4.1 Požadavky aplikace

- Webový server s podporou PHP (Apache)
- Databázový server MySQL

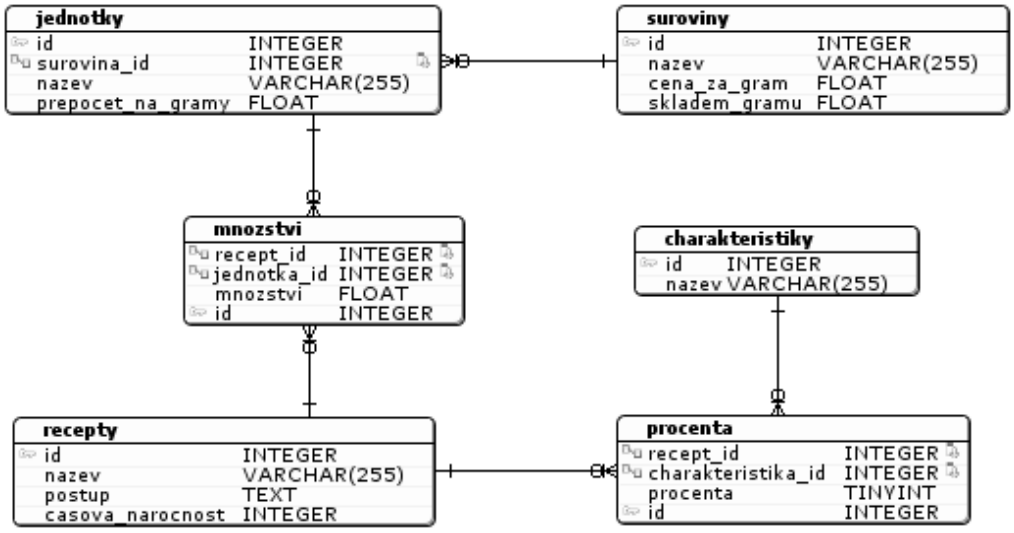
Obě tyto služby jsou dostupné na více operačních systémech a jsou k dispozici zdarma. Aplikace byla vyvíjena v prostředí Windows XP s nainstalovaným balíkem XAMPP, který obsahuje webový server Apache, MySQL i PHP.

4.2 Použité technologie

- Databázový systém MySQL se zpracovatelem tabulek MyISAM
- HTML, CSS
- JavaScript s frameworkem jQuery
- PHP s frameworkem CakePHP

4.3 Návrh databáze

4.3.1 Tabulky



Obrázek 3. E-R diagram databázové struktury

Databáze je navržena tak, aby umožňovala co největší pohodlí při zadávání receptů. Totiž v každé kuchařce mohou být u potřebných surovin různé jednotky a někde dokonce i jednotky nestandardní, jako jsou například hrneček, polévková lžíce a podobně. Proto databáze obsahuje tabulku *jednotky*, ve které má každá surovina uloženy své jednotky. Pomocí této tabulky také dochází k přepočtům speciálních

jednotek na gramy, které používá tabulka *suroviny* pro uložení stavu skladu a ceny suroviny.

Recept může mít mnoho surovin a mnoho charakteristik. Stejně tak jedna surovina a jedna charakteristika může být použita v mnoha receptech. Jedná se o vztahy M:N a musejí být realizovány prostřednictvím spojovacích tabulek. Tyto tabulky se nazývají *mnozstvi* a *procenta*. Každá z nich navíc ještě nese specifickou informaci o spojení. U spojení mezi receptem a surovinou nás zajímá, jak velké množství suroviny je potřeba. Mezi receptem a surovinou je to informace, na kolik procent recept splňuje konkrétní charakteristiku.

Pro všechny tabulky je použit zpracovatel MyISAM, který nepodporuje cizí klíče a tedy i vztahy mezi tabulkami včetně integritních omezení. Důvodem pro použití tohoto zpracovatele je možnost použití aplikace i na zdarma dostupných webhostinzích, které mají ve většině případu pouze MyISAM. Pro aplikaci je použit framework CakePHP a ten se dokáže do určité míry o vztahy mezi tabulkami postarat, a tak je integrita databáze zajištěna.

4.3.2 Indexy

Primární a cizí klíče jsou datového typu *UNSIGNED INT*, tedy kladné celé číslo. S tímto typem dokáže obyčejný index pracovat nejefektivněji. Navíc tyto klíče jsou nejčastěji vyhledávanými sloupci v každé tabulce. Jsou proto ideálními kandidáty pro indexování. Primární klíče obsahují index již sami o sobě, a proto se o ně starat nemusíme. Ale jak už bylo zmíněno je použit zpracovatel MyISAM a ten cizí klíče nepodporuje, tyto sloupce jsou pro něho proto stejné jako každý jiný sloupec. Přidáním indexů k těmto sloupcům proto velice zrychlíme všechny SQL dotazy, které spojují více tabulek.

4.3.3 Pohledy pro výběr optimálního receptu

Algoritmus pro výběr optimálního receptu k uvaření, s ohledem na udržování co nejlepšího stavu zásob, je realizován jako pohled uložený v databázi. Celý složitý výpočet se proto odehrává pouze na databázovém serveru a je proto rychlejší, než kdyby byl realizován jako funkce v jazyce PHP, která by se musela databáze neustále opakovaně dotazovat na potřebné informace. Recepty se řadí postupně podle třech kritérií. Každé kriterium je pro přehlednost reprezentováno samostatným pohledem.

První z nich spočítá, kolik je potřeba utratit peněz za nákup dalších surovin (na skladě není dostatečné množství) potřebných pro přípravu receptu.

```
create or replace view poradi_cena as (

select mnozstvi.recept_id,

sum(if(
  mnozstvi.mnozstvi*jednotky.prepocet_na_gramy<suroviny.skladem_gramu,0,
  (
    mnozstvi.mnozstvi*jednotky.prepocet_na_gramy-suroviny.skladem_gramu
  )*suroviny.cena_za_gram
))
as cena_dalsich_surovin

from  mnozstvi
left join jednotky on mnozstvi.jednotka_id=jednotky.id
left join suroviny on jednotky.surovina_id=suroviny.id

group by mnozstvi.recept_id
);
```

Je použita funkce *if(výraz1,výraz2,výraz3)*, která funguje tak, že pokud je první výraz pravdivý, vrací hodnotu druhého výrazu, jinak vrací hodnotu třetího výrazu. Takže pokud je na recept potřeba méně surovin, než kolik je na skladě, vrátí nulu, jinak vrací rozdíl mezi počtem gramů potřebných pro uvaření receptu a počtem gramů na skladě vynásobený cenou gramu suroviny. Nakonec se sečtou vypočtené hodnoty pro každý recept. Všechny recepty, pro které jsou potřebné suroviny k dispozici, mají jako výsledek výpočtu nulu, a proto se dají dále řadit podle dalších upřesňujících kriterií.

Druhé kritérium znevýhodní ty recepty, které spotřebují surovinu, jíž je na skladě nejmenší množství.

```
select recepty.id as recept_id,
suroviny.id as surovina_id,

ifnull((
  select mnozstvi.mnozstvi*jednotky.prepocet_na_gramy
  from mnozstvi
  left join jednotky on mnozstvi.jednotka_id=jednotky.id
  where mnozstvi.recept_id=recepty.id and jednotky.surovina_id=suroviny.id
),0)
as potrebne_mnozstvi

from suroviny, recepty
```

Tento SQL dotaz vygeneruje tabulku, ve které jsou uvedeny na samostatných řádcích všechny kombinace receptů a surovin (včetně těch, které nejsou pro recept potřeba). Pokud je na řádku recept a surovina, která je v tomto receptu uvedena, zobrazí se ve sloupci nazvaném *potrebne_mnozstvi* množství suroviny potřebné pro uvedený recept. Pokud není v receptu uvedena, zobrazí se ve sloupci potřebné množství hodnota *NULL*. Protože s hodnotou *NULL* se nedají provádět matematické operace, pomocí funkce *ifnull(výraz1,výraz2)*[5] se nahradí za skutečnou nulou.

Ve výsledku nás nezajímá potřebné množství, ale množství, které zbude na skladě po uvaření receptu. Toho dosáhneme tímto:

```
select recepty.id as recept_id,
suroviny.id as surovina_id,

suroviny.skladem_gramu-ifnull((
  select mnozstvi.mnozstvi*jednotky.prepocet_na_gramy
  from mnozstvi
  left join jednotky on mnozstvi.jednotka_id=jednotky.id
  where mnozstvi.recept_id=recepty.id and jednotky.surovina_id=suroviny.id
),0)
as potrebne_mnozstvi

from suroviny, recepty
```

[5] *Ifnull(výraz1,výraz2)* funguje tak, že pokud není první výraz null, je výsledkem jeho hodnota, jinak je výsledkem hodnota výrazu druhého

Nyní stačí pouze řádky seskupit podle id receptu a na vypočítávaný sloupec aplikovat funkci *min()*.

```
create or replace view poradi_minimum as (

select recepty.id as recept_id,

min(
suroviny.skladem_gramu-ifnull((
   select mnozstvi.mnozstvi*jednotky.prepocet_na_gramy
   from mnozstvi
   left join jednotky on mnozstvi.jednotka_id=jednotky.id
   where mnozstvi.recept_id=recepty.id and jednotky.surovina_id=suroviny.id
),0)
)as min_surovina_skladem

from suroviny, recepty

group by recepty.id

);
```

Recepty s nejmenší hodnotou vypočítávaného sloupce jsou nejnevhodnější.

Toto kritérium může často vyhodnotit více receptů jako stejně vhodné, protože potřebují jiné suroviny než tu, které je na skladě nejméně. Proto je potřeba použít ještě poslední kritérium. Používá stejný výpočet jako předchozí, ale místo funkce pro zjištění minimální hodnoty využívá *avg()* – průměrná hodnota.

```
create or replace view poradi_minimum as (

select recepty.id as recept_id,

avg(
suroviny.skladem_gramu-ifnull((
   select mnozstvi.mnozstvi*jednotky.prepocet_na_gramy
   from mnozstvi
   left join jednotky on mnozstvi.jednotka_id=jednotky.id
   where mnozstvi.recept_id=recepty.id and jednotky.surovina_id=suroviny.id
),0)
)as min_surovina_skladem

from suroviny, recepty

group by recepty.id

);
```

Kombinací těchto tří kritérií vytvoříme celkový pohled, který setřídí všechny recepty od nejoptimálnějšího po nejhorší.

```
create or replace view poradi as (
select
poradi_cena.recept_id,
poradi_cena.cena_dalsich_surovin,
poradi_minimum.min_surovina_skladem,
poradi_prumer.avg_surovina_skladem
from poradi_cena
left join poradi_minimum on poradi_minimum.recept_id=poradi_cena.recept_id
left join poradi_prumer on poradi_prumer.recept_id=poradi_cena.recept_id

order by
cena_dalsich_surovin ASC,
min_surovina_skladem DESC,
avg_surovina_skladem DESC
);
```

Při celkovém setřídění se nejprve zařadí úplně na konec ty recepty, které nemají všechny suroviny na skladě a seřadí se podle ceny za dokoupení potřebných surovin. Poté se nad ně zařadí ty recepty, které spotřebují surovinu, které je nejméně na skladě. A nakonec jako nejvhodnější se vybere ten recept, který minimálně sníží průměrný stav zásob. Postup řazení znázorňuje také následující obrázek. Tímto se stav skladu snaží být vždy ve stavu, kdy je možné uvařit co největší počet receptů.

recept_id	cena_dalsich_surovin	min_surovina_skladem	avg_surovina_skladem
4	0	10	260
1	0	10	186.666666666667
3	0	9	369.333333333333
2	0	0	300
5	12	-3	332.333333333333
6	1204	-400	199.666666666667

Obrázek 4. Řazení vypočítaných sloupců receptu

4.3.4 Pohled pro seznam surovin k nákupu

Čas od času vyrazíme na nákup. Většinou s sebou máme seznam se surovinami, které nám docházejí. Když už máme vytvořenou databázi, proč bychom ji nevyužili k vygenerování takovéhoto užitečného seznamu? Lze modifikovat pohled z předchozí kapitoly.

```
create or replace view nakup as (
select suroviny.id as surovina_id,

max(
ifnull((
   select mnozstvi.mnozstvi*jednotky.prepocet_na_gramy
   from mnozstvi
   left join jednotky on mnozstvi.jednotka_id=jednotky.id
   where mnozstvi.recept_id=recepty.id and
   jednotky.surovina_id=suroviny.id
),0)
)-suroviny.skladem_gramu as chyby_gramu
from suroviny, recepty
group by surovina_id
having chyby_gramu>0
);
```

Pohled pro každou surovinu najde recept, ve kterém je ji potřeba nejvíce a od tohoto potřebného množství odečte celkové množství na skladě. Odečtením se získá kladné číslo v případě, že je potřeba surovinu dokoupit a záporné v případě, že skladové zásoby stačí. Nakonec pohled zobrazí pouze kladné hodnoty díky podmínce *HAVING*. Tímto získáme přehledný seznam všech surovin, které je potřeba dokoupit abychom byly schopni uvařit jakýkoliv recept.

4.4 PHP aplikace

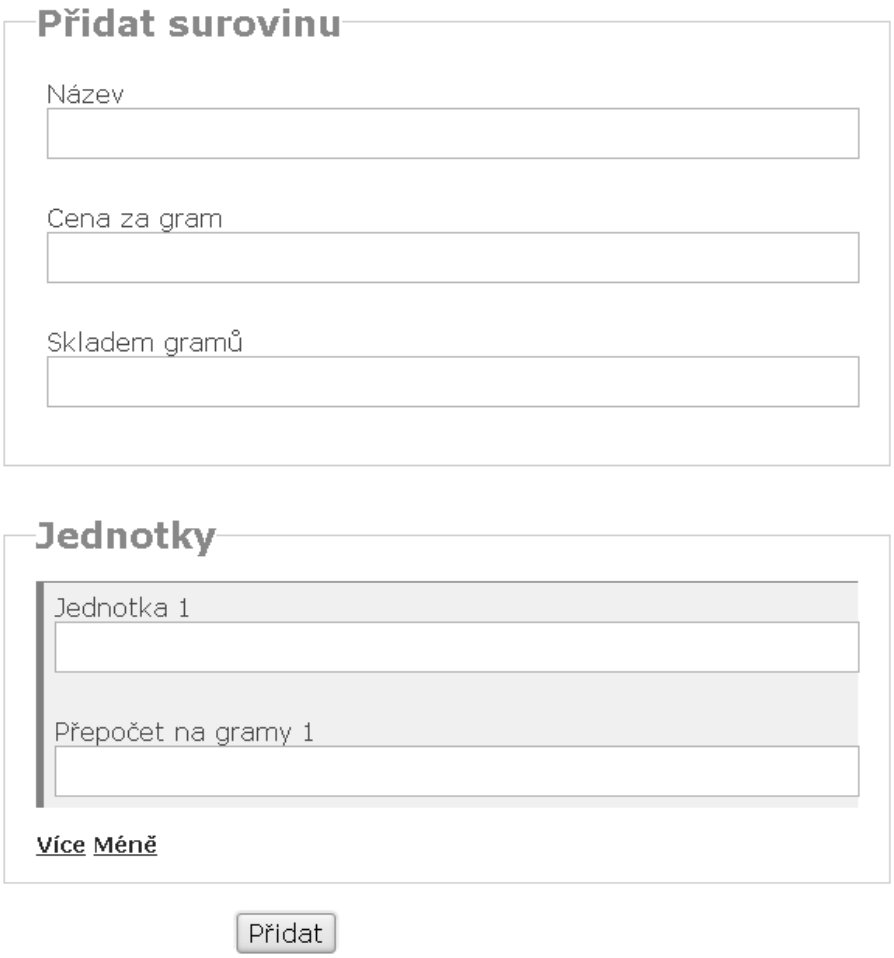
Aplikace běží na frameworku CakePHP a protože cílem aplikace jsou algoritmy pro optimalizaci skladu, byl ji ponechán standardní vzhled frameworku.

Obrázek 5. Obrazovka aplikace - Nejlepší recept

Aplikace se skládá z layoutu, který obsahuje záhlaví s odkazy pro procházení mezi jednotlivými částmi aplikace a zápatí. Layout zůstává na každé stránce stejný. Ze záhlaví a menu s odkazy lze vyčíst, že celá aplikace je rozdělena do dvou částí: editační a provozní.

4.4.1 Editační část

Tato část je určena přímo pro práci s daty uloženými v databázi. Umožňuje pohodlně editovat recepty, suroviny a charakteristiky, včetně jejich vztahů. Nejprve je potřeba vložit do databáze suroviny.



Obrázek 6. Obrazovka aplikace - Přidávání suroviny

Jedna surovina může mít více jednotek, a proto je pro HTML formulář použit ještě navíc javascript, který po kliknutí na odkaz *Více* přidá políčka pro zadání další jednotky. Naopak odkaz *Méně* odebere políčka pro poslední jednotku. Tímto lze pohodlně z jedné stránky vytvořit kompletně celou jednu surovinu.

Dále je potřeba vytvořit charakteristiky, které budou přidávány k receptům. Formulář pro jejich vytvoření je podobný jako u přidávání surovin, pouze je chudší o dynamickou část (žádná taková data u ní nejsou potřeba).

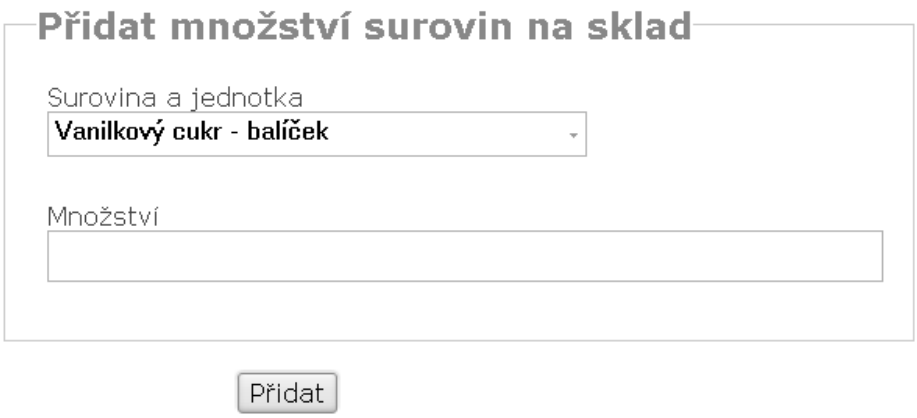
Nakonec můžeme vytvořit samostatný recept. Formulář je opět stejný jako u přidávání surovin ale je větší. Jeden recept totiž může obsahovat více surovin a rovněž

více charakteristik. Proto má dynamické části hned dvě. Jednu pro suroviny a druhou pro charakteristiky.

Stejně jako přidávat nová data, lze již uložená editovat a mazat. Mazání je však pro zachování integrity dat omezeno. Nelze například smazat surovinu, která má nějakou svou jednotku obsaženou v nějakém receptu. Z receptu by se taková surovina ztratila a už bychom nevěděli, jaké suroviny potřebujeme. Na druhou stranu, pokud smažeme recept, smažou se informace o tom, kolik má které suroviny a jaké má charakteristiky. Mažeme řádky ze třech tabulek, aby nedocházelo ke zbytečnému hromadění nepotřebných dat.

4.4.2 Provozní část

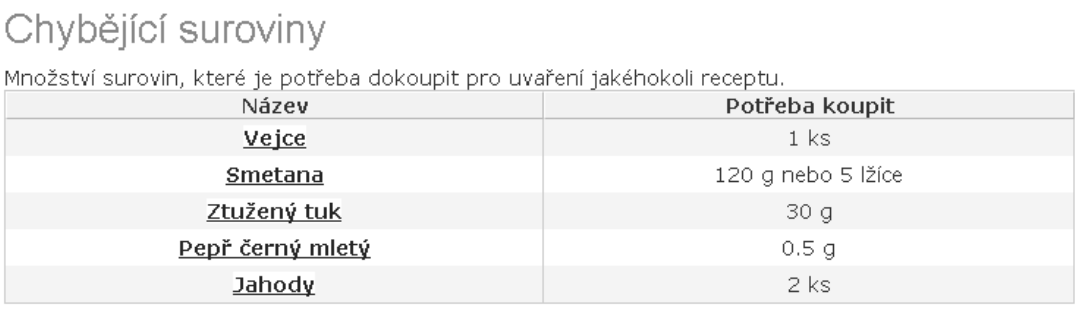
Tato část obsahuje všechny operace s daty potřebnými pro běžný provoz. Nové recepty, suroviny nebo charakteristiky se nepřidávají tak často, jako pouze zvyšování množství určité suroviny na skladě. Tato operace se děje s každým novým nákupem, a proto by bylo zbytečně nepohodlné vždy surovinu najít v seznamu, kliknout na odkaz pro editaci a k jejímu počtu na skladě přičíst přidávané množství. Aplikace proto obsahuje stránku nazvanou *Nákup surovin.*



Obrázek 7. Obrazovka aplikace - Nákup surovin

Po nákupu vybereme surovinu z nabídky i s jednotkou, kterou budeme zadávat do políčka *Množství*. Pokud přidání proběhne v pořádku, dostaneme se zpět na stejnou stránku a můžeme přidávat další surovinu.

Abychom věděli co nakupovat, stačí se podívat na stránku *Chybějící suroviny*.



Chybějící suroviny

Množství surovin, které je potřeba dokoupit pro uvaření jakéhokoli receptu.

Název	Potřeba koupit
Vejce	1 ks
Smetana	120 g nebo 5 lžíce
Ztužený tuk	30 g
Pepř černý mletý	0.5 g
Jahody	2 ks

Obrázek 8. Obrazovka aplikace - Chybějící suroviny

V tabulce vidíme, kolik které suroviny je potřeba koupit, abychom mohli uvařit jakýkoliv recept z databáze.

Nejdůležitější stránka je ale pod odkazem *Nejlepší recept* (Obrázek 5.). Zde je tabulka receptů seřazených od shora podle toho, který je nejlépe uvařit, aby zůstalo na skladě tolik surovin, že půjde příště uvařit co nejvíce receptů. Lze také zvolit, jestli se mají zobrazit všechny recepty nebo jenom ty s určitou charakteristikou. Při zvolení specifické charakteristiky se tabulka rozšíří o další sloupec, který ukazuje, jak moc recept charakteristice odpovídá. Tabulku můžeme rozšířit ještě o další tři sloupce kliknutím na odkaz *Více informací*.

Nejoptimálnější recepty

Zobrazit pouze: vše, Moučník, Hlavní chod

Více informací skrýt/zobrazit

Strana 1 z 1, zobrazuje 6 záznamů z 6 celkem, první je záznam 1 a poslední 6

Pořadí	Název	Moučník	Cena dokoupených surovin	Minimum skladem	Průměr skladem
1	Margotkové řezy	100%	0kč	0	437.24
2	Pařížské kostky	100%	0kč	0	392
3	Bábovka z pomazánkového másla	100%	2.15kč	-30	419
4	Tvarohový biskupský chlebíček	100%	3.5kč	-53	407.41
5	Tvarohová bábovka s rozinkami	100%	3.5kč	-53	407.07
6	Smažené oříškové kroužky	100%	3.5kč	-53	404.86

Obrázek 9. Obrazovka aplikace - Nejlepší recept (rozšířené)

Jak se hodnoty spočítali a jak se recepty podle nich řadí se lze dozvědět kapitole 4.3.3.

4.4.3 Ukázky kódu

Aby správně fungoval převod českých slov z množného do jednotného čísla, je potřeba hned ze začátku nastavit v souboru *config/inflections.php* používané české názvy. Následující řádek nastaví slova, která se nebudou měnit.

```
$uninflectedPlural = array('mnozstvi','procenta','poradi','nakup');
```

Následující řádek nastaví jednotná a množná čísla ostatních slov.

```
$irregularPlural = array('recept'=>'recepty', 'jednotka'=>'jednotky',
'surovina'=>'suroviny', 'charakteristika'=>'charakteristiky');
```

Při ladění aplikace se nám může hodit výpis SQL příkazů, čas potřebný k jejich zpracování a případné chyby. Tyto informace se zobrazují naspodu každé stránky. Při používání aplikace uživatel jsou tyto informace jednak rušivé, ale také z nich uživatel může vyčíst ledasco o struktuře databáze a tyto informace pak zneužít. Proto se v konfiguračním souboru *config/core.php* nachází tento řádek:

```
Configure::write('debug', 0);
```

Změnou čísla se přepíná mezi ladícím a produkčním módem. Ladící má číslo 2 a produkční 0. V produkčním módu jsou potlačeny výpisy všech upozornění a ladících hlášek a výpisů.

4.4.3.1 Model

```
7  class Mnozstvi extends AppModel {
8      var $name = 'Mnozstvi';
9      var $validate = array(
10         'jednotka_id' => array('numeric'),
11         'mnozstvi'=>array(
12             'number'=>array(
13                 'rule'=>array('custom','/^[0-9]+(\.[0-9]+)?$/'),
14                 'message'=>'Musí být zadáno číslo.
15                             (Celé nebo desetinné s tečkou.)'
16             )
17         )
18     );
19     var $belongsTo = array(
20         'Recept' => array(
21             'className' => 'Recept',
22             'foreignKey' => 'recept_id'
23         ),
24         'Jednotka' => array(
25             'className' => 'Jednotka',
26             'foreignKey' => 'jednotka_id'
27         )
28     );
29 }
```

Obrázek 10. Příklad kódu modelu

Z řádku 7 lze vyčíst, že třída modelu *Mnozstvi* je potomkem třídy *AppModel*. Na dalším řádku je v atributu třídy uložen její název, aby o tomto názvu věděl i CakePHP a mohl s ním pracovat.

Řádky 9 až 18 obsahují validační pole. Zde se dá nastavit, jaká data se mohou do kterého sloupce tabulky databáze ukládat a také jaká se má zobrazit chybová hláška u HTML formuláře v případě, že zadané kritérium není splněno. CakePHP má mnoho předdefinovaných kritérií (např. *numeric* na řádku 10) a pokud potřebujeme nějaké specifičtější, lze ho naprogramovat pomocí regulárního výrazu (např. řádek 13).

Na řádcích 19 až 28 se nachází pole s nastavením vazby na další modely. Ze zápisu je zřejmé, že tento model (*Mnozstvi*) patří modelu *Recept* a také modelu *Jednotka*. Z pohledu databáze to znamená, že jeden záznam z tabulky *Recept* má mnoho záznamů v tabulce *Mnozstvi*. Stejně tak jeden záznam z tabulky *Jednotka*.

4.4.3.2 Řadič

```
 7  class SurovinyController extends AppController {
 8  var $name = 'Suroviny';
 9  var $uses = array('Surovina','Nakup');
10
11  function index() {
12      $this->set('suroviny', $this->paginate());
13  }
14
15  function edit($id = null) {
16      if(!is_numeric($id) || !$this->Surovina->hasAny(array('Surovina.id'=>$id))){
17          $this->Session->setFlash('Chybné id suroviny.');
18          $this->redirect(array('action'=>'index'));
19      }
20      if (!empty($this->data)) {
21          if ($this->Surovina->save($this->data)) {
22              $this->Session->setFlash('Surovina byla v pořádku upravena.');
23              $this->redirect(array('action'=>'view',$id));
24          } else {
25              $this->Session->setFlash('Surovina nemohla být upravena.');
26          }
27      }
28      if (empty($this->data)) {
29          $this->data = $this->Surovina->read(null, $id);
30      }
31  }
```

Obrázek 11. Ukázka kódu řadiče

Na řádku 7 je jméno třídy, které musí být podle zvyklostí CakePHP v takovémto tvaru. Další řádek je stejně jako u modelu pouze atribut se jménem řadiče. Řádek 9 obsahuje pole se seznamem modelů, které daný řadič bude využívat. Pokud by nebyl uveden žádný, framework automaticky předpokládá, že se použije právě jeden a se jménem jako má řadič, ale v jednotném čísle (např. řadič *Suroviny* má model *Surovina*).

Na řádcích 11 až 14 je popsána metoda třídy (nebo v řeči CakePHP akce) třídy. Tato konkrétní pouze uloží do proměnné *suroviny* data z databáze. Proměnná bude přístupná pouze v pohledu, který řadič zavolá. K získání dat se v tomto případě používá komponenta *paginate*, která umožní v pohledu zobrazit stránkování pomocí pomocníka *paginator*. Nejčastěji se pro získávání dat používá příkaz:

```
$this->Model->find('all');
```

V další akci dojde nejprve na řádcích 16 až 19 k ověření, zda zadaný parametr je číslo a zda je toto číslo ID nějaké suroviny. Pokud ne, nastaví se chybová hláška a dojde k přesměrování na jinou adresu. Dále se na řádcích 20 až 27 ověří, zda přicházejí nějaká data metodou POST (řádek 20) a pokud ano, na řádku 21 se je model po-

kusí uložit. Podle toho zda se uložení povedlo nebo ne, dojde k vypsání patřičné hlášky a případnému přesměrování jinam. Nakonec se ještě provede zkouška, zda přicházejí pomocí POST data nebo ne (řádek 28) a když ne, jako data se uloží data načtená z modelu. Tato data se totiž v pohledu zobrazí ve formě vyplněného formuláře.

4.4.3.3 Pohled

```
2  <?php echo $form->create('Surovina');?>
3      <fieldset>
4          <legend>Upravit surovinu <?=$form->value('Surovina.nazev');?></legend>
5      <?php
6          echo $form->input('id');
7          echo $form->input('nazev',array('label'=>'Název'));
8          echo $form->input('cena_za_gram',array('label'=>'Cena za gram'));
9          echo $form->input('skladem_gramu',array('label'=>'Skladem gramů'));
10     ?>
11     </fieldset>
12 <?php echo $form->end('Upravit');?>
```

Obrázek 12. Ukázka kódu pohledu

Pohled není třída, a proto na rozdíl od modelu a řadiče začíná rovnou kódem. Ukázka (Obrázek 12) zobrazuje pouze část pohledu pro řadič *Suroviny* a jeho akci *edit*, sloužící pro úpravu dat. Pro vytvoření formuláře je použitý pomocník *FormHelper*, který je přímo součástí frameworku.

Na řádku 2 vytvoříme začáteční tag HTML formuláře. Metoda a vše potřebné se vygeneruje automaticky. Konec HTML formuláře je na řádku 12 a kromě samotného ukončení se automaticky přidává tlačítko pro potvrzení s textem *Upravit*. Na řádcích 6 až 9 se nacházejí samotné prvky formuláře. Nikde není zadáno, zda mají být zobrazeny jako textové pole nebo políčko na jeden řádek textu. Vše pozná CakePHP automaticky podle datových typů v databázi. Rovněž automaticky doplňuje hodnoty z řadiče do správných prvků formuláře a v případě chyby vypíše hlášku u políčka, do kterého uživatel chybná data zadal.

5 Závěr

Návrh a realizace systému pro optimalizaci využití zásob vyžaduje zkušenosti s navrhováním databází a rovněž zkušenosti s programováním. Důležité je si správně navrhnout celou databázovou strukturu a až potom začít vyvíjet samotnou aplikaci. Veškeré změny v databázi totiž znamenají více či méně změn v samotné aplikaci a často se stane, že se veškeré potřebné změny neodhalí a vznikají tak těžko odhalitelné chyby. Použití frameworku CakePHP zjednodušilo a zrychlilo vývoj samotné aplikace. Díky oddělení aplikační logiky od prezentace dat lze snadno modifikovat ať už vzhled nebo jakoukoli část a dokonce lze snadno migrovat na jinou databázi.

Pro tuto bakalářskou práci byl však klíčový návrh algoritmu pro optimalizaci využití zásob. Postupné řazení receptů podle navržených výpočtů toto zadání splňuje. Tyto výpočty by se daly použít i pro jiné sklady, nejenom pro suroviny a recepty. Nicméně určitě by se problém optimalizace dal řešit i jinými cestami.

Pokud by rostl objem uložených surovin a jejich jednotek, bylo by určitě přínosné na stránkách, kde se volí recept a jednotka společně, tyto dvě věci oddělit do samostatných nabídek a jednotky vždy aktualizovat podle zvolené suroviny.

5.1 Použité zdroje

[1] DUBOIS, Paul. *MySQL profesionálně : komplexní průvodce použitím, programováním a správou MySQL*. 1. vyd. Brno : Mobil Media, 2003. 1072 s. ISBN 80-86593-41-X.

[2] GOLDING, David. *Beginning CakePHP : From Novice to Professional*. 1. vyd. California : APress, 2008. 344 s. ISBN10: 1-4302-0977-1.

[3] *MySQL : Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-04-13]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/MySQL>.

[4] *Relační databáze : Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-04-13]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/RDBMS>.

[5] *PHP : Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-04-13]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/PHP>.

[6] PHP Documentation Group. *PHP : PHP Manual* [online]. 1997-2009 [cit. 2009-04-13]. Dostupný z WWW: <http://cz2.php.net/manual/en/>.

[7] *Návrhový vzor : Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-04-13]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/N%C3%A1vrhov%C3%BD_vzor>.

[8] *Model-view-controller : Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-04-13]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Model-view-controller>.

[9] Cake Software Foundation. *All things CakePHP : The Cookbook* [online]. [cit. 2009-04-13]. Dostupný z WWW: <http://book.cakephp.org/>.

[10] jQuery Team. *jQuery : Documentation* [online]. [cit. 2009-04-13]. Dostupný z WWW: <http://docs.jquery.com/>.