Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Návrh Web Content Management systému

Diplomová práce

Autor: Jakub Konopásek

Vedoucí práce: Ing. David Buchtela

Prohlášení

Prohlašuji, že jsem diplomovou práci na téma „Návrh *Web content management* systému“ zpracoval samostatně, pouze za odborného vedení vedoucího diplomové práce. Všechny materiály a zdroje, ze kterých jsem čerpal, jsou uvedeny v seznamu použité literatury.

V Praze dne 30. března 2011 .....

Jakub Konopásek

Poděkování

Tímto bych chtěl vyjádřit poděkování Ing. Davidu Buchtelovi za jeho odborné vedení a cenné rady při vedení mé diplomové práce. Dále bych rád poděkoval svému otci Zdeňku Konopáskovi za pomoc při stylistické úpravě práce a také všem ostatním, kteří mě pomáhali při její tvorbě.

Návrh Web Content Management systému

Design of Web Content Management System

Souhrn

Tato práce se zabývá tvorbou univerzálního systému, který umožňuje načítání dynamického menu a obsahu pro běžnou webovou prezentaci. Práce popisuje jak metodiku používanou při návrhu takovýchto systémů, tak praktické zpracování konkrétního navrhovaného systému. Vytvořený systém byl již během psaní této práce aplikován na několika webových stránkách. Takto získané praktické znalosti pak byly použity na jeho zdokonalení. Byly také odhaleny a opraveny chyby ve vlastním návrhu, kterým se u takto složitého systému dá těžko vyhnout. Vlastní programátorská práce zároveň umožnila výklad doplnit reálnými příklady využití navrženého systému.

Klíčová slova

Web content management systém; CMS; tvorba webové stránky; datové modelování; databáze; SQL; PHP.

Summary

This work describes creation of a universal system designed to load dynamic menu and content within a small- or middle-scale web page. It does not only explain methodology of production of such a system, but describes in detail practical procedures involved in the creation of particular web applications and their usage. This system for loading content and menus has already been used on several web sites, concurrently designed and developed by the author of this thesis. The knowledge gained from this practical use was used to further refine and strengthen the proposed system as well as to correct several design flaws. It also allowed this work to benefit from real-world examples.

Keywords

Web content management system; CMS; web page; data modeling; database; SQL; PHP.

Obsah

Úvod

V minulosti bylo běžnou praxí, že se webová prezentace skládala ze statických stránek. Dnes je většina webových stránek před odesláním uživateli vždy nejprve zpracována na straně serveru. Část obsahu stránky je běžně načítána z připojeného databázového systému. Obsah takovéto webové prezentace se pak může změnit jen pouhým přidáním řádku do správné tabulky. Statické webové stránky se proměnily v dynamické webové aplikace. Ty se stávají čím dál tím složitější a zobrazují mnohem více informací, které jsou strukturovány nejrůznějšími způsoby. S tím samozřejmě vzrostly i nároky na zobrazování, vkládání, údržbu, zpracování a správu celého internetového obsahu. Celou touto problematikou se zabývají takzvané web content management systémy.

Je zřejmé, že jde o velmi obsáhlé téma. Tato práce se zaměřuje pouze na úzkou část tohoto tématu, a to na část zabývající se návrhem systému pro zobrazování a strukturování obsahu webových stránek. Hlavní částí a přínosem této práce je návrh a popis dvou propojených systémů. První z nich se stará o uchovávání a vypisování dynamického menu z databáze. Druhý se zabývá načítáním různých druhů dynamického obsahu. Oba navržené systémy se skládají z návrhu databáze a objektově navržené webové aplikace, která z databáze načítá data a dále s nimi pracuje. Oba systémy se snaží být co nejuniverzálnější, aby je bylo možné rychle a snadno použít v nejrůznějších případech. Tím představují v rukách tvůrce stránek mocný a užitečný nástroj pro tvorbu mnoha druhů obsahu nejrůznějších webových prezentací. Zmíněné systémy byly navrženy v kombinaci jazyků a software pro dnešní běžné webové prezentace nejrozšířenější. Jedná se o jazyk PHP s napojením na systém řízení báze dat MySQL. Oba systémy již byly během psaní diplomové práce použity pro tvorbu obsahu řady webových aplikací. Součástí práce je tedy vysvětlení použití aplikací na reálných případech, včetně řady postřehů z praxe.

Cíl práce a metodika

Cíle práce

Hlavním cílem této diplomové práce je navrhnout dva systémy řešící problematiku načítání a zobrazování menu a obsahů pro dynamickou webovou stránku. Úkolem prvního z těchto systémů je starat se o načítání dynamického menu. Úkolem druhého je pak starat se o načítání obsahu, přičemž může spolupracovat se systémem prvním. Dovoluje tedy pro jednotlivé položky menu snadno vytvářet dynamický obsah. Navržené systémy by měly být určeny především administrátorům, kteří je mohou použít pro komplexní návrhy obsahu různých webových prezentací. Tyto systémy by měly být navrhnuty objektově a s menšími úpravami jsou tedy schopné pracovat s libovolným content management systémem. Druhým hlavním cílem práce je seznámit čtenáře s tím, co návrh takovýchto částí content management sytému obnáší, a jak vlastně práce na něm probíhá. Dílčí cíle práce se dají shrnout do následujících bodů:

- proniknout do problematiky *web content management* systémů;
- navrhnout logické schéma relační databáze;
- vytvořit vlastní návrh databáze;
- v jazyce PHP navrhnout vlastní objektové aplikace načítající obsah webové stránky z databáze.

Metodika

Při návrhu systémů zajišťujících dynamické načítání položek menu a obsahu do webové aplikace bylo použito následující metodiky. Nejprve byla provedena zběžná literární rešerše ohledně možných postupů a nástrojů používaných při navrhování podobných *web content management* systémů. Pro návrh systémů popsaných v této práci pak byla vybrána kombinace dnes nejpoužívanější – skriptovací jazyk PHP, který umožňuje tvořit objektově orientované aplikace, v kombinaci s relačním systémem řízení báze dat MySQL.

Následně byla provedena podrobná literární rešerše těch nástrojů a postupů, které jsme zvolili pro náš případ. Při tvorbě systémů byly pomocí analýzy use case stanoveny a

popsány funkce, které mají navrhované aplikace splnit. Zároveň byl s ohledem na metodologii návrhu relačních databází (včetně datové normalizace) vytvářen příslušný logický model, z něhož pak následně vzešel model funkční. Nakonec byly navrženy vlastní objektové aplikace v jazyce PHP. Budovaný systém byl průběžně testován na několika zpracovávaných webových stránkách. Z této praxe vzešla potřeba několikerých přepracování: byly opraveny chyby v návrhu a samozřejmě také ve vlastním popisu aplikací. Byly také doplněny některé pokročilejší funkce.

Přehled současných metod a technologií

Web content management systémy

V následující kapitole je vysvětlen pojem *web content management* systému. Jeho definice, programové zázemí, nejběžnější nástroje pro jeho tvorbu a jsou zmíněny i aktuálně nejrozšířenější *web content management* systémy.

Web content management systémy obecně

V moderním světě je stále běžnější, že lidé hledají na internetu informace, které v běžném životě potřebují,. Tento trend vede ke stále většímu počtu webových prezentací u různých obchodů, poskytovatelů služeb, podniku a také u fyzických osob. Webové prezentace je však třeba časem upravovat a udržovat tak jejich informativní hodnotu. Nejpřímější cesta je upravit obsah stránky přímo v souboru s kódem stránky. To však vyžaduje znalost jazyků, ve kterých byla daná stránka napsána. V nejjednodušším případě jsou to jazyky html / css. U složitějších stránek pak přibývá nějaký skriptovací jazyk jako PHP, ASP, javascript a jazyk na komunikaci s databází jako například SQL. Navíc u složitějších stránek je třeba se vyznat ve vlastní struktuře a rozdělení souborů, neboť to co uživatel vnímá v prohlížeči jako jednu webovou stránku, může být rozděleno do několika souborů v různých adresářích, ale často i databázích na jiných serverech než je vlastní webová aplikace.

Právě k usnadnění tvorby, administrace a editace obsahu webových stránek slouží *web content management* systémy. Většinou se tedy jedná o systém aplikací, které umožňují uživateli editovat obsah webové stránky, aniž by pracoval přímo s vlastními soubory webu.

Případně, když je třeba, aby se dostal přes uživatelské rozhraní přímo a pouze k vybraným částem těch souborů, které potřebuje. Aplikace *web content management* systému také většinou umožňují snadnou kolaboraci více uživatelů. Obsah webových stránek může tedy vzdáleně měnit kdokoli, kdo má k *web content management* systému přístup.

Co všechno a jakým způsobem umožňuje *web content management* systém měnit, se liší od systému k systému. V nejjednodušších případech se může jednat o jednoduché vkládání novinek přes webový formulář. Ty složitější *web content management* systémy pak umožňují editovat přes rozhraní skripty běžící na pozadí stránky přímo ve skriptovacím jazyce, vytvářet výpisy z databází atd. Součástí takových webových CMS pak je většinou rozdělení uživatelů na řadu úrovní a používat všechny funkce je umožněno pouze kvalifikovaným uživatelům. Zatímco běžnému uživateli se zobrazí pouze zjednodušená podoba.

Lze tedy říci, že *web content management* systémy přidávají aplikační vrstvu mezi administrátora / tvůrce webu a vlastní soubory webu, skrze kterou se dá měnit podle přednastavených pravidel vlastní obsah stránky.[1]

Proč (ne)používat web content management systémy

Web content management systémů je řada druhů a každý z nich má různé výhody a nevýhody, ale pro naprostou většinu platí následující.

Tvorba *web content management* systému je jak časově tak finančně mnohem nákladnější než tvorba normální webové aplikace. Díky tomu se *web content management* systém, nebo alespoň většina jeho částí, tvoří tak, aby šla používat znovu a znovu, na dalších a dalších webových stránkách. To samozřejmě přinese výhodu nízké ceny, neboť se cena za vývoj aplikace dělí mezi více zákazníků. Existují i univerzální *web content management* systémy, které jsou vyvíjeny zdarma jako open source, či pouze za symbolický poplatek. Mezi velkou výhodu těchto systémů patří, že se na nich podílí celá komunita lidí, která pro tyto systémy tvoří řadu užitečných doplňků a také je schopna velmi rychle odhalit a opravit možné chyby. Mezi nejrozšířenější takové webCMS patří například *web content management* systém Joomla! nebo Drupal. Jsou však i případy kdy univerzalita a pomoc komunity na vývoji cms systému není žádoucí či možná. Například pokud systém zajišťuje správu citlivých údajů, není vhodné aby mnoho lidí znalo způsob jakým funguje. Nebo je

struktura údajů zpracovávaných systémem natolik složitá, že je běžný systém, který není „na míru“, nezvládá. V tom případě je nutné navrhnout systém na míru a jeho cena je samozřejmě úměrná složitosti dané aplikace.

Ať je systém jak chce jednoduchý, je vždy nutné zaškolit pracovníky na jeho obsluhu. Pokud se na úpravu obsahu webové prezentace nepoužívá *web content management* systém je zpravidla nutné, aby úpravy webu dělal vysoce školený pracovník. *Web content management* systém zpravidla tuto práci velmi ulehčí a zrychlí, ale nutnost znalosti problematiky nikdy zcela neodstraní. Pokud se obsah webové prezentace mění například během roku pouze několikrát, častokrát přijde levněji zaplatit vždy za úpravy oborníkovi za úpravy než nechat tvořit *web content management* systém. Samozřejmě, že pokud velká část stránky se načítá z databází, které se často mění, je *web content management* systém ideálním způsobem správy. Nejjednodušším příkladem takovéhoto příkladu je například osobní blog. Je jen na zadavateli tvorby *web content management* systému, aby si rozmyslil, zda změny na jeho webové prezentaci budou natolik časté, aby ospravedlnily tvorbu *web content management* systému.

Výhoda většiny content management systémů, která je častokrát zmiňována je ta, že web content systémy jsou naprogramovány tak, že umožňují snadnou úpravu nejen obsahu, ale i designu stránky. Častokrát přímo přes webové rozhraní. Jde však tvrdit, že tato výhoda není tak velká jak se na první pohled zdá. Ve skutečnosti vyplývá z toho, že *web content management* systémy jsou programovány zkušenějšími programátory než ledajaká webová prezentace. Výhoda, že se dá změnit na jednom místě barva a ta se změní na všech instancích kde má, plyne z definice kaskádových stylů, které se používají k tvorbě designu webových stránek. A to samé platí o obsahu – pokud je stránka bez *web content management* systému dobře napsána, má věci co se vypisují několikrát zaznamenané pouze jednou a vždy se vypisuje jen jejich instance. *Web content management* systém vlastně tedy jen opět umožňuje možnost spravovat design přes webové rozhraní.

Jak bylo již několikrát zmíněno hlavní výhoda content management systému je, že řadu práce za odborníka v internetových technologií odpracuje počítač. Z toho vyplývá také značná nevýhoda content management systémů. Tj. že skript, který před odesláním webové stránky uživateli musí server zpracovat je mnohanásobně větší a složitější než u běžné webové stránky. To každopádně zatíží více server na kterém webová prezentace běží a

může dojít i k výraznému zpoždění přenosu stránky k uživateli. Velikost a složitost *web content management* systémů také znamená, že může mnohem snadněji dojít při zpracování skriptů k chybě. A to ať už kvůli špatně napsanému kódu, či kvůli tomu, že byl kód napsán na rozdílnou verzi softwarem než je ta co běží zrovna na serveru [2].

Druhy web content management systémů

Web content management systémy lze v základu rozdělit do několika skupin podle toho v jakých jazycích jsou napsány, resp. jaký software potřebují pro svůj běh. A jak dokážou tyto systémy nakládat s daty, tedy zda používají na správu dat databázi (a potřebují tedy spolupracovat se systém řízení báze dat), nebo zda ukládají data do souborů na disku atd.

Jazyky používané pro psaní *web content management* systému se dají rozdělit do dvou kategorií. První z nich jsou jazyky, které se používají běžně k psaní webových aplikací a zpracovávají se vždy přímo na serveru. Jedná se o skriptovací jazyky jako PHP či ASP. Správa webové prezentace přes takovýto systém lze provádět přímo na webu přes internetový prohlížeč a systém může i relativní amatér snadno zprovoznit na serveru pronajmutém od webového providera. Na druhou stranu jsou zde omezené možnostmi daného skriptovacího jazyka. Druhá kategorie jazyků jsou běžné programovací jazyky které poskytují lepší možnosti pro programátora. Častokrát umožňují spolupráci s celou řadou systémů řízení báze dat. Ale zároveň kladou velké nároky na výpočetní kapacitu serveru, a sama složitost instalace a zavedení systému je také mnohem větší než když je systém dělaný přímo v jazyku určeném pro psaní webových prezentací.

Mezi nejpoužívanější jazyky používané pro psaní *web content management* systémů se dají uvést ASP, Java, Perl, PHP a Python. A jako systémy pro řízení báze dat se používají Microsoft SQL, Mysql, Oracle, PostgreSQL nebo content management systém používá pro správu dat místo relační či objektové databáze pouze ukládání do souborů – tzv. file/flat file systém. Přičemž největší množství *web content management* systémů je pravděpodobně napsáno v kombinaci nástrojů PHP a MySQL. Což je přirozené, neboť vůbec většina webových stránek je napsaná v pro tento zdarma dostupný software. To je také důvod proč v je v druhé části této práce *web content management* systém psán v jazyce PHP a jako systém řízení báze dat používá právě MySQL [3].

Databázové systémy

V této kapitole je popsána způsob návrhu relačních databází. Lze bezesporu říci, že objektové databáze jsou, spolehlivější, mnohem rychlejší a snadněji se používají než databáze relační [3]. Navíc na rozdíl od databází relačních umožňují objektové databáze velice jednoduše používat generalizaci, agregaci a další nástroje objektového přístupu, které značně ulehčují jak navrhování webových i jiných aplikací, tak i jejich provoz. Přes všechny výhody, které mají objektové databáze nad relačními databázemi, jsou to relační databáze, které jsou v dnešním světě nejběžněji používanými databázemi pro *web content management* systémy. Je tomu tak především díky historickému vývoji výpočetní techniky. Relační databáze se zavedli ve chvíli kdy byly pro záznam dat používána pásková média a jejich přístup k zaznamenávání vazeb mezi entitami byl pro tento typ média vskutku mnohem výhodnější než přístup databází síťových (ze síťového modelu později vychází přístup objektový). Relační databáze si od té doby vybudovali obří základnu uživatelů, a i když důvod kvůli kterému se kdysi úspěšně prosadili dávno neplatí, jsou, a v dalších letech pravděpodobně stále budou, zvláště u středních a malých webových aplikací naprostým standardem. Z tohoto důvodu bude v praktické části práce pro tvorbu WebCMS aplikace použita relační databáze. V následující kapitole jsou nejprve vymezeny základní pojmy týkající se databázových systémů obecně. Následně jsou popsány principy relačního modelu jako takového, následují Coddovy pravidla a principy normalizace relačních databází.

Úvod do databázových systémů

Databáze

Databází rozumíme strukturovanou kolekci dat, která se vztahuje k určitému tématu. Počítačové databáze jsou pak databáze, které jsou v digitální podobě uložené v počítači nebo nějakém počítačovém médiu. Tyto databáze mají oproti databázím v „reálném“ světě, jako jsou například různé kartotéky, seznamy či archivy, značné výhody. Digitální databáze může obsahovat obrovské množství různých údajů, včetně vazeb mezi nimi při minimálních uskladňovacích nárocích. Také vyhledávání v počítačové databázi a různé třídění či filtrování údajů je v databázi na počítači mnohem snazší a rychlejší než v běžné kartotéce či archivu, neboť většinu práce odvádí za uživatele právě počítač. Veškeré

operace s databází jsou prováděny počítačem skrze k tomu určený software. Tento software se stará také o řízení přístupu a o organizování dat ukládaných do databáze podle zvoleného datového modelu [4].

Datové modely

Datový model je abstraktní model, podle kterého mohou být data strukturována. Pojem „datový model“ má v českém jazyce dva možné významy. Za prvé můžeme tímto pojmem označovat konkrétní datový model připravený k praktickému využití (*data model instance*). Za druhé tímto pojmem můžeme myslet datový model ve smyslu teorie datových modelů (*data model theory*), tj. soubor pravidel, podle kterých se konkrétní datové modely tvoří.

Datových (databázových) modelů jako formálních popisů struktury a práce s daty je celá řada. Dnes jsou nejpoužívanější relační a objektový datový model. Pro svou jednoduchost a rychlost se ale někde stále používají i starší datové modely, jako je například model síťový. Většina databázového softwaru umožňuje pracovat s jedním, někdy však i s více datovými modely. Například jeden z dnes nejrozšířenějších vývojářů DBMS relačních databází Oracle dnes poskytuje i objektovou nadstavbu k běžným relačním nástrojům [5].

Mezi historicky nejstarší databázové modely patří databázový model hierarchický a databázový model síťový. Oba vznikly v šedesátých letech minulého století. Hierarchický model byl poprvé použit firmou IBM v jejich IMS (*Information Management System*). Podle tohoto modelu jsou data organizována do stromové struktury. Základem je předpoklad, že potomek má pouze jednoho rodiče, tj má pouze jednu vazbu směrem vzhůru. A dále předpoklad, že model obsahuje pole, která určují pořadí záznamů v příslušných listech. Síťový datový model byl ve stejnou dobu nezávisle vyvinut sdružením CODASYL a použit v IDMS (*Integrated Database Management System*). Síťový model umožňuje data propojit pomocí ukazatelů. Tím umožňuje oproti hierarchickému modelu, aby dítě mělo více rodičů [4].

V roce 1970 navrhl britský vědec pracující pro firmu IBM, Edgar Fred Codd, relační datový model. Trvalo však dalších deset let, než se tento model dočkal komerčního využití a byl použit v programech Oracle a DB2 (kolem roku 1980). Základní datová struktura relačního modelu se dá popsat jako tabulka obsahující data o určité entitě. Sloupce v tabulce představují různé atributy dané entity a řádky pak vlastní instance entity. Vlastní

databázi pak tvoří řada tabulek, které mezi mají definované vztahy. Tím pádem relační model, na rozdíl od dřívějších modelů, má vždy jasně stanovenou jak strukturu dat, tak i množinu veškerých možných operací, které se nad nimi mohou provádět [4].

V 90. letech se pak začaly rozvíjet objektově orientované databázové modely. V objektově orientované databázi je informace reprezentována ve formě objektů. Vazby mezi těmito objekty nejsou na rozdíl od relačních databází realizovány pouze pomocí cizího klíče, ale jsou přímo provázány. Díky tomu se s daty daleko snáze pracuje. Objektové databáze také přináší možnost snadno pracovat s děděním, agregací a dalšími objektovými nástroji [4].

Database management system

Základní myšlenkou počítačových databázových technologií je oddělení uživatele od vlastních dat v databázi. *Database management system* (DBMS), česky systém řízení báze dat, je specializovaný program, který se stará právě o to. DBMS tedy kontroluje, v jaké podobě, kam a jakým způsobem se budou data do databáze ukládat, i jak se z ní budou zpětně načítat. Existence DBMS jako mezistupně v komunikaci mezi daty a uživatelem přináší řadu výhod. Patří mezi ně především: snížený výskyt redundance a nekonzistence vkládaných dat; nezávislost dat a aplikace a tím snadnou možnost přechodu na novější software bez náročného předělávání dat; snadná správa přístupů jednotlivých uživatelů databáze a s tím související možnost nechat uživatele vidět či upravovat pouze jemu příslušnou část dat [1, 4].

DBMS se běžně dělí na 4 základní části. Za prvé z modelovací jazyk, pomocí kterého lze definovat struktura tvořené databáze. Modelovací jazyk, který DBMS používá, závisí tedy na datovém modelu, se kterým bude DBMS pracovat. DBMS dále obsahuje datové struktury optimalizované pro práci s velkým množstvím dat, a také jazyk pro zprostředkování komunikace mezi databází a uživatelem (*database query language*). Pomocí tohoto jazyka jde nejen zobrazovat a upravovat databáze, ale i nastavovat veškerá práva uživatelů databáze. Pro komunikaci s relačními databázemi je dnes nejpoužívanější jazyk SQL (*structured query language*). Ten obsahuje jak modelovací jazyk, tak i jazyk pro komunikaci s uživatelem. Poslední hlavní částí systému pro řízení báze dat je transakční mechanismus, který se stará o datovou integritu v databázi a o integritu dat upravovaných či do databáze vkládaných.

Datové modelování

Datovým modelováním se rozumí vlastní proces využití dané teorie datových modelů k tvorbě vlastní instance datového modelu. A je to první a velmi důležitý krok v tvorbě databázového systému. V praxi datové modelování představuje snahu o vytvoření organizace a struktury dat. A to do takové podoby, aby potom s těmito daty byla požadovaná práce (vyhledávání, úpravy, atd.) co nejjednodušší a nejrychlejší, pro uživatele přinášela co nejvíce užitené hodnoty a přitom ho zbytečně nezatěžovala. Výsledná instance datového modelu pak může mít jednu ze tří podob [4, 5, 6] (podle ANSI).

- Konceptuální schéma popisuje celkovou logickou strukturu databáze. Obsahuje entity, jejich atributy a vztahy mezi nimi. Je zcela nezávislé na jakémkoli software ve kterém by s mohla databáze tvořit, a způsobu ukládání vlastních dat.
- Logické schéma tvoří mezistupeň mezi fyzickým a konceptuálním schématem. Na rozdíl od konceptuálního schématu obsahuje věci, které souvisejí se zvolenou technologií organizace a úschovy dat, jako jsou například primární a cizí klíče v relačních databázích. Stále však zobrazené schéma není spojeno s žádným určitým DBMS.
- Fyzické schéma na rozdíl od předchozích již bere v úvahu jak software, ve kterém se bude vlastní databáze tvořit, tak i datové struktury, které bude používat

Existují i další typy rozdělení instancí datových modelů, například Zachmanova struktura[1], ale pro splnění cílů této práce nám stačí základní výše popsané rozdělení podle standardu ANSI.

Při navrhování relační databáze je běžné, že před tvorbou databáze vždy připravíme a odladíme konceptuální, případně rovnou logické schéma. Následně z něj tvoříme schéma fyzické a teprve pak vlastní databázi. Dobře promyšlené logické schéma později zamezí řadě problémů s výskytem redundantních dat a údržbou databáze. Může také pak umožnit snadnější a při velkém množství záznamů i výrazně rychlejší formulaci dotazů na databázi.

[1] Více např. zde: http://www.enterpriseunifiedprocess.com/essays/zachmanFramework.html.

Entity-relationship diagram a Class diagram

Při relačním modelování zobrazujeme konceptuálně způsob strukturování dat buď pomocí standardního Chenova *Entity-Relationship* diagramu, nebo pomocí novějšího UML Class diagramu. Běžně se dnes můžeme setkat při návrhu relačních databází jak s Chenovým ER diagramem, tak s class diagramem, či nějakou z jeho variant. Proto jsou zde zmíněny oba. Class diagram navíc umožňuje oproti staršímu Chenovu diagramu zaznamenat řadu vlastností objektového přístupu, které však při navrhování relační databáze nelze použít. Class diagram také bude použit pro popis struktury vlastního návrhu *web content management* systému zrealizovaném v PHP. V zásadě si pro účel návrhu relační databáze oba diagramy v tom co vyjadřují odpovídají akorát mají trochu jinou konvenci zápisu.

Při tvoření ER diagramu budoucí databáze tedy znázorňujeme struktury tabulek, ve kterých se budou data později uchovávat a vazby mezi nimi. Mezi hlavní prvky v ER modelování patří entita, vazba a atribut [6, 7, 8, 9].

- Entitou rozumíme nějaký objekt v reálném světě, který lze jasným způsobem rozeznat od ostatních [4]. V ER diagramu se ve skutečnosti nezobrazují entity a jejich vazby, ale skupiny entit a skupiny všech možných vazeb mezi těmito entitami. Pro jednoduchost však budeme mluvit jen o entitách a vazbách. Entitní skupinu je v relační databázi možno přirovnat k tabulce a jednotlivé entity jako řádky tabulky.V ER diagramu se entita zobrazuje obdélníkem s názvem entity. Název je většinou podstatné jméno.
- Vazba mezi entitami se zobrazuje spojením entit čarou s kosočtvercem obsahujícím popisek vazby uprostřed. Popisek vazby je většinou sloveso. U vazeb se v ER diagramu často značí i jejich kardinalita (násobnost vazeb). Kardinalitu značíme většinou pomocí popisku u vazby nebo některou z alternativních konvencí, jako je třeba konvence *Crow's feet*. Při tvorbě webových stránek se většinou setkáme s kardinalitou typu „0:n“, „1:n“ nebo „n:m“. V relační tabulce se vazby mezi tabulkami řeší pomocí cizích klíčů.
- Atribut se zobrazuje elipsou s názvem atributu. Může být propojen jak s entitou, tak s vazbou. V relační tabulce odpovídá atribut sloupci tabulky. Primární klíč, pokud se zobrazuje, se odlišuje pomocí podtržení názvu.

V ER diagramu se nezobrazují cizí klíče a asociační tabulky.

UML neboli unified modeling language je standardizovaný univerzální jazyk na modelování při objektovém návrhu softwaru. Class diagram se v rámci UML používá podobně jako Chenův ER diagram, tj při popisu statické struktury systému. Oproti Chenovu ER diagramu však umožňuje znázornit všechny vymoženosti objektově orientovaného modelování, jako je generalizace, agregace aj..

Základní prvek Class diagramu je třída. Ta je znázorněná obdélníkem rozděleným na tři části. Ve vrchní částí je název třídy, pod ním jsou atributy třídy a v poslední části metody či operace třídy. Třídy se stejně jako v chenově diagramu spojují vazbami s danou kardinalitou. Případně se symbolem určujícím, o jaký typ vazby se jedná[2].

Relační databáze

Relační databáze je databáze vytvořená a řídí se podle pravidel relačního datového modelu. Základní pojmy relační databáze jsou relace, tuple a atribut. Místo nich se ale často používá terminologie známý z jazyka SQL [6, 9, 10].

- Relace je skupina *tuple*, která má stejné atributy. V relačních databázích se k veškerým datům přistupuje skrz relace. Relace, které skutečně obsahují data, se nazývají základní (*base relation*). Relace, které data přebírají od jinud a pouze je mění pomocí relačních operací, se nazývají derivované (*derived relation*). V SQL se základní relace nazývá tabulka (*table*) a derivovaná relace se nazývá pohled (*view*).
- *Tuple* (česky se tento výraz dá přeložit jako n-tice) je neseřazený souhrn hodnot, přičemž typ sledovaných hodnot je přesně určen atributy relace. V SQL *tuple* odpovídá řádce (*row*).
- Atribut je soubor dat přesně určeného typu. V SQL odpovídá atribut sloupci.

---

[2] Více o class diagramu a jeho tvorbě se lze dozvědět nap například zde: http://www.developer.com/design/article.php/2206791/The-UML-Class-Diagram-Part-1.htm (naposledy navštíveno 20.3.2011).

Protože v SQL bude v druhé části práce realizován návrh *web content management* systému, budou se místo klasických pojmů z relační terminologie v dalším textu pro zjednodušení používat pojmy, které se používají v SQL.

Principy relačního modelu a relační databáze

Relační model se dá charakterizovat následujícím [1, 6, 9, 11]:

- Relace se dá zobrazit jako dvojrozměrná tabulka s atributy jako jejími sloupci a záznamy (*tuple*) jako jejími řádky. Průsečík řádků a sloupců se označuje jako buňka nebo pole tabulky (*field*).
- Každý sloupec tabulky musí mít jasně dané jméno. Jméno sloupce v tabulce musí být unikátní pro danou tabulku. Hodnoty v každém sloupci musí být homogenní, tj. musí mít být stejného typu (např. název, barva, datum vytvoření, atd). Každý sloupec musí mít stanovenou doménu hodnot, které může obsahovat. Nesmí záležet na pořadí sloupců v tabulce.
- Hodnoty každého řádku se musejí vztahovat k jedné věci nebo její části. Nesmí záležet na pořadí řádků. V tabulce nesmí být násobné řádky, tj. 2 řádky se stejnými hodnotami.
- Veškeré údaje v tabulce musí mít pouze jednu hodnotu (musí být atomické).
- Veškerá práce s daty probíhá skrze relační nástroje. K práci s tabulkami se dá používat operací výrokové logiky a relační algebry (viz dále).

Tyto základní charakteristiky by měla mít každá relační tabulka. Nicméně dnešní DBMS a jazyky, které tyto DBMS používají, často aby umožnily uživateli snazší práci povolují prohřešky proti těmto pravidlům. Například dnes nejrozšířenější jazyk pro práci s relačními databázemi SQL povoluje mít v jedné tabulce dva totožné řádky.

V relační databázi lze sloupci v tabulce přiřadit řadu funkcí, které ovlivňují, jak se pak s daty, která obsahuje, pracuje. Některé jsou již součástí relačního datového modelu (např. klíče). Jiné vznikly z praktických důvodů při jeho implementaci (index). Mezi základní funkce patří:

- Klíče (*keys*) – Klíče představují omezení, která zabraňují duplicitním záznamům. V každé tabulce by měl být nastaven alespoň jeden sloupec označený jako primární klíč (*primary key*). Primární klíč zaručuje, že v tabulce nebudou existovat násobné řádky. Klíčů lze nastavit libovolné množství, a to i přes více atributů. Takové klíče se nazývají složené klíče (*compound keys*).
- Cizí klíče (*foreign keys*) – Přestože název říká, že se jedná o klíče, nezapadají cizí klíče do definice z předchozího bodu. Definují sloupce, které pak obsahují hodnoty z jiné tabulky, a vytvářejí tak mezi nimi spojení. V praxi to většinou znamená, že obsahují primární klíče z jiné tabulky. V případě kardinality „m:n“ se tabulky spojují pomocí zvláštní (třetí) tabulky, která pak jako primární klíč používá vlastně dva klíče cizí. Tato tabulka se nazývá asociační.
- Indexy (*indexes*) slouží ke zrychlení přístupu k datům. Fungují na stejném principu jako rejstřík na konci knihy. Lze je vytvořit jak na jednom, tak i na kombinaci několika atributů. Nejsou vlastně součástí databáze ani nejsou součástí relačního modelu. Nicméně správně nastavené indexy mohou několikanásobně zrychlit operace s databází.

12 Coddových pravidel

Je to soubor 13 pravidel seřazených od 0 do 12, které navrh Edgar F. Codd, stvořitel relačního modelu. Tato pravidla by podle něj měl dodržovat DBMS, aby byl považován za relační. Ani u dnešních DBMS však nejsou všechna tato pravidla pro jejich tvrdost a náročnost většinou bezezbytku dodržována [1, 12].

- Pravidlo 0: Systém se musí kvalifikovat jako relační systém řízení báze dat. Musí tedy spravovat data pouze pomocí relačních operací.
- Pravidlo 1: Informační pravidlo – všechny informace v databázi musí být reprezentovány pouze pomocí hodnoty v tabulce. Tedy jako hodnota v buňce, která je tvořená sloupcem a řádkem.
- Pravidlo 2: Pravidlo zajišťující přístup – všechna data musí být přístupná vždy jedním způsobem, a to tak, že ke každému údaji se lze logicky dostat specifikací jména tabulky, jména sloupce a primárního klíče určující řádek.

- Pravidlo 3: Systematické zacházení s prázdnými hodnotami – DBMS musí dovolit každé buňce, aby zůstala prázdná, tzn. musí podporovat reprezentaci chybějící informace, která je odlišná od všech ostatních výrazů.
- Pravidlo 4: Systém katalogu založeném na relačním modelu – DBMS musí podporovat katalog struktury tabulek, ke kterému lze přistupovat pomocí stejného jazyka, jako se přistupuje k ostatním datům v databázi.
- Pravidlo 5: Pravidlo srozumitelného datového jazyka – DBMS musí obsahovat alespoň jeden relační jazyk, který:
    - má lineární syntaxi
    - může být použit jak interaktivně, tak ve skriptech a aplikacích
    - obsahuje:
        - Jazyk pro definici dat (*data definition language*) – Obsahuje příkazy pro změny a tvorbu struktury tabulek. Obsahuje příkazy na tvorbu pohledů.
        - Jazyk pro manipulaci s daty (*data manipulation language*) – Pomocí něj se dají do tabulek vkládát nové záznamy. Následně se s jeho pomocí dají tyto záznamy měnit a udržovat. A samozřejmě vyhledávat a vracet uživatelům podle zadaných kritérií.
        - Jazyk pro určení integritních omezení
        - Jazyk pro práci s transakcemi – obsahuje například příkaz *rollback.*
        - Jazyk pro určení omezení práv přístupu – pomocí něj se nastavují práva jednotlivců i skupin uživatelů přístupovat k databázi.
- Pravidlo 6: Pravidlo práce s pohledy – DBMS musí umožňovat práci s pohledy, a to včetně jejich aktualizace.
- Pravidlo 7: Vysokoúrovňové vkládání, mazání a upravování dat – DBMS musí podporovat možnost upravit, vložit a smazat více řádků podle udaných podmínek, a to i při práci s více tabulkami.

- Pravidlo 8: Pravidlo nezávislosti fyzických dat – změny na fyzické úrovni dat (jak jsou data uložena, jestli v polích, spojových seznamech, atd) nesmí vyžadovat změnu aplikace postavené na struktuře tabulky.
- Pravidlo 9: Pravidlo nezávislosti logických dat – změny na logické úrovni (tabulky, sloupce, řádky, atd) nesmí vyžadovat změnu aplikace založené na struktuře dat.
- Pravidlo 10: Pravidlo nezávislosti integritních omezení – integritní omezení musí být tvořena odděleně od aplikací a musí být uložena v katalogu. Mělo by být možno změnit omezení bez ovlivnění existujících aplikací.
- Pravidlo 11: Pravidlo nezávislosti distribucí – i když je databáze rozdělená fyzicky na více částí, nemělo by to být pro uživatele vidět a nemělo by to ovlivnit výsledky vyhledávání či jiné práce s databází.
- Pravidlo 12: Pravidlo nenarušitelnosti databázového systému – žádný interface připojený k databázi nesmí narušovat pravidla databázového systému (např. porušovat integritní omezení).

Relační operace

Relační algebra představuje formální popis práce s relačními databázemi. Pomocí ní se přistupuje k datům v databázi. Takže vlastně tvoří základ, na kterém je postaven jazyk SQL, se kterým budeme pracovat. Nejběžnější operace relační algebry, se kterými se setkáme, jsou [6]:

- Projekce (*projection*) – slouží k výběru sloupců z tabulky. Standardně při dotazu na relační tabulku musíme specifikovat, jaké sloupce chceme, aby nám byly vráceny. V další části používaný jazyk SQL se odchyluje od striktního relačního modelu možností vybrat všechny sloupce v tabulce pomocí zástupného znaku „*“. Na jednu stranu je to pohodlné, ale častokrát si necháváme vrátit sloupce s nepotřebnými údaji a zvyšujeme tím náročnost operace.
- Selekce (*selection*) – slouží k výběru řádků podle uživatelem specifikovaných podmínek.
- Spojení (*join*) – slouží ke spojení více tabulek primárně pomocí kartézského součinu. Existují však i další typy spojení jako right join a left join, které umožňují

napojit tabulku na druhou pouze jednostraně. Jakékoli spojení lze samozřejmě omezit podmínkou.

Normalizace

Datová normalizace byla navržena E. F. Coddem jako součást relačního datového modelu. Jedná se o algoritmus pro eliminování nebezpečí redundance dat v navrhovaném modelu. Tím se také předchází různým anomáliím při zadávání, úpravě a mazání dat.

V současné době se v relačních modelech používá 6 normálních forem, přičemž první a druhá normální forma jsou považovány za nejdůležitější a měla by je v zásadě splňovat každá relační databáze. Platí, že pokud databáze splňuje normální formu n-tého stupně, tak splňuje i všechny normální formy nižších stupňů.

- První normální forma: Vyřadit opakující se skupiny dat – tj. pokud jsou v tabulce sloupce s násobnými daty nebo více sloupců obsahujících data stejného druhu, je třeba pro ně udělat zvláštní tabulku. Pokud by tato forma nebyla splněna, nejednalo by se vlastně ani o relační tabulku, protože předpoklad atomicity obsahu buňky je dán již pravidly relačního modelu.
- Druhá normální forma: Relační tabulka musí být v první normální formě a všechny neklíčové atributy relace musí funkčně záviset na celém primárním klíči, ne pouze na jeho části. Tato normální forma se týká pouze tabulek se složeným klíčem. V případě, že atribut závisí pouze na části klíče, přesuneme ho do zvláštní tabulky.
- Třetí normální forma: Relační tabulka musí být ve druhé normální formě a každý neklíčový atribut musí být závislý přímo na všech klíčích. To znamená, že každý sloupec se musí přímo podílet na popisování entity jednoznačně označené klíčem, jinak musí být přeřazen do jiné tabulky.
- Boyce-Coddova normální forma (BCNF): Doplňuje třetí normální formu a říká, že v rámci složeného klíče by neměly být netriviální funkční závislosti mezi jeho atributy. Jinak by měly být atributy, kterých se to týká, rozděleny do zvláštních tabulek. Případ, že by tabulka splňovala třetí normální formu a nesplňovala Boyce-Coddovu normální formu, je velmi řídký.

- Čtvrtá normální forma: Relační tabulka musí být v BCNF a nesmí obsahovat 2 a více „1:n“ nebo „n:m“ vazeb, které nejsou na sobě přímo závislé. Tato normální forma se týká především asociativních tabulek při „m:n“ vazbách.
- Pátá normální forma – Relační tabulka musí být ve čtvrté normální formě a všechny dvojice vazeb na další tabulky musí být vždy unikátní.

Domény atributů

Pojem doména atributu vyjadřuje veškeré možné hodnoty, které může jakákoli hodnota daného atributu dosahovat. V praxi se toho dosahuje nastavením charakteristik daného atributu. Mezi běžně nastavované charakteristiky patří nastavení počtu znaků, možnosti použití nulové hodnoty a nastavení datového typu.

Datový typ určuje, jak se obecně bude se sloupcem zacházet, jaká data se dají do sloupce vložit, jaká je výchozí hodnota a maximální počet znaků (omezení stanovená druhem datového typu mají prioritu vůči dalším nastavením). Protože v další části se bude pracovat s DBMS MySQL, uvádí se zde konkrétní datové typy používané v MySQL. Základní uvedené datové typy se však většinou používají v každém běžném relačním DBMS [15, 16]. Číselné typy se dají nastavit buď se znaménkem (*signed* – pro nulu, kladná i záporná čísla) nebo bez znaménka (*unsigned* – pro nulu a kladná čísla).

- int – Zkratka pro celé číslo (integer). Používá se pro běžně velká celá čísla. Má rozsah od -2^31 až 2^31 - 1 nebo 0 až 2^32 - 1.
- tinyint – Pro ušetření místa v paměti je možno použít některý z datových typů odvozených od int. Jejich jediný rozdíl je odlišnost ve velikosti čísla, které mohou uchovat, a tak i v paměti, kterou zabírají. Běžně se používají tinyint (-128 až 127, nebo 0 až 256), smallint (-32768 až 32767 nebo 0 až 65535), mediumint (-2^23 až 2^23 -1, nebo 0 až 2^24 - 1) a bigint (-2^63 až 2^63 – 1, nebo 0 až 2^64 - 1)
- float – Float se používá pro zachycení desetinných čísel. Pro desetinná čísla velkých rozměrů pak jde použít datový typ double, který povoluje větší přesnost.
- char – Char se používá pro zachycení krátkých řetězců znaků (0 až 256 znaků).
- varchar – Varchar se také používá na zachycení krátkých řetězců, ale na rozdíl od charu šetří místo tím, že zabírá vždy pouze místo podle délky řetězce, který je do

ného vložen (char zabírá vždy stejně paměti, podle nastavené hodnoty, i kdyby byl celý prázdný). Nevýhodou varcharu je, že je pro DBMS obtížněji zpracovatelný než char, který má pevně danou velikost. Od MySQL verze 5.0.3 se možná délka varcharu zvětšila na 65535 znaků. V dřívějších verzích to bylo 256 jako u char pouze.

- text – Text se používá pro dlouhé řetězce znaků až 2^16. Má několik variant měnících se podle délky textu, kterou umožňuje uchovat (longtext umožňuje až 2^32 znaků). Stejně jako varchar, pole datového typu text zabírá pouze prostor, který zabrala hodnota do něj vložená.
- set – Set umožňuje nastavit přesně dané řetězce, které půjdou do sloupce vložit.
- date – Datový typ date umožňuje vložit datum ve formátu RRRR-MM-DD. Pro uložení času můžeme použít datový typ time (pouze čas ve formátu HH:MM:SS), nebo datetime (pro kombinaci data a času).

Mezi upřesňující omezení domény atributu pak patří nastavení možného rozsahu (horní a dolní meze v případě číselné hodnoty), nebo stanovení přípustných hodnot (většinou v případě slov), jedinečnosti.

Základy SQL

SQL a MySQL

SQL (*structured query language*) je počítačový databázový jazyk, který se používá pro práci s relačními databázemi. Původně byl navržen Donaldem D. Chamerlinem a Raymondem F. Boycem začátkem 70. let minulého století. Nyní je SQL součástí jak mezinárodního standardu ISO, tak i amerického ANSI. Obsahuje příkazy pro tvorbu a úpravu struktury tabulek, i příkazy pro vkládání, úpravu a vyhledávání dat a i příkazy pro řízení přístupů k databázi. Dnes je SQL součástí prakticky všech relačních DBMS, jako jsou například Oracle, MySQL a PostgreSQL. Základní syntaxe SQL je ve všech DBMS, které ho používají, stejná. Liší se pouze pár názvech složitějších funkcí a dalších drobnostech. Tato práce se bude zabývat verzí SQL, obsaženou v produktu MySQL od Sun Microsystems. Je to z důvodu, že většina pronajímatelů webového prostoru v České republice dává spolu s místem pro webovou stránku databázi MySQL zdarma k použití. A

také, že MySQL lze bezplatně stáhnout, nainstalovat a používat pod licencí GNU (*general public licence*). Na většině serverů lze k databázi MySQL přistupovat pomocí webového rozhraní [3]. Pokud je MySQL nainstalován přímo na osobním počítači, lze k němu přistupovat pomocí nainstalovaného klienta, nejjednodušeji ve formě příkazového řádku. Tato práce se dále zaměří jen na příkazy MySQL používané pro vytvoření tabulek použitých v druhé části této práce. A samozřejmě příkazů pro následnou práci s těmito tabulkami[4].

Základy práce s SQL

V této práci se tedy předpokládá, že uživatel pracuje s nějakou již vytvořenou databází. Pokud tomu tak není, je třeba jí vytvořit pomocí příkazu CREATE DATABASE jméno_databáze; a následně ji vybrat pro užívání pomocí příkazu USE jméno_databáze [17].

Při vysvětlení syntaxe SQL bude v dalším textu používáno označení nepovinných částí příkazů do hranatých závorek.

Mezi obecné věci, které je při používaní MySQL k tvorbě relační databáze dobré vědět patří:

- Každý příkaz se ukončuje středníkem, ale při přistupování k MySQL při psaní skriptů v PHP se středník psát nemusí.
- I když by každá relační databáze měla umožňovat práci s transakcemi MySQL s v základu nastavným modelem správy ukládání dat MyISAM to neumí. Pro práci s transakcemi je třeba pro ukládání tabulky použít systém ukládání dat InnoDB.
- Slova která jsou součástí jazyku SQL je vhodné psát velkými písmeny. Je to proto, aby byly tyto slova snadno oddělitelná od názvů sloupců a dalších výrazů a dalo se tak lépe v dotazech vyznat. Není to vyžadováno, ale je to běžná konvence.

---

[3] Nejrozšířenější je asi phpMyAdmin (http://www.phpmyadmin.net/), který umožňuje prakticky kompletní administraci přes internet.

[4] Pro nainstalování a nastavení MySQL či příkazy pracující s právy uživatelů vysvětluje většina literatury [10, 16, 17] s MySQL spojená a je k dispozici i řada návodů na internetu [15].

- Datum se vždy vkládá do databáze ve formátu RRRR-MM-DD.
- V příkazech se konstanty uzavírají do apostrofů ' ' či uvozovek. Je třeba nezaměňovat konstanty se jmény tabulek a sloupců v databázi – ty se často dávají do zpětných apostrofů (``). Používání zpětných apostrofů není vyžadováno, ale zamezí se tím problémům při pojmenování tabulek a sloupců slovy, které mají v SQL zvláštní význam.
- V příkazech se dají používat aritmetické operátory * - krát, / - děleno, % - zbytek po dělení, + - plus, - - mínus. A jejich priorita je přesně jak jsme zvyklí z matematiky. Dá se samozřejmě upravovat kulatými závorkami.
- V příkazech se dají používat porovnávací operátory <, <=, = (nebo <=>), != (nebo <>), >=, >; dále výrazy [NOT] BETWEEN min AND max pro určení, zda je hodnota v daném rozsahu (x=between min and max je ekvivalentní s (min<=x and x>=max)); dále výraz [NOT] IN (hodnota1, hodnota2,…) pro porovnání s přesně danými hodnotami.[15, 17].
- V příkazech se mohou použít logické operátory ! (nebo NOT), AND (nebo &&), OR (nebo ||) a XOR.
- V příkazech se dá použít operátorů pro hledání shody se vzorkem. To lze provézt úomocí operátoru LIKE. Ten porovnává podle zadaného řetězce, do kterého je možno zakomponovat zástupné znaky „%“ (místo libovolné skupiny znaků, včetně prázdného řetězce) a „_“ (místo jednoho libovolného znaku); a za druhé je možné pomocí operátoru REGEXP zjišťovat, zda se ve výrazu vyskytuje regulerní řetězec (viz dodatky).

Následující příkazy jsou pro lepší čitelnost strukturovány do několika řádků, ale funkčně to nemá žádný význam.

Vytvoření struktury tabulky a práce s ní

Vytvořit novou tabulku se dá pomocí příkazu CREATE TABLE. Syntaxe tohoto příkazu, obsahující nejběžněji používané specifikace sloupce, je následující (méně používaná nastavení tabulky zde nejsou pro zjednodušení uvedena) [15]:

```
CREATE TABLE
```

```
    `jméno_tabulky` (
    `jméno_sloupce1` datový_typ_sloupce1[počet znaků] [NOT
NULL] [UNSIGNED],
    `jméno_sloupce2` datový_typ_sloupce2 [počet znaků]
[NOT NULL] [UNSIGNED],
    …,
    PRIMARY KEY (`jméno_sloupce1`, …)
    INDEX (`jméno_sloupce1`, …)
);
```

Pro zrušení tabulky se používá příkaz DROP TABLE se syntaxí:

```
DROP TABLE
`jméno_tabulky`;
```

Po vytvoření tabulky můžeme měnit její strukturu pomocí příkazu ALTER TABLE. Má následující strukturu:

```
ALTER TABLE `jméno_tabulky` akce1, akce2,…;
```

Nejběžnější akce proveditelné pomocí příkazu ALTER TABLE jsou přidání, odebrání, nebo změna parametrů sloupce a samozřejmě přejmenování celé tabulky. Méně používané akce (jako změna primárního klíče, předefinování indexů, nebo definice dalších klíčů) zde nejsou rozpracovány.

Přidání sloupců do již vytvořené tabulky:

```
ALTER TABLE `jméno_tabulky`
    ADD COLUMN (
    `jméno_sloupce1` datový_typ_sloupce1[počet znaků] [NOT
    NULL] [UNSIGNED],
    `jméno_sloupce2` datový_typ_sloupce2 [počet znaků]
    [NOT NULL] [UNSIGNED],
    …,
    );
```

Odebrání sloupce z tabulky:

```
ALTER TABLE `jméno_tabulky`
```

```
    DROP COLUMN `jméno_sloupce`;
```

Změna atributů a názvu sloupce (pro změnění pouze atributů se většinou používá akce MODIFY, která má skoro stejnou syntaxi, jen se neuvádí nové jméno):

```
ALTER TABLE `jméno_tabulky`
    CHANGE COLUMN `jméno_sloupce`
    `nové_jméno_sloupce` datový_typ_sloupce[počet znaků]
[NOT NULL] [UNSIGNED];
```

Vkládání a úprava dat v databázi

Data do databáze se vkládají přes příkaz INSERT. Může se použít jedna následujících tří syntaxí. První se většinou používá na vložení všech hodnot řádku, neboť se nemusí specifikovat sloupce. Mohou se jen vypsat hodnoty a ty se pak vloží podle pořadí sloupců v tabulce. Druhá syntaxe se používá většinou tehdy, když se vyplňují pouze některé sloupce, protože v takovém případě je třeba vždy specifikovat, do jakého sloupce se má hodnota vložit. Toho se dá samozřejmě dosáhnout i prvním zápisem, pokud se sloupce specifikují. Záleží jen na vkusu uživatele. Třetí syntaxe se používá pro zkopírování řádků z jiné tabulky vloženým příkazem SELECT (viz dále). V tomto případě je třeba, aby SELECT vracel sloupce pojmenované stejně, jako sloupce v tabulce, do které je vkládá [15, 17, 18].

```
INSERT INTO `jméno_tabulky`
[(`jméno_sloupce1`
    ,`jméno_sloupce2`
    ,…)]
    VALUES (
          'hodnota_sloupce1',
          'hodnota_sloupce2',
          …
    );
```

nebo

```
INSERT INTO `jméno_tabulky`
SET
```

```
    `jméno_sloupce1`='hodnota_sloupce1',
    `jméno_sloupce2`='hodnota_sloupce2',
    ,…;
```

nebo

```
INSERT INTO `jméno_tabulky`
[(`jméno_sloupce1`,
    `jméno_sloupce2
    ,…)]
SELECT …
);
```

Pro úpravu dat již vložených do databáze se používá příkaz UPDATE. Jeho základní syntaxe je následující:

```
UPDATE `jméno_tabulky`
    SET
        `jméno_sloupce1`='hodnota_sloupce1',
        `jméno_sloupce2`='hodnota_sloupce2',
        …
    [WHERE podmínka]
    [ORDER BY
        `jméno_sloupce1` [ASC nebo DESC],
        `jméno_sloupce2` [ASC nebo DESC]
        ,…]
    [LIMIT
        [počet_přeskočených_řádků, ]
        počet_vrácených_řádků];
```

Příkaz UPDATE tedy postupně projde všechny řádky, které odpovídají podmínkám stanoveným v parametru WHERE a přiřadí hodnoty daným sloupcům podle parametru SET. Pokud nebude dán parametr WHERE, provedou se změny v celé tabulce. Parametry WHERE, ORDER BY a LIMIT budou popsány u příkazu SELECT, kde se používají

mnohem častěji. Následující příklad příkazu UPDATE projde tabulku menu a menu, jehož „parent“ se rovná 5, změní v sloupci „lang“ hodnotu na cs.

```
UPDATE `menu`
     SET `lang`='cs'
     WHERE `parent`='5';
```

Výběr dat z databáze

Pro výběr dat z databáze se používá příkaz SELECT. Pomocí tohoto příkazu se dá přesně definovat množina řádků, kterou nám má databáze vrátit. Podle náročnosti požadavků uživatele může být tento příkaz značně propracovaný a složitý. Syntaxe obsahující běžně používané části příkazu SELECT se dá napsat takto:

```
SELECT [DISTINCT] seznam_vrácených_sloupců
     [FROM seznam_tabulek_a_typů_spojení]
     [WHERE podmínka]
     [GROUP BY
           `jméno_sloupce1`,
           `jméno_sloupce2`,
           …]
     [HAVING podmínka]
     [ORDER BY
           `jméno_sloupce1` [ASC nebo DESC],
           `jméno_sloupce2` [ASC nebo DESC],
           …]
     [LIMIT
           [počet_přeskočených_řádků, ]
           počet_vrácených_řádků];
```

Pro příkaz SELECT je povinný výčet sloupců, který se má vrátit. Běžně se žádá pouze o ty sloupce z tabulek, které jsou zrovna zapotřebí. V tom případě se napíší jména sloupců za sebe, oddělené čárkou. V případě práce s více tabulkami se přistupuje ke sloupcům pomocí tečkové notace.

```
SELECT
    `jméno_tabulky1`.`jméno_sloupce1`,
    `jméno_tabulky1`.`jméno_sloupce2`,
    `jméno_tabulky2`.`jméno_sloupce1`,
    …
    FROM `jméno_tabulky1`, `jméno_tabulky2`;
```

Jméno sloupce ve spojení se jménem tabulky se uvádí proto, aby se předešlo chybám v případě, kdy v obou tabulkách bude sloupec stejného jména. V případě, že chceme vybrat všechny sloupce z tabulky, můžeme místo jejich vyjmenování použít zástupný znak „*“. Vraceným sloupcům lze přiřadit libovolné jméno, a to pomocí klauzule „AS“.

```
SELECT
    `jméno_sloupce` as 'Název sloupce'
    FROM `jméno_tabulky2`;
```

SQL umí zobrazit také sloupce, které se v žádných tabulkách nevyskytují. A to buď transformací existujících sloupců, či vygenerováním podle zadaných funkcí či dat. K pracování s daty a vytváření odvozených sloupců má MySQL řadu funkcí. Mezi takové nejvíce používané funkce patří funkce COUNT, SUM, MIN, MAX a AVG. Funkce COUNT počítá výskyty v tabulce a vrátí jejich počet. Buď se požívá pro získání počtu řádků, které mají vyplněnou hodnotu v nějakých určených sloupcích (seznam sloupců jde nahradit zástupným znakem „*“ pro všechny sloupce):

```
SELECT COUNT(`jméno_sloupce1`, `jméno_sloupce1`)
    FROM `jméno_tabulky`;
```

Nebo v kombinaci s funkcí DISTINCT, která zaručí, že každý řádek bude započítán vždy pouze při prvním výskytu nové hodnoty z vybraného sloupce.

```
SELECT COUNT(
    DISTINCT
        `jméno_sloupce1`,
        `jméno_sloupce2`,
        …
```

```
)
FROM `jméno_tabulky`;
```

Funkce SUM vrací součet všech položek daného sloupce. Funkce MAX a MIN vrací největší a nejmenší hodnotu daného sloupce. A funkce AVG vrací aritmetický průměr z hodnot daného sloupce. Všechny tyto funkce mají stejnou syntaxi, jenom se mění název funkce.

```
SELECT
    AVG(`jméno_sloupce`) as 'Průměrná hodnota',
    SUM(`jméno_sloupce`) as 'Součet',
    MIN(`jméno_sloupce`) as 'Minimum',
    MAX(`jméno_sloupce`) as 'Maximum'
    FROM `jméno_tabulky`;
```

Nejběžnější forma příkazu SELECT je s použitím parametrů FROM a WHERE a ORDER BY. Pomocí parametru FROM se nastavují tabulky, ze kterých se budou data vybírat. Standardní spojení tabulek je kartézským součinem. Do SELECTU se zadává jako seznam tabulek ke spojení, kde jednotlivé tabulky jsou od sebe odděleny čárkou. Spojení kartézským součinem většinou vyprodukuje plno nežádoucích řádek, proto se toto spojení obvykle kombinuje s parametrem WHERE.Parametr FORM není povinný. Jde napsat SELECT i bez jeho použití, což se ale většinou nepoužívá. Například následující dotaz vrátí tabulku se sloupcem „vypocet“, která obsahuje jeden řádek s hodnotou 2.

```
SELECT 1+1 AS `vypocet`;
```

Pomocí parametru WHERE lze specifikovat podmínky, podle kterých budou následně vybrány řádky z tabulky. Při specifikaci podmínek lze použít aritmetických, porovnávacích i logických operátorů, popsaných v kapitole Základy práce s SQL. Podmínka má většinou tvar porovnání hodnoty sloupce s jiným sloupcem, nebo porovnání sloupce se zadanou hodnotou.

Nejrozšířenější typ podmínky je zúžení výběru řádků pouze na ty, které obsahují nějakou žádoucí pevně danou hodnotu. Dosahuje se toho tak, že se určitý sloupec porovná podle daného aritmetického nebo porovnávacího operátoru s nějakou zadanou hodnotou. Takových podmínek samozřejmě může být více a mohou být spojeny logickými operátory.

Následující příkaz by například: (1) zobrazil sloupec „name“ z tabulky „menu_lang“; (2) vybral ty řádky, kde hodnota sloupce „name“ začíná na písmeno „k“ nebo na písmeno „p“; a (3) seřadil je podle sloupce „lang“.

```
SELECT `jmeno`, `prijmeni`
        FROM `menu_lang`
        WHERE
              `name` LIKE 'k%' or
              `name` LIKE 'p%'
        ORDER BY `lang`;
```

Klasické využití podmínky, kde se porovnávají mezi sebou dva sloupce, nastává při práci s více tabulkami, a to tam kde se spojením tabulek kartézským součinem vytvoří nová dočasná tabulka s řadou přebytečných hodnot. Klauzulí WHERE se pak vyřadí nežádoucí řádky porovnáním sloupce s primárním klíčem z jedné tabulky se sloupcem z tabulky druhé, přičemž tento sloupce obsahuje instance primárních klíčů z první tabulky – neboli je cizím klíčem.

```
SELECT výčet_sloupců
     FROM `jméno_tabulky1`, `jméno_tabulky2`
     WHERE `jméno_tabulky1`.`primární_klíč_tabulky1` =
`jméno_tabulky2`.`primární_klíč_tabulky1`;
```

Parametr GROUP BY má za úkol umožnit seskupení několika řádek do jedné. Jaké řádky seskupit se určuje pomocí výběru sloupců za tímto parametrem. Parametr GROUP BY seskupí řádky, které mají v těchto sloupcích stejné hodnoty. Parametr HAVING pak omezuje vybrané souhrnné řádky podobně jako parametr WHERE pro celou tabulku.

Parametr ORDER BY určuje, jak budou údaje, které se uživateli vrátí, řazeny. Za ORDER BY vždy následuje seznam sloupců, které jsou doplněny volitelným parametrem DESC (sestupně) nebo ASC (vzestupně), což určuje směr řazení.

Parametr LIMIT určuje, kolik z řádků, které prošly až na konec třídícího procesu, se uživateli vrátí. První číslo určuje, od kolikátého záznamu se počítá, a druhé pak, kolik záznamů se vrací.

PHP

PHP je skriptovací jazyk, který se používá pro psaní webových aplikací běžících na straně serveru. Tento jazyk byl navržen specielně pro psaní dynamických webových stránek a tomu odpovídá i jeho nejčastější využití. Pro bezproblémový běh příkazů, které budou popsané dále, je zapotřebí PHP alespoň ve verzi 4, nejlépe však 5 [19, 20].

Zda server podporuje PHP, případně jaká verze na něm běží lze snadno poznat následovně. V poznámkovém bloku, nebo v libovolném editoru webových stránek se vytvoří nový soubor, pojmenuje se „phpinfo.php“ a vloží se do něj následující skript:

```
<?
phpinfo();
?>
```

Pokud se tento soubor otevře v internetovém prohlížeči, zobrazí informace o verzi PHP běžící na serveru kde se soubor se skriptem nachází. (Pokud se nic nezobrazí, PHP není na tomto serveru nainstalováno.)

Základní charakteristiky práce s PHP

Aby překladač PHP na serveru interpretoval skript uložený v souboru, musí být splněno několik předpokladů. Za prvé musí překladač vědět, že soubor, který server zpracovává, obsahuje nějaký skript PHP. To je zařízeno tím, že koncovky souborů, které mohou obsahovat PHP skript, jsou na serveru vyjmenovány v nastavení. Pokud tedy přijde požadavek na soubor s danou koncovkou, server ho automaticky pošle překladači PHP. Běžně jsou ke zpracování překladačem posílány soubory s koncovkou „php“. Veškerou práci se skriptem provádí překladač; data, která jsou vidět v prohlížeči, jsou pouze ta, která se mu skrz překladač pošlou [20, 21].

I když překladač dostane ke zpracování celý soubor, zpracovává pouze ty jeho části, které jsou označeny jako PHP skript. Označení takových částí se v rámci souboru provádí jejich uzavřením do <?php a ?> nebo jen <? a ?>. Pro vložení komentáře (např. vysvětlivky) do skriptu se používá dvou lomítek – ta způsobí, že cokoli na řádku za nimi bude překladačem ignorováno. Pro komentář přes více řádků se používá syntaxe /* komentář */.

```
<?
```

```
// script PHP
?>
```

V PHP se příkaz vždy ukončuje středníkem. Středník se ale nepoužívá na ukončení komentáře a ani po složených závorkách, například u podmínek nebo cyklů.

Jeden z nejdůležitějších termínů pro jakýkoli programovací či skriptovací jazyk je proměnná. Proměnná jde popsat jako identifikátor, který v sobě skrývá hodnotu, která se může měnit. PHP pracuje s proměnnými celé řady druhů. Základní proměnné platí pouze v rámci hlavního těla a pouze po dobu běhu skriptu. Nemusí se předem definovat, jak je tomu v jiných skriptovacích jazycích. PHP je také na rozdíl od většiny programovacích jazyků velmi ležérní k práci s typy proměnných – v mnoha případech nevyžaduje přetypování proměnné a provede to za programátora samo. Proměnné v PHP se vždy značí znakem dolaru ($), po kterém může následovat libovolná kombinace písmen, čísel a podtržítek – ovšem s tou výjimkou, že po samotném dolarovém znaku musí následovat pouze písmeno nebo podtržítko. Následující skript tedy například vytvoří tři proměnné, poté je sečte do proměnné čtvrté a tu následně vypíše pomocí příkazu echo.

```
<?
$prvni_promena=2;
$druha_promena=4;
$treti_promena=6;
$soucet=$prvni_promena + $druha_promena +$treti_promena;
echo $soucet;
?>
```

V PHP se běžně lze setkat s 5 druhy typů proměnných. Jsou to: *integer* (celé číslo), *double* (desetiné číslo), *string* (znakový řetězec), *array* (pole), *object* (objekt). Typ proměnné se určí ve chvíli, kdy se do ní přiřadí data. Podle typu se s proměnnou pak zachází (např. při řazení několika proměnných).

Integer a *double* obsahují data odpovídající jejich názvům. Jejich hodnoty není třeba uzavírat do uvozovek. Hodnota proměnné typu *string* se zadává buď v uvozovkách nebo apostrofech. Pokud potřebujeme do řetězce zahrnout znak, který má pro PHP speciální význam, je třeba ho označit zpětným lomítkem. Díky zpětnému lomítku pak bude tento

znak překladač ignorovat. Pomocí zpětného lomítka lze do řetězce vložit některé speciální znaky. Stručný výběr znaků, které se dají pomocí zpětného lomítka zařadit do řetězce, je následující:

- \\ - „\“ zpětné lomítko
- \n – nová řádka
- \t – tabulátor
- \" - „"“ uvozovka
- \$ - „$“ znak dolaru

Speciální znaky jdou vložit do řetězce pouze tehdy, pokud ten je definován v uvozovkách. Obsah v apostrofech není s ohledem na tyto speciální znaky kontrolován.

Typ objekt se používá při pokročilejší práci s PHP a nebudeme ho zde popisovat[5]. *Array* neboli pole je datová struktura, která umožňuje jedné proměnné typu pole obsahovat více hodnot. K těmto hodnotám pak lze přistupovat pomocí klíče (indexu). Klíč prvku pole se zapisuje do hranatých závorek za proměnnou. Pokud se nespecifikuje jméno klíče, kterému hodnotu chceme přiřadit, přidělí se hodnotě první volný číselný klíč počínaje číslem 0. Následující ukázkový skript tedy například vytvoří dvě pole. Jedno obsahující hodnoty s klíči 0 1 2 a druhé s hodnotami pod klíči „jmeno“ a „prijmeni“. A následně pomocí příkazu echo vypíše obsah druhého klíče prvního pole.

```
<?
$pole1[]="hodnota1";
$pole1[]="hodnota2";
$pole1[]="hodnota3";
$pole2["jmeno"]="Jan";
$pole2["prijmeni"]="Novák";
echo $pole1[1];
?>
```

[5] Přesnější definice typů proměnných lze najít v prakticky každé literatuře popisující základy PHP [18, 19, 20, 21].

PHP umožňuje práci s klasickými matematickými operátory plus (+), mínus („-“), krát (*), děleno (/) a zbytek po dělení („%“). Priorita operátorů je stejná jako v matematice a dá se samozřejmě měnit závorkami. Pro urychlení operací lze v PHP při přičítání čísla k proměnné použít zkráceného zápisu „+=“, „-=“, „*=“, „/=“. Další způsob zkráceného zápisu je možný při zvětšování či zmenšování čísla o jedničku, a to zápisem „$x++;“ nebo „$x--;“

```
<?
$x=10;
$x+=5;
$x++;
echo $x; // vypíše 16
?>
```

Často používaným operátorem je v PHP operátor pro spojování řetězců. Je to speciální operátor „.“. Ten své operandy převede na řetězce a následně spojí v jeden. Opět lze u něj využít zkráceného zápisu „.=“pro připojení na konec, nebo „=.“ pro připojení na začátek.

Často potřebujeme zkonstruovat výraz, jehož výsledek je buď pravda (TRUE), či nepravda (FALSE). K tomu se v PHP, jako v každém skriptovacím jazyce, používá logických operátorů. Typické operátory pro porovnání dvou výrazů jsou relační operátory:

- $x==$y – proměnná x se rovná proměnné y;
- $x!= $y – proměnná x se nerovná proměnné y;
- $x< $y – proměnná x je menší než proměnná y;
- $x>$y – proměnná x je větší než proměnná y;
- $x<= $y – proměnná x je menší, nebo rovná proměnné y;
- $x>=$y – proměnná x je větší nebo rovná proměnné y.

Jednotlivé logické výrazy lze spojovat dohromady logickými spojkami „a zároveň“ („&&” nebo „and“) a „a nebo“ („||“ nebo „or“). Ke znegování logického výrazu slouží operátor záporu „!“.

Základní funkce PHP

Nejběžnějším využitím výše uvedených logických výrazů je podmínka (IF). Když je podmínka splněna, vykoná se skript, který je uzavřen v závorkách k ní příslušnících [21, 22]. Pokud je logickým výrazem pouze proměnná, je FALSE v případě, že je prázdná nebo je rovna „0“;

Má tuto syntaxi:

```
<?
if (logický_výraz1){
     // vykoná se při splnění logického výrazu 1
     }
else if (logický_výraz2){
     // vykoná se při nesplnění logického výrazu 1
     // a splnění logického výrazu 2
     }
else {
     // vykoná se při nesplnění logického výrazu 1
     // a nesplnění logického výrazu 2
     }
?>
```

Pokud za podmínkou v příkazu „if“ následuje jen jeden příkaz, nemusíme ho uzavírat do složených závorek.

Velmi často používané příkazy jsou v PHP příkazy cyklů. Je jich celá řada a zde jsou uvedeny jen dva nejčastěji používané příkazy, a to příkaz „while“ a příkaz „for“.

Příkaz „while“ je cyklus s na počátku stanovenou podmínkou. Když překladač PHP narazí na tento příkaz, zkontroluje nejprve podmínku. V případě, že platí, začne vykonávat skript, který se nalézá uvnitř příkazu. Po jeho skončení znovu zkontroluje podmínku a v případě, že stále platí, začne skript vykonávat znovu. A tak dál, dokud podmínka nepřestane platit. Zde je třeba si dát pozor, aby se cyklus „while“ nezacyklil, tj. aby podmínka opravdu přestala časem platit.

```
<?
$x=0;
while ($x<2){
echo "$x ";
$x++;
}
// vypíše čísla 0 a 1 oddělená mezerami
?>
```

Příkaz „for“ je poměrně složitý cyklus se syntaxí „for (výraz1; výraz2; výraz3){ tělo cyklu }“. Výraz1 je vyhodnocen již před prováděním cyklu. Výraz2 obsahuje podmínku, která určuje, zda se bude provádět tělo skriptu. A výraz3 se vyhodnotí vždy na konci provádění těla skriptu. Tento cyklus přináší zejména snadnou práci s poli:

```
<?
$pole[0]="Edgar";
$pole[1]="Frank";
$pole[2]="Codd";
for ($i=0; $i<3; $i++)
{
echo $pole[$i]." ";
}
// vypíše „Edgar Frank Codd “
?>
```

Často používaný a hodně užitečný příkaz v PHP je příkaz „include“. Slouží k načtení skriptu z jiného souboru. Načtený obsah souboru se pak chová, jako by byl již v původním souboru. Ve spojení s podmínkami lze tedy načíst část souboru podle zvolené proměnné.

```
<?
include "soubor.php";
?>
```

Práce s MySQL

Jakákoli práce s databází se v PHP vždy musí zahájit připojením k databázovému serveru. Pro systém řízení práce dat je to funkce mysql_connect. Syntaxe této funkce je následující (jméno a heslo nejsou povinné, ale v praxi jsou prakticky vždy zadávány).

```
mysql_connect(server_mysql, uživatelské_jméno, heslo );
```

Po přípojení k serveru se volí databáze funkcí mysql_select_db v syntaxi:

```
mysql_select_db("jméno_databáze", $proměná_s_
identifikací_přenosu);
```

Proměnná s identifikací přenosu není povinná. Většinou se pracuje pouze s jedním připojením k databázi a tehdy se nemusí uvádět. Identifikace přenosu je vrácena výše uvedeným příkazem mysql_connect při každém připojení k databázovému serveru [19].

Po té se již dá pracovat s tabulkami ve zvolené databázi pomocí příkazu mysql_query. (V případě, že se vytvoří více spojení, je proměnnou s identifikací přenosu nutno uvádět i u příkazu mysql_query. Jinak se většinou neuvádí.)

```
$sql=mysql_query("příkaz_sql", $proměná_s_
identifikací_přenosu);
```

Tento příkaz se přiřazuje do proměnné. Té pak vrací číslo identifikující výsledek zadaného dotazu. Toto číslo pak můžeme použít k dalším funkcím, z nichž nejpoužívanější jsou funkce pro práci s údaji vrácenými příkazem SELECT. Funkce Mysql_fetch_array postupně přiřazuje do pole řádky údajů z databáze (každou vrácenou buňku řádku tabulky do klíče pojmenovaného podle jména sloupce). Tím pádem při opětovném zavolání vypíše vždy následující řádek, a tak dále, dokud neprojde všechny vrácené řádky. Funkce mysql_num_rows vrátí počet řádků, které byly databází vráceny. Následující skript se připojí k databázi a vypíše identifikace a české názvy všech menu, které mají české nazvy přiděleny.

```
$hostname_server = "localhost";

$database_server = "jmeno_databaze";

$username_server = "login";

$password_server = "heslo";
```

```
$server = mysql_connect($hostname_server, $username_server,
$password_server) or die(mysql_error());

$pripojeni = mysql_select_db($database_server, $server) or
die(mysql_error());

$sql=mysql_query("SELECT *
                        FROM dp_menu, dp_menu_lang
                        WHERE index_menu = fkey_menu
                        and lang = „cs“
                        ORDER BY sort");
while ($menu = mysql_fetch_array($sql)){
echo $menu[„index_menu“]." ".$menu[„name“]."<br>";
}
```

Na řádku 6 a 7 se využívá výhodné vlastnosti operátoru „or“. Ten totiž postupně vyhodnocuje části, které spojuje, a to do té doby, než jedna část vyjde „true“, a pak skončí. Funkce „die“ vypíše svůj obsah a pak ukončí práci se skriptem. Funkce mysql_error vypíše poslední chybovou hlášku vrácenou mysql databází. Cyklus „while“ vypisuje postupně všechny řádky tabulky. Podmínka „while“ je splněna, pokud se podařilo do pole „menu“ přiřadit další řádek tabulky.

Práce s formuláři

Formuláře jsou nedílnou součástí HTML. Vytváří se pomocí tagu <form> a s využitím tagů pro konkrétní buňky (input, checkbox, textarea, submit, atd). Data, která se do formuláře vytvořeného v html zadají a odešlou, jsou pak přístupná skrze proměné v PHP. Jak se k nim dostat záleží na metodě, kterou formulář použil k odeslání dat. Tato metoda se nastavuje přímo v tagu „form“ a může nabývat hodnoty POST a GET [18]. Metoda GET je vhodná pro menší formuláře, neboť data připojí na konec url (např. index.php?bunka1=hodnota1&bunka2=hodnota2). Metoda POST se naopak hodí pro větší formuláře, neboť při ní jsou data přenášena v těle http požadavku. Z hlediska psaní skriptu není ve volbě metody prakticky žádný rozdíl. K údajům přeneseným metodou GET přistupujeme pomocí proměnné $_GET["jméno_buňky_formuláře"] a k údajům přenesených metodou POST obdobně – $_POST["jméno_buňky_formuláře"]. Při manipulaci s daty od uživatele a jejich vkládání do tabulky je třeba, aby nemohlo dojít

k takzvané *SQL injection*, neboli k tomu, že uživatel schválně zadá proměnnou tak, aby při vložení do dotazu SQL způsobila změnu struktury dotazu. Zda proměnná obsahuje potenciálně nebezpečné znaky (hlavně apostrofy a uvozovky), můžeme zkontrolovat pomocí speciální funkce. Ta před takové znaky přidá zpětnou uvozovku, a tak je vyřadí. Jedná se o funkci mysql_real_escape_string. Je však třeba si dát současně pozor, zda není v PHP zapnutá funkce magic_quotes_gpc, která by způsobovala zdvojení některých zpětných lomítek. Stav této funkce lze zjistit v nastavení PHP přes příkaz phpinfo();. Pokud jsou magic quotes zapnuté, je třeba se nejprve zbavit přebytečných lomítek funkcí „stripslashes“. Následující ukázkový skript načte z proměnné $_POST hodnotu v klíči jméno „jmeno“, která byla poslána z předešlé stránky pomocí formuláře, a aplikuje na tuto hodnotu zabezpečení proti *SQL injection*.

```
<?
$_POST["jmeno"]= stripslashes($_POST["jmeno"]);
// za předpokladu zapnutách magic_quotes
$jmeno= mysql_real_escape_string($_POST["jmeno"]);
// proměnná jméno je připravena k použití v příkazu
pracujícím s databází
?>
```

Objektově orientovaný návrh aplikace a její programování

Při návrhu a tvorbě aplikací je stále větším trendem používat objektové či objektově orientované programování. Jedná se přístup kde je na rozdíl od klasického strukturovaného programování, které odděluje data a funkce, kladen důraz na objekt, který je právě soubor dat a funkcí. Výhody objektově orientovaného přístupu jsou především v lepší správě, údržbě a rozšiřitelnosti kódu. Je to právě díky tomu, že objektové programování strukturuje návrh do jednotlivých objektů, které spolu komunikují pouze přes stanovené metody. Plyne z toho, že objektové programování je především vhodné pro větší projekty, jako je například návrh komplexního *web content management* systému. Nicméně klasické strukturované programování se v PHP často stále využívá, častokrát i v kombinaci s objektovým přístupem. Je to tím, že některé webové skripty jsou častokrát jen velmi jednoduché a napsat je se všemi náležitosti objektového přístupu by velikost aplikace několikanásobně zvětšilo. [23]

Při návrhu objektově orientované aplikace nejprve je třeba důkladně porozumět problému jímž se aplikace zabývá a aplikaci tvořit tak aby odpovídala všem požadavkům a měla všechny funkce, které se od ní vyžadují. Pro návrh používáme následující nástroje.

Zpravidla je třeba zjisti kdo a jakým způsobem bude chtít s aplikací pracovat. To zjistíme provedením use case analýzy– tj. zjistíme potenciální aktory, kteří mohou se systémem pracovat. A nalezneme a popíšeme všechny možné interakce, které se systémem mají. Tedy vlastně způsoby jakými systém mohou chtít použít.

Use case analýza se standardně skládá z use case diagramu a vlastního popisu jednotlivých použití systému. Pro tento popis se používají různé předlohy. Nejběžnější je use case template navržený v roce 1998 Derekem Colemanem [24]. Tento template s mírnou variací je použit pro popis použití systému v druhé části práce.

Pokud je problém složitější tak se zpravidla vytváří diagramy aktivit, které dopodrobna rozvíjejí jednotlivá use case, či scénáře, které popisují průchody vybraným use case.

Pro návrh systému je také třeba vytvořit vlastní statickou strukturu systému. Tedy popsat jednotlivé třídy objektů jejich atributy, metody a vazby mezi nimi. V UML se pro vyjádření statické struktury používá Class diagram. Toto bylo již popsáno v kapitole zabývající se navrhováním databází.[6]

Základy práce s funkcemi a třídami v PHP

Jazyk PHP verze 5 již nabízí velmi dobrou podporu objektově orientovaného programování. V této kapitole jsou probrány pouze základní principy tvorby tříd, jejich metod a atributů.

Kdekoli v kódu PHP můžeme definovat takzvanou funkci. Funkce se vytvoří pomocí klíčového slova function následovaného jménem a definicí dané funkce. Funkce musí být jasně ohraničená složenými závorkami a jednoznačně pojmenovaná. Jméno funkce musí začínat na písmeno či podtržítko a může obsahovat libovolné množství písmen, podtržítek a čísel a nesmí být pro PHP vyhrazené slovo (např.While nebo If). Vytvořená funkce pak

[6] Více k použití UML konceptů (use case, class a aktivity diagramů) lze najít například zde http://www.mahmoud-a.com/
course_files/swe/Applying%20UML%20concepts%20course%20management%20system.pdf

může být pomocí svého označení volaná odkudkoli z kódu. Funkce může mít stanovené parametry volání. Proměnné stanovené v hlavičce funkce jsou pak naplněny při jejím volání a při běhu funkce jsou k dispozici. V hlavičce lze stanovit standardní hodnotu parametru prostě tím, že parametru přiřadíme nějakou hodnotu. Takovýto parametr lze pak při volání funkce vynechat. Všechny proměnné uvnitř funkce spolu s ukončením funkce zanikají. Pokud chceme aby nám funkce vrátila nějakou hodnotu musíme jí stanovit na konci v příkazu return. Uvnitř funkce může být jakýkoli kód php včetně deklarace další funkce, či volání fukce včetně volání sebe sama.

Syntaxe pro vytvoření funkce je následující:

```
<?
Function jmeno_funkce($parametr1, $parametr2 = NULL){
Echo "zpracovavany kod";
$out = "funkce vrati tento text a za nim vypise promennou
parametr1, jejiz hodnota je zadana pri volani teto funkce -
".$parameter1;
return $out;
}
?>
```

Tato funkce má dva volací parametry, z nichž parametr1 je povinný – tj musí být zadán při volání funkce. Funkce vrací proměnnou out, ve které bude text stanovený uvnitř funkce. Příklady volání této funkce jsou například:

```
<?
Echo jmeno_funkce(„ahoj“);
$vypis = jmeno_funkce(„ahoj“, „todle je parametr 2“);
Echo $vypis;
Echo $parameter2;
?>
```

Pokud je v kódu spolu s těmito příkazy výše napsaná funkce a nic jiného tak, tyto příkazy vrátí dvakrát napsán text z proměnné out a budou končit slovem ahoj. Řádek s echo

$parameter2 nic nevrátí protože $parameter2 je proměnná stanovená pouze ve funkci a nevrací se nám.

Třídy se v PHP vytváří pomocí klíčového slova class, za ním následuje jméno a definice třídy. Pojmenování třídy má stejná pravidla jako výše popsané pojmenování funkcí. Je běžnou konvencí pojmenovávat třídy jménem s velkým písmenem na začátku. Třída může obsahovat svoje vlastní konstanty, proměnné a funkce. V návaznosti na objektový návrh lze zjednodušeně říci, že odpovídají atributy stanoveným globálním proměnným dané třídy a metody odpovídají funkcím dané třídy.

Všechny proměnné a funkce ve třídě se dají volat jen uvnitř dané třídy pomocí pseudo-kombinací proměnné $this, operátoru -> a jména funkce či proměnné. Nebo pokud nejsou omezeny tak zvenčí přes identifikátor instance třídy a opět operátoru -> a názvu proměnné či funkce. V objektech se v PHP pohybujeme pomocí Jedním ze základních principů objektového programování je zapouzdření. To jest princip tvoření tříd tak, aby k datům uvnitř třídy se přistupovalo jen pomocí k tomu přidělených metod. V php nám k zajištění zapouzdření slouží takzvané popisovače, které při deklarování proměnné či funkce určí odkud se k ní může přistupovat. Popisovač public nám značí, že k funkci či proměnné se může přistupovat odkudkoli. Popisovač protected značí, že k ní smí pouze třída, která ji vytvořila, nebo z ní odvozená třída. A popisovač private značí, že k ní smí pouze třída, která ji vytvořila.

Proměnnou pro danou třídu vytvoříme pomocí klíčového slova „var“, případně v kombinaci s popisovačem, který omezí přístup k ní. Funkci vytvoříme klasicky pomocí klíčového slova „function“, opět případně doplněného o popisovač.

Ve třídě lze definovat konstruktor třídy. To je speciální funkce, která se buď může pojmenovat „_construct“ nebo může mít stejné jméno jako třída a volá se automaticky při vytvoření instance třídy.

Ve chvíli kdy máme definovanou třídu, tak můžeme tvořit její instance pomocí operátoru new. [25]

Následuje příklad vytvoření třídy s konstruktorem. Tato třída při vytvoření vypíše „Ahoj“. Na konci jsou jí vytvořeny 2 instance, takže 2x vypíše ahoj. Dále se pro první instanci nastaví pomocí funkce „set_promena()“ do proměnné ve třídě slovo „Dobrý den“. A

nakonec se zavolá a vypíše funkce „get_promena()“ nejprve pro první a pak pro druhou instanci třídy. U první instance se vypíše „Dobrý den“ a u druhé „Ahoj“.

```
<?
class jmeno_tridy
{
   Private var $promenna = "Ahoj";
   function jmeno_tridy()
      echo $this->promenna;
      return;
   }
   Function set_promenna($nastav){
      $this->promenna = $nastav;
   }
   Function get_promenna(){
      return $this->promenna;
   }



$instance1 = new jmeno_tridy();
$instance2 = new jmeno_tridy();
$instance1->set_promenna("Dobrý den");
Echo $instance1->get_promenna();
Echo $instance2->get_promenna();


?>
```

Vlastní návrh systému

V rešeršní části práce bylo popsáno, co je *web content management* systém. Byl objasněn běžný proces návrhu relačních databází a proces přetvoření tohoto návrhu pomocí jazyka

SQL do fyzické databáze podle konvencí systému řízení báze dat MySQL. Dále byly popsány základy jazyka PHP následované popisem základního přístupu používaného při navrhování objektových aplikací. Nakonec bylo vysvětleno, jakým způsobem je v PHP podporována práce s objekty. V následující kapitole budou znalosti z literární rešerše využity při popisování tvorby dvou konkrétních částí *web content management* systému.

Popis zadání a návrhu řešení

Protože tato práce má omezený rozsah a protože při tvorbě *web content management* systému se lze zabývat mnoha různými otázkami, bylo po domluvě se školitelem zaměření práce zúženo. V této části práce tedy bude popsána problematika a návrh pouze dvou specifických systémů. Ty budou obsahovat aplikaci v jazyce PHP, která bude načítat informace z relační databáze spravované systémem řízení báze dat Mysql. Tyto aplikace budou napsány objektově, a mohou se díky tomu velmi jednoduše zakomponovat do běžného širšího celku *web content management* systému. Bude se jednat o následující dvě aplikace:

- zaprvé, o aplikaci mající za úkol uchovávat menu webové prezentace v databázi a následně vypisovat menu pro webovou prezentaci na vlastní zobrazovanou na stránku;
- druhá aplikace se bude starat o uschovávání a vypisování vlastního obsahu stránky – bude spolupracovat s první aplikací tím, že jednotlivé obsahy bude umisťovat do příslušných menu.

Zdánlivě se jedná o jednoduché aplikace. V jednoduché podobě by je bylo možné pro konkrétní použití na konkrétní stránce naprogramovat během odpoledne. V této práci je však popsán návrh univerzální, který se dá jednoduše a rychle přizpůsobit a použít prakticky pro jakoukoli webovou aplikaci. Obě aplikace budou mít za cílovou uživatelskou skupinu administrátory a tvůrce stránek. To znamená uživatele, kteří strukturu stránek tvoří a mají znalosti programování a práce s databází. Navrhovaná aplikace by těmto uživatelům měla usnadnit vlastní tvorbu webových stránek ve větším rozsahu a pro různé klienty, a to na jednotném základě. Měla by totiž umožnit jednoduchou tvorbu různě strukturovaných webů a zahrnutí nejběžnějších druhů obsahu - od běžné stránky až po výpis a formátování položek z databáze. V práci nebude popsána tvorba samotných formulářů na vkládání a

úpravu menu a obsahu. Pokud je naržena struktura databáze pro uchovávání dat, vytvořit k ní funkční formulář pro vkládání obsahu je pak už poměrně triviální úloha, v literatuře mnohokrát popsaná .[7]

Jak bylo zmíněno v literární rešerši, tak před začátkem vlastního návrhu systému je vždy třeba prozkoumat, co vlastně od finální aplikace požadujeme. Pokud bychom na začátku návrhu takovou úvahu neprovedli, snadno by se mohlo později stát, že by se objevila nutnost od základu předělat celý návrh, a to třeba i jenom kvůli přidání nějaké nové funkce či zajištění kompatibility s jinými aplikacemi. Pro vlastní analýzu možných využití aplikace použita metoda *use case*.

Aplikace pro načítání menu

***Use case*: Načtení menu**

Popis: Aplikace vytvoří menu v jazyce html. Menu se bude skládat z jednotlivých položek – odkazů (html tag „anchor“). Odkazy budou mít nastavený parametr „title“ a samozřejmě parametr „href“ obsahující url adresu na danou položku menu. Jednotlivé odkazy budou strukturovány jako položky v neuspořádaném seznamu (html tagy „ul“ a „li“). Vlastní grafiku zobrazení není třeba řešit, neboť o tu se postarají css styly každé webové prezentace jinak. Jedna z položek menu bude vždy vybraná a bude označená specifickou třídou css (např class=“aktivni-menu“), což tuto položku umožní snadno v menu na stránce zvýraznit. Aplikace musí být schopna pracovat s menu ve více jazycích. Aplikace také musí být schopná pracovat s více menu současně a musí umět rozdělit jedno menu na dvě části – tedy zobrazit jednu část menu na jednom místě stránky a druhou část na druhém. Aplikace dále musí spolupracovat s nějakým systémem správy přístupů. Musí tedy umožňovat právo vidět jednotlivá menu podle uživatelských oprávnění. Aplikace by měla dovolit pracovat s menu, která mají libovolně úrovní. S menu by tedy měla pracovat jako se stromem. Aplikace musí umět vytvořit celé menu – tj. vypsat celý strom s vyznačenou cestou ke zvolenému vrcholu. To je typický požadavek pro menu, které má být vidět celé, a nebo se načítá celé, zatímco jeho části se ukazují a schovávají pomocí kaskádových stylů.

[7] Této otázce jsem se ostatně sám věnoval už v mé bakalářské práci [28], která se zabývá návrhem jednoduché webové prezentace napojené na databázi publikací. Práce je dostupná na adrese http://dp.mangaya.cz/xkonj108_BP.pdf.

Aplikace musí být schopna vytvořit pouze ty části menu, které mají stejného rodiče jako právě zvolená položka, nebo jsou přímými potomky právě zvolené položky. To se uplatní v případě menu, kde se podnabídka zobrazí až po kliknutí na položku rodičovského menu, a stránka se tedy znovu načte. Aplikace musí být schopná pracovat s položkami menu označenými pouze jako kategorie. Takto označená položka při rozkliknutí nenačte sebe, ale prvního svého potomka, který sám takto označen není; pokud je načteno celé menu, takováto položka se nezobrazí jako odkaz – nepůjde na ní tedy kliknout (pokud je zvolen typ menu, kde se podnabídka zobrazuje až po zvolení rodičovského menu, musí jít samozřejmě kliknout i na kategorii). Aplikace by měla umožnit napojení na aplikaci tvorby „hezkých“ url: url adresa vygenerovaná pro danou položku menu se odešle k možnému přetvoření do hezké (jednoduché, snadno zapamatovatelné) podoby a pro další práci se pak použije tvar, který se mu vrátí (například z „/index.php?m=5“ bude „/dipomova-prace.html“).

Účastníci: Uživatelé skrze CMS webové prezentace,

Předpoklady: Předpokládá se zakomponování této aplikace do fungující webové prezentace. Z ní by měla aplikace získávat informaci o aktuálně používaném jazyku. Aplikace předpokládá nastavení hodnot potřebných pro běh v konfiguračním souboru webové prezentace. Proměnné, které je třeba nastavit, budou popsány v hlavičce hlavního souboru aplikace. Pro správný běh aplikace je potřebné napojení na databázi MySQL. Aplikace předpokládá přístup do této databáze přes třídu C_MySQL (viz přílohy). Je třeba, aby byly v přistupované databázi vytvořeny tabulky tak, jak jsou specifikovány v hlavičce aplikace. Předpokládá se, že menu jsou do databáze vkládány nějakým formulářem skrze systém pro úpravu tabulek v MySQL. Aplikace neřeší vkládání menu do databáze – pouze jejich načítání.

Kroky: Inicializace třídy; následuje libovolný počet volání zobrazení menu s definovaným vrcholem stromu, jehož potomci se mají zobrazit.

Use case rozšíření: Vytvoření *sitemap* ze struktury menu

Popis: Aplikace vytvoří pole, které bude obsahovat všechny dostupné url . Toto pole pak dá k dispozci pro další části *content management* systému webové prezentace. Aplikace by

měla umožnit napojení na aplikaci tvorby „hezkých“ url. V případě, že je pro dané menu stanovena „hezká“ url, měla by být použita v seznamu adres pro *sitemap* místo klasické url.

Účastníci: Uživatelé CMS webové prezentace.

Předpoklady: Stejné jako u prvního *use case*.

Kroky: Inicializace třídy; proběhne vypsání menu; webová prezentace si zažádá pomocí funkce o vrácení pole s url.

Use case rozšíření: Vypsání drobečkové navigace[8]

Popis: Aplikace vytvoří menu jako v prvním *use case*. Následně umožní vytvořit pro tuto nabídku tzv. drobečkovou navigaci. To je navigace, která umožňuje snadnou orientaci uživatele webu, které položky nabídky jsou právě zvolené. Aplikace umožní zvolit oddělovač mezi jednotlivými odkazy pomocí zadání parametru při volání.

Účastníci: CMS webové prezentace.

Předpoklady: Stejné jako u prvního *use case*.

Kroky: Inicializace třídy; následuje libovolný počet volání zobrazení menu s definovaným vrcholem stromu, jehož potomci se mají zobrazit. Po každém zobrazení menu si webová prezentace může zažádat pomocí funkce o vytvoření drobečkové navigace pro naposledy vytvořené menu.

Use case rozšíření: Vrácení podružných informací o menu

Popis: Aplikace vytvoří menu jako v prvním *use case*. Následně umožní content management systému webové prezentace přístup k datům o zvolené položce menu. Načte id, název, popisek stránky a klíčová slova přidělená jednotlivým menu v právě používaném jazyce.

Účastníci: Uživatel CMS webové prezentace.

Předpoklady: Stejné jako u prvního *use case*.

---

[8] Více o drobečkové navigaci například zde [26].

Kroky: Inicializace třídy; následuje libovolný počet volání zobrazení menu s definovaným vrcholem stromu, jehož potomci se mají zobrazit. Po každém zobrazení menu si webová prezentace zažádat pomocí funkce o informace k naposledy vytvořenému menu.

Aplikace pro načítání obsahu

Obecný popis: Aplikace vytvoří a vypíše obsahovou část webové stránky – tj. část normálně se nacházející v tagu „body“, nebo část stránky, která byla aplikaci přidělena. Aplikace dovolí zvolit *template* pomocí parametru při volání spouštěcí funkce. *Template* (vzor) se rozumí soubor s vlastním obsahem, který má být uživateli webové prezentace vypsán. *Template* může obsahovat další PHP kód ke zpracování, včetně možnosti provádět další volání funkcí v této aplikaci. Pokud není zvolen *template* nebo pokud zvolený *template* není k dispozici, načte se *template* základní, přednastavený.

***Use case*: načtení jednoduchého statického obsahu pro menu**

Popis: Aplikace bude spolupracovat s výše popsanou aplikací, která tvoří menu. Aplikace bude umožňovat vytvořit *template*, který (ve spolupráci s výše popsanou aplikací) zobrazovat menu a umožní vypsání obsahu, který k dané nabídce přísluší. Obsah může mít více částí. Zároveň musí být schopen přizpůsobit se jazykovým potřebám webové prezentace. Obsah se bude vypisovat do *tepmplate* skrze šablonu[9], která bude umožňovat jeho dodatečné formátování. Například obsah má dvě části – nadpis a text. Šablona vypíše část „nadpis“ do tagu „h1“ a část „text“ do tagu „p“. Šablona i obsah mohou být dynamické, resp. obsahovat příkazy php.

Účastníci: Uživatelé skrze hlavní řídící soubor webové prezentace.

Předpoklady: Předpokládá se zakomponování této aplikace do fungující webové prezentace. Ta by měla opět dodávat naší aplikaci údaj o tom, jaký jazyk prezentace je momentálně zvolený. Aplikace předpokládá nastavení hodnot potřebných pro běh v konfiguračním souboru webové prezentace. Proměnné, které je třeba nastavit, budou i

[9] Výraz template (česky vzor či šablona) je v popisu aplikace používán pro vzory použité načítání původního rozvržení stránky. Kdyžto šablona je používány výhradně pro vzory používané k načítání obsahu přiřazeného k menu. Podrobnější vysvětlení se nachází ve vlastním návrhu PHP aplikací.

zde popsány v hlavičce hlavního souboru aplikace. Pro správný běh aplikace je potřebné napojení na databázi MySQL. Aplikace předpokládá přístup do této databáze přes třídu C_MySQL (viz přílohy). Je třeba, aby byly v přistupované databázi tabulky vytvořeny tak, jak jsou specifikovány v hlavičce aplikace. Předpokládá se, že obsah je do databáze vkládán formulářem skrze systém pro úpravu tabulek v MySQL. Ani tato aplikace neřeší samotné vkládání obsahu do databáze – řeší pouze jeho načítání. Předpokládá se vhodné nastavení *template*.

Kroky: Hlavní řídící řídící soubor načte zvolený template. Vrátí se mu jeho obsah.

***Use case* rozšíření: načtení alternativního statického obsahu podle url**

Popis: Pokud je tak stanoveno proměnnou, aplikace bude moci při načítání obsahu připojeného k nabídce načíst alternativní obsah (místo klasického obsahu menu) a vložit ho do *template* (místo klasického obsahu připojeného k menu). Tento obsah se načte voláním funkce se jménem, které je stanoveno parametrem.

Účastníci: Uživatel skrze hlavní řídící soubor webové prezentace.

Předpoklady: Stejné jako u prvního *use case*. V *template* musí být stanovena podmínka, kdy se má načíst alternativní obsah a jaká funkce ho zpracovává.

Kroky: Hlavní řídící soubor načte zvolený *template*. Vrátí se mu jeho obsah.

***Use case* rozšíření: načtení a vypsání dat z databáze**

Popis: Stejně jako u prvního *use case* načte aplikace obsah připojený k menu. Obsah však bude obsahovat sql dotaz na databázi. Aplikace následně provede dotaz a jeho výsledky vypíše uživateli (opět přes zadanou šablonu).

Účastníci: Uživatelé skrze hlavní řídící soubor webové prezentace.

Předpoklady: Stejné jako u prvního *use case*. Správně zadaný sql dotaz a zadaná šablona.

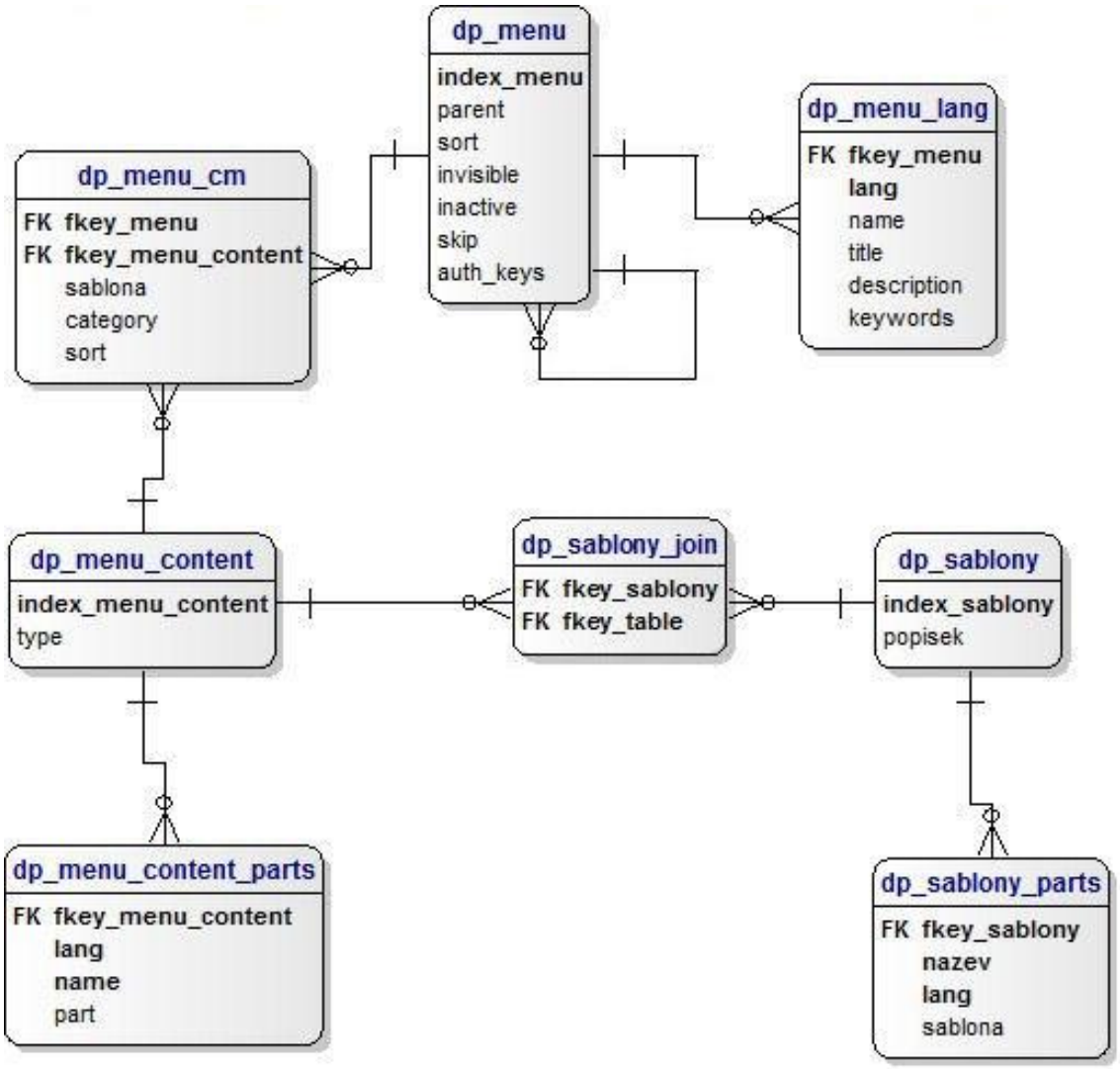
Kroky: Hlavní řídící soubor načte zvolený *template*. Vrátí se mu jeho obsah.

Návrh databázového modelu

V následující kapitole je popsán databázový model navržený pro uchování struktury menu, obsahů a šablon k nim náležejících.

Obě aplikace, kterými se zabývá tato diplomová práce, dovolují stanovit si předponu pro jména tabulek, které k nim náležejí. Díky tomu se lépe udržuje pořádek mezi tabulkami v databázi; a zároveň to umožňuje běh více oddělených systémů, resp. webových prezentací, současně připojených k jedné databázi. Podle konvencí definice databáze by k něčemu takovému nemělo docházet. Nicméně častokrát je k doméně, na které běží více webových prezentací, přiřazena administrátorem serveru jen jediná databáze a tvůrce/navrhovatel vlastních webových prezentací s tím nic nezmůže. V průběhu celé práce je pro jednoduchost a lepší pochopení vysvětlované problematiky používána při popisu navrhovaných aplikací předpona „dp_“. Tato předpona je použita i v následujícím diagramu, který popisuje logickou strukturu. Zde by správně neměla být, ale pro snadnější pochopení navrhování systému byly i tady použity stejné názvy tabulek a atributů jako ve vlastním návrhu databáze.

Na níže uvedeném diagramu je vidět kompletní navržený logický model včetně klíčů a asociační tabulek. Tučně jsou zvýrazněny primární klíče. Cizí klíče mají u sebe popisek FK. Tento diagram byl vytvořen v demoverzi programu *DeZign for Databases* verze 6.3.1. Model byl v procesu tvorby samozřejmě několikrát refaktorován, aby odpovídal požadavkům výše uvedených *use cases* a samozřejmě aby byl vhodný pro realizaci v relační databázi.



Část pro aplikaci načítající menu

Aplikace načítající menu pracuje pouze s tabulkami dp_menu a dp_menu_lang. Tabulka dp_menu má za primární klíč atribut index_menu. Je to „umělý“ atribut sloužící pouze k identifikaci daného řádku v tabulce. V této tabulce budou uchované všechny informace o jednotlivých menu, které nezávisí na jazyku webové prezentace. Každému menu odpovídá jeden řádek. Tabulka dp_menu má vazbu s kardinalitou 1:n s tabulkou dp_menu_lang. V této tabulce jsou uchovány informace o menu, které závisí na jazyku webové prezentace. Jako primární klíč byla zvolena kombinace atributů fkey_menu a atribut lang. Atribut fkey_menu je zároveň cizí klíč obsahující hodnoty atributů index_menu z tabulky dp_menu. Tato struktura umožňuje vytvářet neomezené množství jazykových mutací každého menu; jak přesně bude vkládání vypadat, se ale bude samozřejmě lišit stránku od

stránky, a také na vytvořeném formuláři pro vkládání dat. Následuje vysvětlení jednotlivých atributů.

Tabulka dp_menu – Obsahuje jednotlivá menu – jeden řádek v tabulce odpovídá jedné položce menu na stránce.

- **index_menu** – Vygenerovaná unikátní číselná hodnota sloužící jako identifikátor jednotlivých položek menu (primární klíč tabulky).
- **parent** – Tento atribut obsahuje hodnotu atributu index_menu z položky menu, které je této položce nadřazené. Celé zobrazované menu začíná položkou menu, která se nezobrazuje a nemá nastavenou žádnou „parent“ hodnotu. Je tedy vrcholem celého stromu. Ostatní položky menu jsou na něj navázané a v atributu „parent“ mají vždy hodnotu svého rodiče.
- **sort** - Tento atribut určuje pořadí, v jakém se vypisují jednotlivé položky menu v rámci potomků jednoho rodiče.
- **invisible** - Tento atribut označuje menu, které uživatel vidí či nevidí. Pokud je menu označeno jako „invisible“, tak jej uživatel jako položku nabídky na stránce neuvidí. Pokud však zná adresu url, může se na něj i tak dostat.
- **inactive** - Tento atribut označuje menu, do kterých může/nemůže přistupovat normální uživatel. Pokud je menu označeno jako „inactive“, je pro všechny potřeby vypnuto. Vidí ho pouze administrátor stránky. Uživatel se na něj nedostane dokonce ani tehdy, když zná url, která by na tuto nabídku běžně vedla.
- **skip** – Tento atribut označuje, že menu nemá vlastní obsah. Pokud se načítá celé menu, je takovéto menu zobrazeno pouze jako kategorie, na kterou nejde kliknout. Pokud se načítá pouze větev menu, tak se po vybrání tohoto menu automaticky zvolí jeho první položka, která tímto způsobem není označena.
- **auth_keys** – Tento atribut je připraven k napojení na aplikaci, která se stará o správu uživatelů. Ve webových prezentacích, ve kterých byla tato aplikace používána, se do daného sloupce většinou umisťovala informace o tom, jaké oprávnění je třeba, aby uživatel viděl dané menu. A to buď ve formě čísla jako úrovně zabezpečení, nebo jako seznam skupin uživatelů. Pokud je pole necháno prázdné, vidí ho jakýkoli uživatel webové prezentace.

Tabulka dp_menu_lang – Obsahuje jazykové mutace informací o jednotlivých menu. Jedna položka menu na stránce může mít neomezeně řádků v této tabulce. To v praxi většinou znamená, že má tolik řádků, v kolika jazycích je webová prezentace navržena.

- **fkey_menu** – Cizí klíč obsahující napojení na tabulku dp_menu, tedy vlastně na jednotlivé položky menu.
- **lang** – Označení dané jazykové mutace.
- **name** – Název menu, tj. text zobrazující se uživateli jako položka menu.
- **title** – Obsah parametru „title“ v tagu „a“, tedy text, který se zobrazí při najetí myši nad danou položku menu.
- **description** – Text pro metatag, který se načítá do hlavičky webové stránky.[10]
- **keywords** – Text pro další metatag, který se načítá do hlavičky webové stránky.

Během psaní této práce byly navržené aplikace v několika různých verzích využity na celkem pěti webových prezentacích. V současné době ovšem všechny tyto prezentace používají aktuální verzi, která se nachází v příloze této práce.

Na níže uvedeném obrázku je vidět příklad formuláře použitý na stránce českého petanque klubu PK1293 Vojnův Městec (http://pk1293.vojnuvmestec.cz). Protože se o zmíněnou stránku stará zkušený správce obsahu, umožňuje formulář editace měnit veškeré nastavení a lze volně přidávat a mazat jednotlivé jazykové mutace. U jiných webových prezentací, kde se o obsah stará méně zkušený uživatel internetu, je samozřejmě lepší pokročilá nastavení v editaci nezobrazovat a nastavit i jazyky napevno – když tedy vytváříme instanci menu v tabulce dp_menu, přidáme rovnou předem zvolené jazyky do dp_menu_lang a zobrazíme pro ně formulář (vkládajícího tím nutíme vždy vyplnit všechny jazykové varianty).

[10] Více o metatazích lze najít například zde http://searchenginewatch.com/2167931.


Editace menu
Index menu:
Založení nového menu
Jazykové mutace:
nová jazyková mutace
Řazení:
Nadřazené menu:
Neviditelné:
Inaktivní:
Přeskočit:
save
Editace menu
Index menu:14
Editace
Jazykové mutace:
Editace tabulky pk1293_menu_lang
Jazyk:
cs
Název:
Turnaje
Title:
Turnaje pořádané PK1293 Vojnův Městec
Description metatag:
Přehled petanque turnajů pořádaných PK1293 Vojnův Městec
Keywords metatag:
petanque, turnaj, vojnův městec, čapek
smazat
Editace tabulky pk1293_menu_lang
Jazyk:
en
Název:
Tournaments
Title:
Tournaments organized by PK1293 Vojnův Městec
Description metatag:
List of petanque tournaments organized by PK1293 Vojnův Městec
Keywords metatag:
petanque, tournament, vojnův městec, čapek
smazat
nová jazyková mutace
Řazení:
2
Nadřazené menu:
1
Neviditelné:
Inaktivní:
Přeskočit:
save
smazat

Část pro aplikaci načítající obsah

Zbytek tabulek v modelu používá druhá aplikace, tedy aplikace načítající obsah. Ve výše popsaných specifikacích *use case* bylo zadáno, aby aplikace starající se o správu obsahu byla schopná načítat obsah k jednotlivým položkám menu. Toho bylo dosaženo napojením tabulky dp_menu, tedy tabulky uchovávající informace o jednotlivých položkách menu (popsané v předchozí kapitole), na tabulku dp_menu_content. Vlastní aplikace byla navržena tak, aby byla schopna uchovávat běžný obsah stránky přímo v databázi. Tento obsah je uložen právě v tabulce dp_menu_content a v na ní navázané tabulce dp_menu_content parts. Tyto dvě tabulky podobně jako tabulky dp_menu a dp_menu_lang umožňují, aby jeden obsah měl různé jazykové mutace. Dp_menu_content_parts však nemá jako součást primárního klíče pouze cizí klíč z dp_menu_content a atribut obsahující označení jazyka, ale i atribut „name“. To umožňuje skládat jeden obsah nejen z více jazykových mutací, ale i z více částí stanovených jménem. Aby byl celý systém maximálně uživatelsky pružný a otevřený, uvažujeme také možnost přiřadit k jedné položce menu více rozdílných obsahů. A samozřejmě také pracujeme s možností připojit jeden obsah k více položkám menu. Proto je mezi tabulky dp_menu a dp_menu_content vložena asociační tabulka dp_menu_cm. Na tabulku dp_menu_content je napojena i tabulka obsahující šablony. Obsah je na stránku vypisován a formátován podle předem připravených pravidel, resp. šablon. I šablony se stejně jako obsahy ukládají ve dvou tabulkách, aby bylo umožněno mít pod „hlavičkou“ jedné šablony více částí ve více jazykových mutacích. Přesné fungování obsahů a šablon bude dopodrobna popsáno níže v kapitole „Realizace navrhovaného řešení“. Následuje vysvětlení jednotlivých atributů.

Tabulka dp_menu_content - Obsahuje jednotlivé obsahy – jeden řádek v tabulce odpovídá jednomu obsahu.

- **index_menu_content** – Vygenerovaná unikátní číselná hodnota sloužící jako identifikátor jednotlivých obsahů (primární klíč tabulky).
- **type** – Určuje, jakým způsobem se bude s obsahem zacházet. Součástí aplikace popsané v této práci jsou metody pro vypisování dvou typů obsahů. Metoda „print“ pro jednoduché vypsání textu a metoda „sql“ pro načtení dat z jiné tabulky.

Tabulka dp_menu_content_parts - Obsahuje různé součásti jednotlivých obsahů. Jeden obsah může mít v této tabulce neomezeně řádků; resp. v praxi tolik řádků, v kolika jazycích je webová prezentace, násobeno počtem součástí obsahu.

- **fkey_menu_content** – Cizí klíč obsahující napojení na tabulku dp_menu_content, tj. vlastně na jednotlivé obsahy.
- **lang** - Označení dané jazykové mutace.
- **name** – Tento atribut obsahuje název obsahu. Název je používán pouze pro práci se šablonami a s aplikací. Je tedy důležitý pro administrátora, ale uživatel ho nikdy neuvidí. Atribut je také součástí složeného primárního klíče.
- **part** – Vlastní obsah, který určuje, co se vypíše na stránku.

Tabulka dp_sablony - Obsahuje jednotlivé šablony – jeden řádek v tabulce odpovídá jedné šabloně.

- **index_sablony** – Vygenerovaná unikátní číselná hodnota, která slouží jako identifikátor jednotlivých šablon (primární klíč tabulky).
- **popisek** – Atribut čistě pro interní potřebu administrátora, aby se šlo v šablonách snadněji vyznat; dalo by se říci název šablony.

Tabulka dp_sablony_parts - Obsahuje různé součásti jednotlivých šablon. Jedna šablona může mít v této tabulce neomezeně řádků; resp. v praxi tolik řádků, v kolika jazycích je webová prezentace, násobeno počtem součástí šablony.

- **fkey_sablony** – Cizí klíč obsahující napojení na tabulku dp_sablony, tedy vlastně na jednotlivé šablony.
- **nazev** – Tento atribut obsahuje název šablony. Název je používán pouze pro práci s obsahy a s aplikací. Je tedy opět důležitý pro administrátora, ale uživatel ho nikdy neuvidí. Atribut je také součástí složeného primárního klíče. Protože při běžném použití se více druhů šablon nevyužije, představuje standardní název šablony hodnota „default“.
- **lang** – Označení dané jazykové mutace.
- **sablony** – Vlastní šablona, která určuje, co se vypíše na stránku.

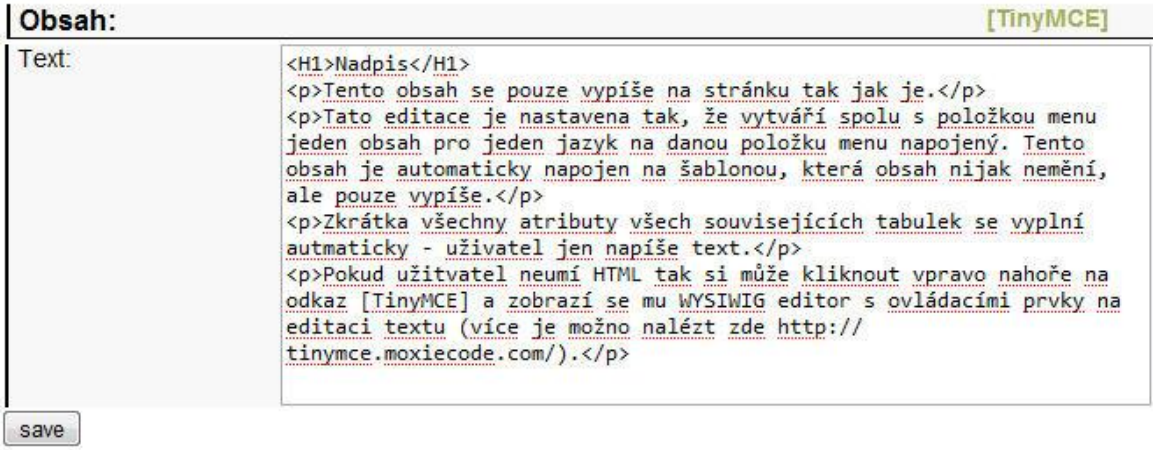
Tabulka dp_sablony_join – Asociační tabulka navržená na přiřazování šablon k obsahům.

- **fkey_sablony** – Cizí klíč obsahující napojení na tabulku dp_sablony, tedy vlastně na jednotlivé šablony.
- **fkey_table** – Cizí klíč obsahující napojení na tabulku dp_content, čili na jednotlivé obsahy. Tento atribut by se správně měl jmenovat fkey_menu_content, aby bylo přesně vyjádřeno, co obsahuje. Není tomu tak kvůli zpětné kompatibilitě. V návrhu aplikace totiž byly provedeny změny ve chvíli, kdy se již uvedený název používal pro běžící webovou aplikaci. Tento název také umožňuje snazší psaní dotazů SQL na šablony a obsahy.

Tabulka dp_menu_cm - Asociační tabulka navržená na přiřazování obsahů k položkám menu.

- **fkey_menu** - Cizí klíč obsahující napojení na tabulku dp_menu, a tedy na jednotlivé položky menu.
- **fkey_menu_content** – Cizí klíč obsahující napojení na tabulku dp_menu_content, tedy vlastně na jednotlivé obsahy.
- **sablona** – Určuje, která část šablony se má využít. Šablona je jako celek připojena k jednotlivým obsahům. V napojení na jednotlivá menu se však může specifikovat, která její část se má použít pro dané menu. Políčko „sablona“ obsahuje název šablony, který je vyplněn u části šablony v atributu „nazev“. Je to tedy nepřímo napojení na dp_sablony_parts. Protože běžně více druhů šablon nevyužijeme, je opět hodnota „default“ standardním názvem šablony.
- **category** – Jedná se o označení obsahu, které využije pouze administrátor a aplikace. Standardní název kategorie je hodnota „main“ (opět proto, že typicky se použije jen jedna kategorie obsahu).
- **sort -** Tento atribut určuje řazení obsahu v jednotlivých menu.

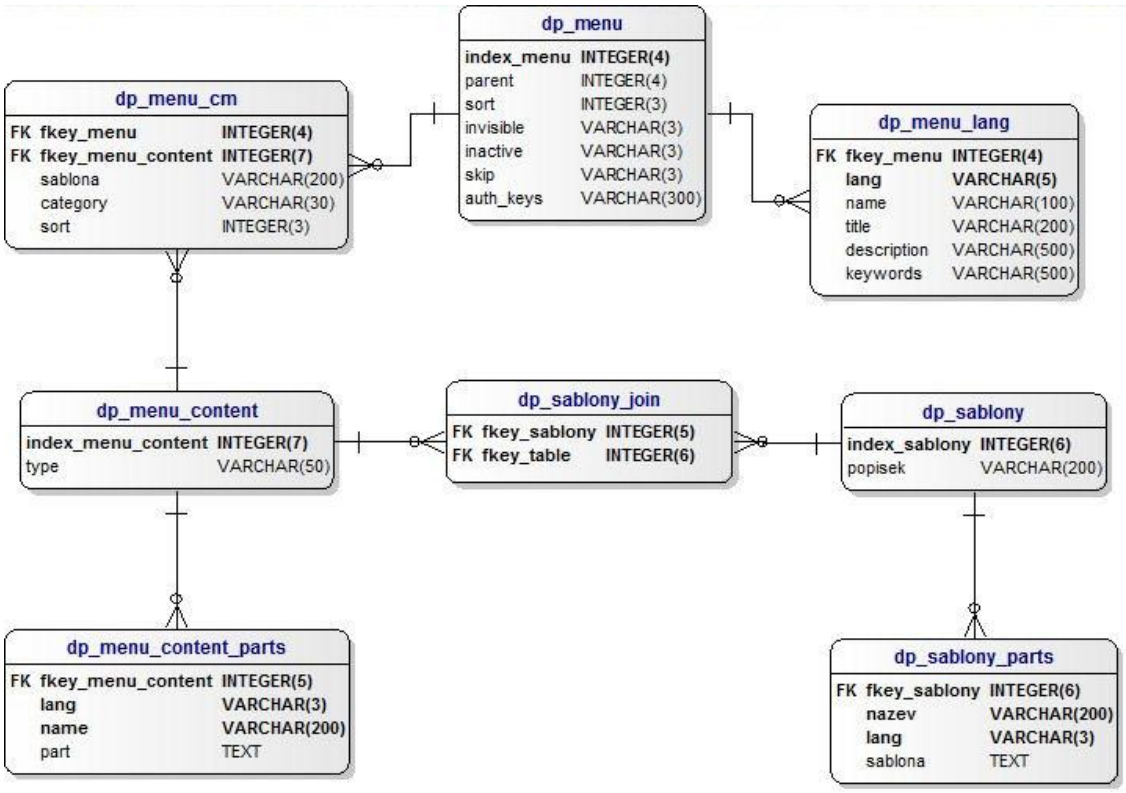
Narozdíl od návrhu menu je princip a obsah tabulek na editaci šablon a obsahů značně složitější a formulář se může výrazně lišit případu od případu. Níže je vidět nejjednodušší podoba formuláře, určená pro uživatele-začátečníka. Formulář na obrázku je z druhé stránky, kde byla tato aplikace použita - http://eko-consult.cz/.



Složitější příklady, jak by mohly vypadat formuláře pro editaci výše uvedených tabulek, najdete v části Realizace navrhovaného řešení, a to spolu s vysvětlením, jak vlastně načítání obsahů přes šablony pracuje.

Realizace modelu v MySQL

Model byl realizován v MySQL verze 5. Oproti databázovému modelu, který byl navržen v předchozí kapitole, byly doplněny vhodné typy a omezení atributů. Protože se předpokládá využívání aplikace primárně pro české stránky, byl charset tabulek nastaven na utf8_czech_ci. Namísto v MySQL přednastaveného MyISAM byl jako engine úložiště dat zvolen InnoDB. Stalo se tak proto, že v případě webových prezentací, pro které byly zde popisované aplikace vyvinuty a použity, využíval systém pro tvoření formulářů na vkládání dat možnosti transakcí; a ty bohužel engine MyISAM nepodporuje. Pokud by byl použit jiný systém pro tvorbu formulářů, je možné bez problému převést tabulky do MyISAM, který je o trochu rychlejší než InnoDB [15]. Vlastní příkazy na tvorbu všech tabulek lze nalézt v příloze. Pro úplnost následuje již jednou výše uvedený diagram doplněný o datové typy a omezení jednotlivých atributů. Kvůli zjednodušení a snadné přenositelnosti mezi jednotlivými SŘBD byly použity nejběžnější datové typy „integer“, „varchar" a „text“.



Realizace navrhovaného řešení

V této kapitole bude nejen vysvětleno, jakým způsobem byla vytvořena PHP část navrhovaných aplikací, ale budou zde i popsány příklady toho, jak je možno dané aplikace využít (resp. jak již byla tato aplikace využita v reálných webových prezentacích). Kapitola je opět rozdělena do tří částí. První dvě se zabývají každá jednou ze dvou popisovaných aplikací a třetí část popisuje složitější praktické příklady využití obou.

C_menu

První popisovaná aplikace, která se primárně stará o zobrazování menu, byla v PHP zpracována do jedné třídy nazvané C_menu. Tato třída se nachází v přílohách ve stejnojmenném souboru. Třída odpovídá požadavkům stanoveným ve výše popsaném *use case*. Třída potřebuje samozřejmě napojení na databázi, která obsahuje výše popsané tabulky. Počítá se s napojením na databázi nikoli přímo, ale jak je dnes běžně zvykem přes další třídu, která se stará o autentizaci připojení a vlastní vytváření tunelů. Ve webových prezentacích, kde byla zde popisovaná aplikace testována, se jednalo o třídu C_MySQL. Volání metod této třídy se však dají v případě potřeby velice jednoduše nahradit za volání metod třídy jiné, případně zcela vynechat. Třída také počítá s napojením na další třídy,

které jsou běžné pro jakýkoli *web content management* systém. Jedná se o třídu spravující práva uživatelů a třídu spravující přesměrování a tvorbu hezkých url. Tyto třídy nejsou pro chod popisované třídy C_menu vyžadované, ale jejich použití je rozhodně pro řadu situací vhodné a užitečné. Ve webových prezentacích, kde byla popisovaná aplikace použita, byla pro správu přístupů použita třída C_Auth a pro tvorbu hezkých url pak třída C_router. V přiložených souborech jsou odkazy na tyto třídy zachovány, což slouží jako příklad realizace daných připojení a využití funkčnosti těchto volitelných tříd. Odkazy na volání metod těchto tříd je nutno před použitím třídy C_menu nahradit odkazy na třídy konkrétní webové prezentace, kde je třída C_menu použita.

Další, co třída C_menu pro svoji funkčnost potřebuje, je nastavení proměnných, které určují její chod. Tyto proměnné by měly být nastaveny před voláním vlastní třídy v konfiguračním souboru dané webové prezentace. Mezi nejdůležitější proměnné, bez kterých se třída C_menu neobejde, patří nastavení jazyka, který webová prezentace momentálně používá do proměnné $_SESSION["lang"]. Tuto proměnnou pak aplikace používá, aby zjistila jakou jazykovou verzi menu načíst z databáze. Jako další důležité proměnné je třeba zmínit:

- $GLOBALS["OLV"]["Menu"]["tableprefix"] – Tato proměnná určuje předponu před jménem tabulky v databázi – například tabulky, se kterými se pracuje v této práci, mají předponu „dp_“.

- $GLOBALS["OLV"]["Menu"]["style"] – Tato proměnná určuje strukturu html kódu, jak se má menu vytvořit. Teoreticky by měl stačit jeden styl doporučovaný jako standard[11] a design by se měl přizpůsobit skrze kaskádové styly. Nicméně standardů je vždy několik, stejně jako způsobů, které jsou vyžadovány zadavatelem. Během roku vývoje a používání této aplikace na webových prezentacích bylo nutno přidat 2 alternativní druhy struktur. Výchozí hodnota této proměnné je 2, což je struktura tohoto typu:

```
<ul>
  <li>menu1</li>
```

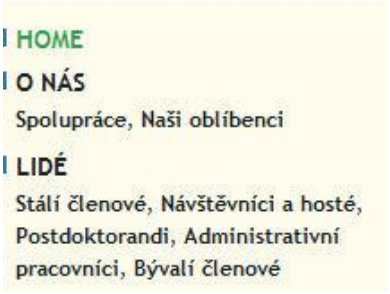
[11] Více o standardu listů v xHTML lze nalézt zde http://xhtml.com/en/xhtml/reference/ul/

```
    <li>
      menu2
      <ul><li>menu2.1</li></ul>
    </li>
  </ul>
```

- $GLOBALS["OLV"]["Menu"]["full"] - Tato proměnná určuje, zda se má načíst celé menu, nebo pouze ty části, které obsahují položky zvoleného menu a nebo jsou potomci zvoleného menu. Pokud je hodnota této proměnné „false“, načítají se vždy pouze ty položky, které mají se zvolenými položkami společného rodiče, plus děti zvolené položky menu. Viz následující příklad ze stránky, kde je aplikace s tímto nastavením použita (http://akcnitisk.cz):

Na prvním obrázku je zvolena první položka, a vidíme tedy i její potomky. Na druhém obrázku byla zvolena druhá položka a následně její podpoložka, a tím se nám potomci první položky z nabídky schovali.

Pokud je tato proměnná nastavená na „true“, načítá se celé menu. Je pak buď zobrazované a schovávané pomocí javaskriptu či css, nebo je prostě vypsané celé. Tento styl menu je použit například na stránce Centra pro teoretická studia (http://cts.cuni.cz/):



- $GLOBALS["OLV"]["menu"]["div"] – Na výše uvedeném obrázku je vidět i použití této proměnné. Hodnota této proměnné se vepisuje mezi jednotlivá menu. Na obrázku výše je tato hodnota ‘, ‘.
- $GLOBALS["OLV"]["Menu"]["home"] - Tato proměnná by měla obsahovat hodnotu atributu index_menu položky menu, která se načítá jako výchozí, pokud uživatel nezvolí žádné menu.

Potřebné proměnné jsou blíže popsány též v souboru třídy, včetně toho jak nastavit proměnné do polí, pokud je na jedné stránce nabídek více. Jsou tam také popsána dodatečná volitelná nastavení.

Načtení menu

Pokud jsou splněné výše vyjmenované předpoklady, je chod třídy při vypisování menu následující: třída je při vytváření obsahu webové aplikace načtena a je vytvořena její instance.

```
include ("cesta_k_souboru/C_Menu.php");

$this->C_menu = new C_Menu();
```

Při vytvoření instance třídy se spustí konstruktor třídy a ten načte z proměnných od uživatele (tj. z $_GET či $_POST) informaci o tom, jaké menu je zvoleno. Výchozí pro posílání hodnoty indexu menu je proměnná „m“ či menu, nicméně jméno proměnné lze nastavit přes volitelné nastavení třídy. Pokud není proměnná od uživatele k dispozici, načte konstruktor jako zvolenou položku menu tu položku, která je nastavená jako výchozí v nastavení třídy. Následně konstruktor zavolá funkci find_selected_menu. Tato funkce načte z databáze cestu od vybrané položky menu do vrcholu daného menu. Načítá a ukládá tedy do proměnné informace o rodičích, dokud nenarazí na položku menu, která v atributu „parent“ nemá vyplněno nic. Kontroluje při tom také to, zda jsou stránky dostupné ve

zvoleném jazyce, a případně vrátí zprávu o nedostupnosti dané stránky. Aplikace pracuje s jazyky tak, že porovnává hodnotu v atributu „lang“ s hodnotou v proměnné SESSION["lang"]. K dispozici je navíc pro usnadnění práce zástupná hodnota „all“. Pokud je v tabulce jako atribut „lang“ nastavena tato hodnota, zadaná jazyková mutace je platná pro jakýkoli jazyk. Kontroluje také, zda není první zvolená položka menu označená jako „skip“. V tom případě automaticky načte jejího prvního potomka jako nové zvolené menu. Zároveň se ověřuje, zda má uživatel právo k menu přistupovat. Kontroluje se tedy, jestli není menu označené jako „inactive“ či zda má nastavené a dodržované „auth_keys“. Pokud některé z těchto ověření selže, je zobrazena chybová hláška a je načtena výchozí položka menu. Pokud funkce načte všechny zvolené položky menu úspěšně, uloží je do pole, které je potom přístupné pro další zpracování.

Jednotlivá menu či části menu se pak dají načíst voláním funkce „load_menu“. Tato funkce má návratovou hodnotu v podobě vlastních již vytvořených menu. Funkce se dá volat takto:

```
$menu  =  $this->C_menu->load_menu(1);
```

Tento příkaz vrátí do proměnné „menu“ všechny položky nabídky, které mají jako vrchol stromu položku menu s indexem 1. Tato funkce načte nastavující proměnné a zjistí, zda je v daném menu nějaká položka zvolená. Pokud ano, zapamatuje si jí a následně jí při vypisování menu označuje jako zvolenou. Poté zavolá funkci build_menu. Tato funkce funguje tak, že cyklem „while“ vypisuje jednotlivé položky menu, které mají stejného rodiče. Začne tím menu, které mají jako rodiče položku menu s indexem, který je zadaný funkcí load_menu. V příkladu volání výše by se tedy začaly načítat položky menu, které mají jako parent nastavenou hodnotu „1“. Při načítání se kontroluje index načtené položky menu s indexy zvolených položek, a pokud je shoda, je menu ve výpisu příslušně označeno. Pokud se zvolí načtení full_menu, tak funkce build_menu při každé načtené a zaznamenané položce znovu rekurzivně volá sama sebe – a to s jedinou změnou, totiž že místo hodnoty dodané funkcí load_menu používá index vždy tu položky nabídky, kterou zrovna načetl. Tím se postupně načte celé menu. Pokud je zvoleno načítání pouze momentálně vybrané části menu, je toto načítání navíc omezeno podmínkou, ale princip je stejný.

Tvorba *sitemap*

Sitemap je stránka, která by žádné webové prezentaci neměla chybět. Jedná se o soubor s informacemi o struktuře a obsahu daného souboru internetových stránek. Zobrazuje hypertextové odkazy na jednotlivé podstránky, obvykle v hierarchické struktuře. Slouží pro orientaci uživatelů, ale také usnadňuje procházení internetových stránek jednotlivých webových prezentací robotům. Je to jedna z důležitých součástí SEO (*search engine optimalization*). Tvorba *sitemap* se provádí obdobným způsobem jako načítání menu. Seznam linků do *sitemap* se automaticky vytváří a shromažďuje při běhu funkce load _menu. Z toho plyne, že při tvorbě *sitemap* by měla být samozřejmě proměnná $GLOBALS["OLV"]["Menu"]["full"] nastavena na „true“, aby se načetly všechny odkazy, a to ať už je menu ve webové prezentaci koncipováno jakkoli. Následující příklad vypíše *sitemap* v xml formátu všech položek menu ve stromu pod položkou menu 1.

```
<?
   $out .= '<?xml version="1.0" encoding="UTF-8"?>
   <urlset
xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">';
   $this->C_menu = new C_Menu();
   $x = $GLOBALS["OLV"]["Menu"]["full"];
   $GLOBALS["OLV"]["Menu"]["full"] = 1;
   $menu  =  $this->C_menu->load_menu(1);
   $GLOBALS["OLV"]["Menu"]["full"] = $x;
   if (is_array($this->C_menu->sitemap)){
      foreach ($this->C_menu->sitemap as $i){
         $out .= '<url>';
            $out .= ' <loc>'.$i.'</loc>';
         $out .= '</url>';
      }
   }
   $out .= '</urlset>';

?>
```

Drobečková navigace

Pro usnadnění orientace uživatelů by součástí jakékoli stránky s vnořenými nabídkami měla být drobečková navigace.[12] Podobně jako u *sitemap* se informace potřebné pro drobečkovou navigaci sbírají při běhu funkce load_menu. Po té co je menu načteno se pro něj může nechat vypsat drobečková navigace pouhým zavoláním funkce k tomu určené – funkce makepath. Tato funkce má jediný parametr a tím je hodnota, která má oddělovat jednotlivé odkazy. Následuje příklad volání a příklad výsledku volání ze stránky Centra pro teoretická studia (http://www.cts.cuni.cz/):

```
$this->C_menu = new C_Menu();

$menu  =  $this->C_menu->load_menu(1);

$navigace = $this->C_menu->makepath(" > ");
```

C_CMS

Druhá popisovaná aplikace, která se primárně stará o zobrazování obsahu, byla v PHP zpracována do jedné třídy nazvané C_CMS. Tato třída se nachází v přílohách ve stejnojmenném souboru. Třída odpovídá požadavkům stanovaným ve výše popsaném *use case*. Třída potřebuje samozřejmě napojení na databázi, která obsahuje výše popsané tabulky. Počítá se s napojením na databázi nikoli přímo, ale přes další třídu, která se stará o autentizaci připojení a vlastní vytváření tunelů. Stejně jako v případě výše popsaného C_Menu se o tento úkol v přiloženém kódu stará třída C_MySQL, ale lze jí nahradit jakoukoli podobnou třídou, kterou je administrátor zvyklý používat. Třída CMS nevyžaduje pro svou práci žádné další třídy. Vzhledem k univerzální práci s obsahem je však žádoucí ji používat v kombinaci s třídami jinými. Kromě toho, velká část funkčnosti této třídy je navázána na výše popsanou třídu C_Menu (i když ji nevyžaduje). Mezi třídy, jejichž funkce se dají považovat za důležité a na jejichž metody je odkázáno v kódu, patří již výše zmíněná třída pro správu uživatelských práv C_Auth a třída pro tvorbu hezkých URL C_router.

[12] Definici a úvod do drobečkové navigace lze nalézt například zde [26].

Jediná nastavující proměnná, kterou C_CMS potřebuje pro svůj chod, je $GLOBALS["OLV"]["CMS"]["tableprefix"]. Stejně jako u C_Menu i tato proměnná určuje předponu před jménem tabulky v databázi – například tabulky, se kterými operujeme v této práci, mají předponu „dp_“. Pokud se využívá napojení na třídu C_Menu, třída C_CMS potřebuje, aby jí byl přednastaven jazyk, který má momentálně webová prezentace používat. Tato informace se ukládá do stejné proměnné jako pro třídu C_Menu – tedy do $_SESSION["lang"].

Pokud jsou výše popsané požadavky pro funkčnost splněny, může být vytvořena funkční instance třídy C_CMS následujícími příkazy:

```
include ("cesta_k_souboru/C_CMS.php");

$C_CMS = new C_CMS();

$out  =  $C_CMS->load_template();
```

Tyto příkazy nejprve načtou třídu C_CMS, následně vytvoří její instanci v proměnné $C_CMS. Konstruktor třídy C_CMS pouze vytvoří a zaznamená do pole seznam funkcí této třídy, ale sám nic nevrací. Poslední příkaz výše uvedeného volání zavolá metodu load_template. Ta vrátí nastavený obsah stránky.

Templates

Tato podkapitola se zabývá práce s takzvanými *templates*. *Template* se dá z angličtiny přeložit jako šablona. Aby nedošlo k záměně se šablonami používanými pro vypisování obsahu připojeného k menu, používá se zde anglický výraz *template*. Hlavní funkce třídy C_CMS je načítat soubory uložené v adresáři „inc“. Každý soubor představuje jeden *template*. Při zavolání metody $C_CMS->load_template($parametr) třída zjistí, zda v adresáři „inc“ existuje soubor s názvem "template_$parametr.php". Pokud tomu tak je, načte tento soubor a vrátí jeho obsah. Pokud se v adresáři „inc“ zadaný soubor nenachází, nebo pokud není parametr při volání funkce zadán, vrátí obsah souboru template_main.php. Pokud není parametr specifikován, lze také alternativně zadat název *template* přes proměnné $_POST["template"] a $_GET["template"]. Protože načítaný soubor má koncovku .php, je zpracováván serverem, a může tedy obsahovat jakýkoli php kód.

Soubor obsahující *template* může obsahovat buď zvolenou část stránky nebo stránku celou, podle toho, jakým způsobem se s vráceným obsahem dále nakládá. Opětovným voláním

metody metody $C_CMS->load_template($parametr) lze samozřejmě načíst více *templates*.

Načítání obsahu z funkce

Do souboru s *template* je možné umístit i statický obsah. Pak se ovšem nejedná o šablonu. Primární funkce *template* je nicméně sloužit jako šablona pro celou stránku, a tedy obsahovat strukturu designu webové prezentace. *Template* tedy určuje strukturu prvků na stránce a určuje místa, do kterých se má načíst dynamický obsah, jako jsou menu, hlavní obsah, pruh s reklamami či patička stránky, a další. Následně pomocí volání příslušných metod do těchto míst obsah načítá. Příklad obsahu jednoduchého *template* souboru je například kód výše popsaný v kapitole popisující tvorbu *sitemap*.

Pro usnadnění práce je k dispozici načítání externího obsahu skrze metodu třídy C_CMS. Stará se o to metoda $C_CMS->main. Pokud má tato metoda skrze parametr natavenou proměnnou $see, zjistí, zda v třídě C_CMS existuje metoda se jménem "see_$see", anebo zda existuje globální funkce s tímto jménem. Pokud ano, vrátí její hodnotu. V této metodě lze tedy definovat předpoklady, parametry a volání další metody či metod a ty jsou vráceny do *template*. Vše toto by se dalo definovat přímo v *template*, nicméně pokud máme několik *templates* a v každém bychom museli nastavovat stále stejné složité volání, můžeme si práci usnadnit a stanovit načítání přímo ve třídě C_CMS. Volání skrze funkci $C_CMS->main zajišťuje, že volaná metoda je opravdu definovaná. Protože hlavní účel funkce „main“ je načítání obsahu připojeného k menu, první tři parametry funkce nebudeme vůbec používat. Volání funkce „main“ vypadá pro načtení obsahu z funkce "see_$see" následovně:

```
$content  =  $this->main(null, null, null, $see);
```

Lze samozřejmě skládat stránku z více *templates*, ať už paralelním voláním či načítáním jednotlivých *templates* do sebe. To se může provést skrze volání funkce $this->load_template($parametr) uvnitř jiného *template*.

Načítání jednoduchého obsahu připojeného k menu

Součástí třídy C_CMS je také načítání a vypisování obsahu z tabulky vytvořené speciálně pro tuto třídu. Počítá se s tím, že načítání obsahu je voláno z obsahu výše popsaných *templates*, avšak není to vyžadováno. Potřebné je ale vědět, obsah jaké položky z nabídky

má být načten. To je třeba specifikovat buď přímo indexem dané položky menu, a nebo (v případě, že před načítáním obsahu již bylo načteno menu) pomocí třídy C_Menu. můžeme načíst obsah pouze uvedením pořadového čísla menu při načítání (počítáno od nuly). Druhá důležitá volba je, jakou část obsahu připojeného k dané položce menu načíst. Jedním voláním vždy načteme veškerý takový obsah připojený k jedné položce menu, který má určen jednu danou kategorii. Kategorie se při načítání dat z tabulky porovnává s atributem „category“ v tabulce dp_menu_cm.

V nejběžnějším případě, kdy chceme vypsat obsah uživatelem zvolené položky menu, tak nejprve načteme třídu C_Menu a necháme si menu vypsat. V třídě C_Menu jsou tak dostupné informace o zvoleném menu. Poté nám stačí volat načtení obsahu takto:

```
$content = $this->main($category, $poradi_menu);
```

Výchozí a nejběžnější hodnota pro category je „main”. Pokud se tedy pracuje jen s jednou kategorií, parametr „category“ se nemusí zadávat. Pro pořadí menu to znamená hodnotu „0“. Pokud se ve webové prezentaci pracuje pouze s jedním menu, netřeba tento parametr zadávat. Volání by tedy vypadalo pouze takto.

```
$content = $this->main();
```

Pokud chceme načíst obsah položky menu, které není zvolené, musíme - jak již bylo zmíněno - zadat parametr obsahující identifikaci dané položky. Volání funkce pak vypadá takto:

```
$content = $this->main($category, null, $index_menu);
```

Průběh metody „main“ je následující. Pokud je parametrem při volání stanovena proměnná $see, tak metoda „main“, jak bylo již popsáno předchozí kapitole, slouží pouze ke kontrole existence a volání zadané funkce a vrací její návratovou hodnotu. Pokud proměnná $see není stanovena, načtou se obsahy korespondující se zadaným menu a kategorií. Obsahy se řadí podle atributu „sort“ z tabulky dp_menu_cm. Zároveň se stejně jako v aplikaci pro práci s menu porovnává hodnota v atributu „lang“ z tabulky „dp_menu_content_parts“ s hodnotou v proměnné „SESSION["lang"]“. K dispozici je navíc zástupná hodnota „all“ – opět nám může usnadnit práci. Pokud je v tabulce atribut „lang“ nastavena právě tato hodnota, zadaná jazyková mutace je platná pro jakýkoli jazyk. Jednotlivé obsahy a jejich části („atributy“ part z tabulky „menu_content“) se postupně načítají a zpracovávají v cyklu „while“. Vždy se shromažďují parametry obsahů a ve chvíli,

kdy se změní atribut index_menu_content, což značí, že všechny parametry byly načteny, provede se vypsání obsahu.

Obsah se zpracuje metodou pojmenovanou podle hodnoty z atributu „type“ v tabulce „menu_content“. Např. pokud je hodnota atributu „type“ „print“, zavolá se metoda load_print. Pokud metoda neexistuje, metoda „main“ vrátí chybovou hlášku. Pokud existuje, vrátí její obsah. To se opakuje pro všechny obsahy načtené z databáze.

V přiložené třídě jsou dvě základní metody pro zpracování obsahu. Lze jich nicméně přidat libovolné množství. První metoda je metoda „load_print“. Tato metoda je oproti druhé metodě velmi jednoduchá. V parametrech při volání dostane informace o položce menu, ke kterému obsah náleží, a také pole s názvem, „$db“, které obsahuje jednotlivé části obsahu. V prvním kroku se v této metodě načtou data o šabloně připojené k danému obsahu pro dané menu a daný jazyk. Poté se pomocí PHP funkce „eval“ zajistí provedení jakéhokoli PHP kódu v každé z částí načteného obsahu. Použití funkce „eval“ v aplikaci je následující:

```
eval ('$out = \''.$value.'\';');
```

V praxi to pak vypadá tak, že pokud se z databáze načte pouze normální obsah, zpracuje se klasický příkaz:

```
Hodnota $value načtená z databáze je „text“

// v eval proběhne příkaz

$out = 'text';
```

Použitím apostrofů však lze doplnit do obsahu dynamický obsah pomocí PHP. Příkaz, kterým se „eval“ provede, může tedy vypadat například takto:

```
Hodnota $value načtená z databáze je “Jazyk je nastaven na
- '.($lang=='cs'? 'česky':'anglicky').'.”

$lang = 'cs';

// v eval proběhne příkaz

$out = 'Jazyk je nastaven na – česky.';
```

Toto je samozřejmě pro správce webu velmi mocný nástroj, ale zároveň potenciální nebezpečí pro celou webovou aplikaci. Pokud by měl obsah vyplňovat někdo neoprávněný či neznalý, může snadno způsobit, že se celý obsah webové stránky bude špatně

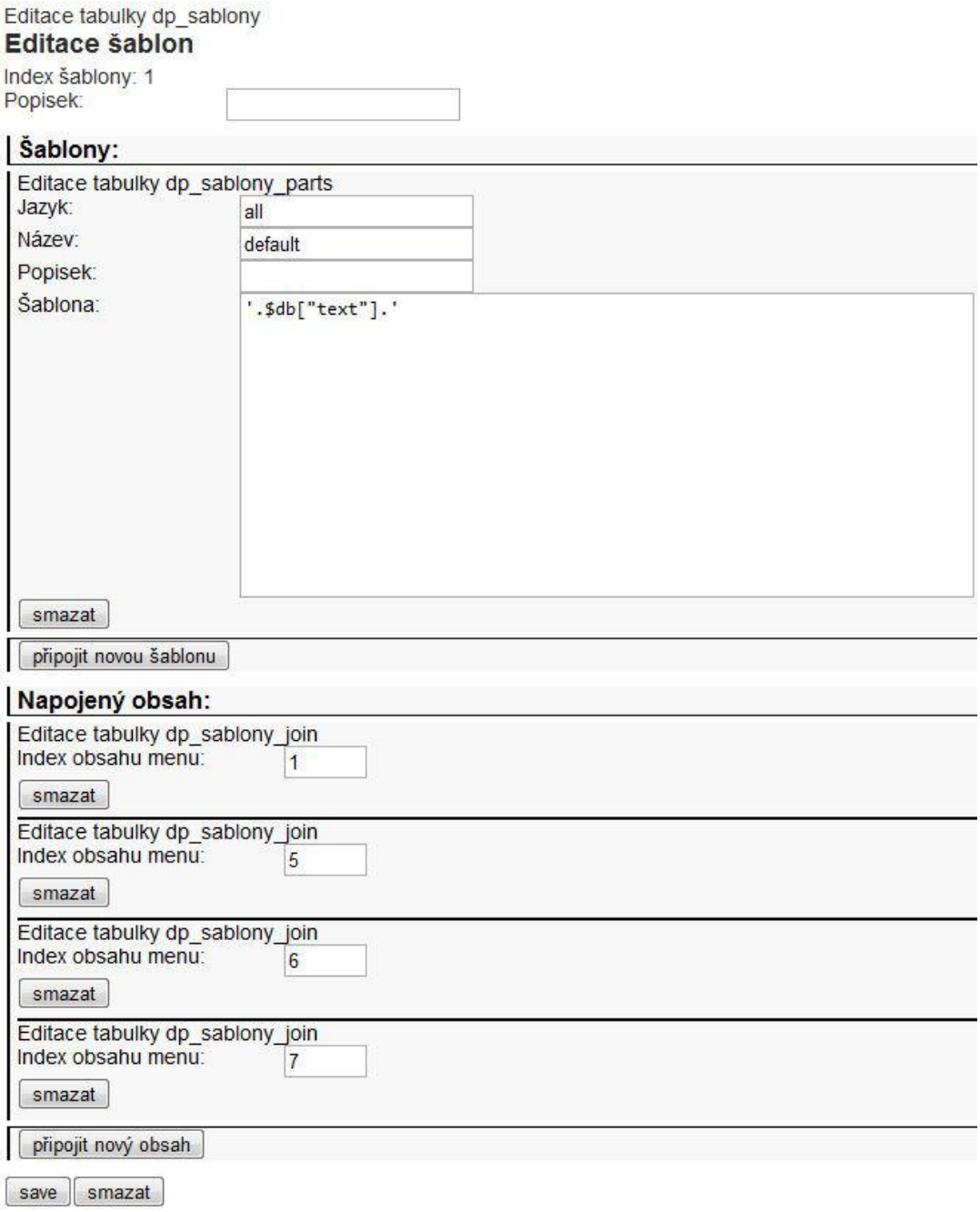
zobrazovat, případně že uživatel bude přistupovat k datům, ke kterým by jinak neměl oprávnění. Pokud je třeba zpřístupnit vyplňování obsahu ještě někomu jinému než administrátorovi, je nejlepší zamezit vyplnění kódu PHP při samotném vkládání obsahu automatickým přidáváním zpětných lomítek před apostrofy.

Po provedení PHP kódu v jednotlivých částech obsahu se provede opět funkce „eval“, ale tentokrát na šablonu. Ta obsahuje, kromě statického HTML a obsahu, který je pro všechny použití šablony stejný, výpis proměnných vlastního obsahu. Následuje příklad šablony pro obsah se dvěma částmi:

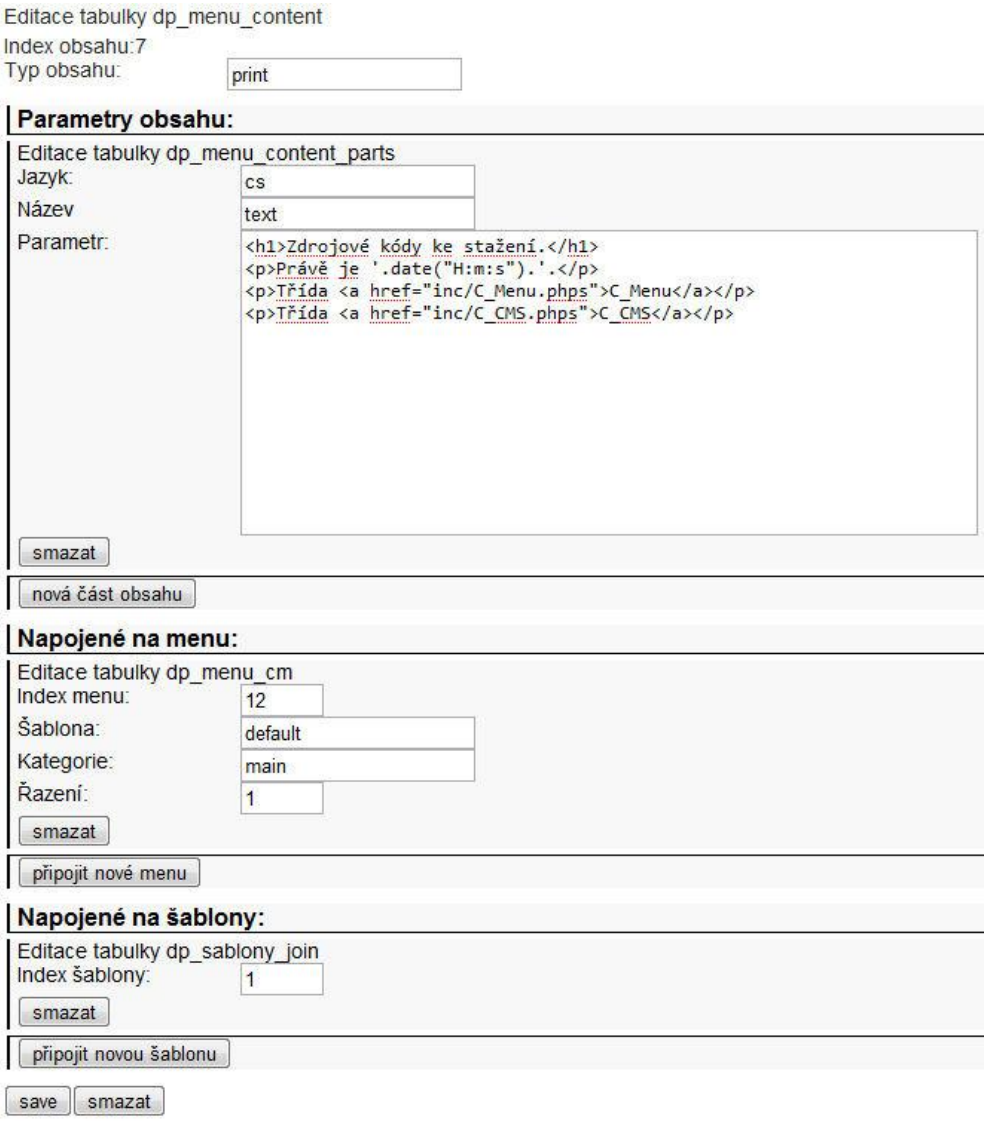
```
Hodnota šablony načtená z databáze je
„<h1>'.$db['nadpis'].'</h1>

<p>'.$db['text'].'</p>“

Hodnoty vlastního obsahu jsou

$db['nadpis'] = 'Diplomová práce';

$db['nadpis'] = 'Cíle a metodika…';

// v eval proběhne příkaz

$out = '<h1> Diplomová práce</h1>

          <p> Cíle a metodika…</p>';
```

Takto získaný výstup se vrátí metodě „main“.

Následuje názorný příklad, jak se zobrazuje statický obsah skrze šablonu. Šablona je v tomto případě nejjednodušší, jaká může být.



Z obrázku je vidět, že tato šablona je pro všechny jazyky stejná a je již napojená na čtyři různé obsahy. Tato šablona má poue jednu část. Z jejího obsahu je zřejmé, že po provedení funkce „eval“ šablona vypíše beze změny část obsahu se jménem „text“.



Následující obsah je napojen na výše uvedenou šablonu. Lze to poznat porovnáním položky Index obsahu ve druhém řádku s indexy obsahu uvedenými u formuláře editace šablony. Nebo naopak porovnáním indexu šablony vyplněného ve formuláři editace obsahu s indexem šablony uvedeným ve druhém řádku editace šablony.

Z formuláře můžeme vidět, že se obsah bude zobrazovat u menu s identifikačním číslem 12, a to na prvním místě. Vlastní obsah má pouze jednu část, a to pro český jazyk. Název této části je „text“. Pokud by název neodpovídal, v kombinaci s touto šablonou by se daný obsah nezobrazoval. Obsahová část se skládá ze statického textu doplněného PHP funkcí, která vypíše aktuální čas vypsání obsahu. Výsledek, jak ho vidí uživatel, lze vidět zde:

Načítání obsahu z databáze

Druhá metoda, která slouží ke zpracování načtených částí obsahu, se nazývá load_sql. Jak název napovídá, jedná se o metodu, která načítá data z dalších tabulek databáze.

Metoda se používá následujícím způsobem. Hlavní je část obsahu, která se jmenuje „sql“. Pokud je v této části nějaký PHP kód, metoda load_sql jej nejprve opět vyhodnotí pomocí funkce „eval“. Následně použije tuto část obsahu jako sql příkaz na databázi. A každý z databáze vrácený řádek si uloží skrze šablonu, jak by šlo o běžný obsah, tak jak bylo popsáno v předchozí kapitole. Po zpracování všech vrácených řádků vrátí veškerý vypsaný obsah metodě „main“.

Pro ilustraci si ukážeme další příklad z praxe. Jedná se o seznam aktuálních členů klubu na stránce PK1293 Vojnův Městec. Jak je vidět, máme zde klasický jednoduchý sql dotaz s podmínkou WHERE. Do podmínky se dynamicky vkládá aktuálně zvolený jazyk.

Index obsahu:4

Typ obsahu: sql

Parametry obsahu:

Editace tabulky pk1293_menu_content_parts

Jazyk: all

Název sql

Parametr:

```
select * from pk1293_content_lidi LEFT JOIN pk1293_content_lidi_klub
ON `index_content_lidi` = `fkey_content_lidi` and `lang` = "'.
$_SESSION[ "lang"].'" WHERE clenem_do IS NULL ORDER BY prijmeni,
jmeno
```

smazat

nová část obsahu

Šablona pro tento obsah vypadá následovně:

Jazyk: all

Název: default

Popisek: členové - seznam

Šablona:

```
<div class="topmarginsmall">
<div>
<span class="b">'
.($db["info"]?'<a href="'.($_SESSION["lang"] == "cs"?"clenove":"en/
members").'.html?show='.$db["index_content_lidi"].'" title="'.
($_SESSION["lang"] == "cs"?"zobrazit podrobnosti":"show
details").'">':"")
.$db["jmeno"]." ".$db["prijmeni"].
($db["info"]?"</a>":"").
"</span>"
.($db["klub_fce"]?" - ".$db["klub_fce"]:"").'</div>
<div>'.$db["popisek"].'</div>
</div>
```

Z obrázku je vidět, že se jedná o jednoduchou strukturu HTML. Z databáze se načítají atributy „jmeno“ a „prijmeni“. Podmínkami je ošetřeno, že se zobrazí odkaz na detail

osoby pouze tehdy, pokud má v databázi zaznamenanou hodnotu ve sloupci „Inko“, a že se zobrazí klubová funkce a popisek pouze v takovém případě, kdy jsou tyto hodnoty vyplněny. Na vlastní výpis tohoto obsahu se lze podívat na následující webové adrese http://pk1293.vojnuvmestec.cz/clenove.html.

Metoda load_sql je doplněna o řadu doplnění, která ulehčují práci a umožňují tvorbu komplexnějších obsahů. Jedná se především o možnost doplnit stránkování údajů, které se vracejí z databáze, a to tak, že nastavíme proměnnou $paging do obsahu s sql dotazem. Výsledek takového dotazu se bude zobrazovat například po maximálně deseti řádcích na stránku:

```
select * from pk1293_content_novinky,
pk1293_content_novinky_lang WHERE `index_content_novinky` =
`fkey_content_novinky` and `lang` = "'.$_SESSION["lang"].'"
ORDER BY datum desc '; $paging = '10
```

Pokud by bylo vráceno více než deset záznamů, bude na horní a dolní stranu výpisu přidán ovládací prvek pro stránkování ve tvaru „[předchozí] 1 »2« 3 4 [další]“. Praktické využití je vidět například na stránce internetového obchodu Mangaya na adrese http://www.mangaya.cz/.

Další rozšíření umožňuje přidávat další části obsahů k již použité části obsahu „sql“. Metoda „load_sql“ zpracovává také obsahy s názvy „top“ a „bottom“. Klasicky spustí PHP skripty, které se v těchto částech obsahu nacházejí, a vypíše je nad, resp. pod výpis vytvořený hlavní funkcí metody, tedy dotazem sql.

Poslední a velmi zásadní rozšíření funkčnosti metody sql je možnost tvořit poddotazy. Jedná se o připojení dalších dotazů sql, které se vykonávají pro každý řádek vrácený původním sql dotazem. K zadání těchto dotazů je použita obsahová část se jménem „sql_more“. V ní mohou být zadány dotazy v podobě co dotaz, to řádek, a začínat musí jedním či více znaky „x“. Podle toho, kolik je znaků x, je daný dotaz zanořený.

```
x SELECT * FROM table1 WHERE fkey = '.$db['index'].'
xx SELECT * FROM table2 WHERE fkey = '.$dbin['index'].'
x SELECT * FROM table3 WHERE fkey = '.$db['index'].'
```

Výše uvedený zápis v části „sql_more“ způsobil, že by se pro každý řádek vrácený normálním dotazem z části „sql“ provedl každý z výše uvedených příkazů, který začíná

jedním x. Navíc pro každý výsledek prvního příkazu z „sql_more“ by se provedl ještě příkaz se dvěma x. V příkladu je vidět podmínka na cizí klíče. Není samozřejmě nutná, ale většinou se tímto způsobem načítají relevantní data, takže v praxi se téměř vždy uplatní. Platí, že v poli „$db“ je uchovávána hodnota vrácená původním dotazem z části obsahu „sql“. V poli „$dbin“ jsou pak vrácené hodnoty dotazu přímo nadřazeného. V našem příkladu je to tedy hodnota proměnné „$dbin['index']“ hodnota vrácená prvním dotazem z obsahu „sql_more“. Tedy dotazem na tabulku „table1“. Protože je toto načítání v C_CMS realizováno rekurzivním voláním té samé metody, může být takto připojených poddotazů teoreticky neomezené množství. Veškeré vrácené hodnoty se ukládají do pole „$db“, a to v následujícím tvaru: vrácená hodnota prvního vnořeného příkazu je $db["db1"], druhého $db["db2"] atd.. Konkrétní vrácené řádky jsou pak indexovány od nuly – tedy například $db["db1"][0]. A k jednotlivým atributům se jde dostat přidáním asociativního klíče $db["db1"][0]["nazev"]. K hodnotě vrácené prvním vnořeným poddotazem (tedy dotazem začínajícím „xx“) pod prvním poddotazem (tedy dotazem začínajícím „x“) se pak jde dostat logicky - $db["db1"][0] ["db1"][0]["hodnota"]. Práci se šablonami pro takto složitý obsah usnadňují dvě metody. První je metoda „pa“. V ní se zadá pole výsledků – například $db["db1"], parametr obsahující šablonu a parametr obsahující rozdělovač; metoda pak vrátí zpracované výsledky. Následující praktický příklad ze stránky Centra pro teoretická studia používá metodu „pa“ k vypsání jmen autorů publikace. Publikace samotná byla načtena částí obsahu „sql“, a protože může mít více autorů, a autoři jsou tedy zaznamenáni ve zvláštní tabulce, byli autoři načteni pomocí poddotazu skrze obsah „sql_more“.

Vlastní dotaz z části obsahu sql_more:

```
x SELECT * FROM cts_c_publikace_autori, cts_c_lide WHERE
fkey_lide = index_lide and fkey_publikace =
'.$db["index_publikace"].' ORDER BY sort, prijmeni
```

Část šablony, která vypisuje jména za pomocí metody „pa“.

```
.$this->pa($db["db1"], '\'.$dbin["prijmeni"].\',
\'.$dbin["jmeno"].\'', '; ').
```

Protože šablony prochází příkazem „eval“ hned dvakrát, je nutné při definování šablony v jiné šabloně nepříjemně „escapovat“ apostrofy – v případu je to vidět. Nicméně pokud je třeba vypsat pouze dva atributy, jako v tomto případě, je tento zápis použitelný. Poslední

parametr funkce „pa“ je „rozdělovač“, který udává, co vložit mezi jednotlivé vrácené výsledky. Metoda je vytvořena tak, že pokud se vrátí pouze jeden výsledek, rozdělovač se nezobrazí. Výše uvedený příklad by tedy mohl k jedné publikaci vypsat například dva autory tímto způsobem „Ajvaz, Michal; Havel, Ivan M.“.

Druhá metoda, která se dá použít na zpracování výsledků, které byly vytvořené částí obsahu „sql_more“ je metoda „pa2s“. Tato metoda má téměř identické volání s předchozí metodou. Jediným rozdílem je, že místo parametru obsahující přímo vypsanou šablonu má parametry dva, a to parametr obsahující index šablony již existující v databázi a parametr obsahující název té části dané šablony, kterou chci použít. Tato metoda je mnohem příjemnější na používání a jdou s ní snadno vytvářet složité výpisy poddotazů. Je ale vždy nutné mít připravenou šablonu v databázi. Volání metody pa2s může tedy být následující:

```
$this->pa2s($db["db1"], 1, 'default')
```

Praktický příklad použití metody „pa2s“ se nachází v další kapitole.

Složitější příklady využití

V této kapitole je popsáno několik zajímavých praktických využití výše popsaných aplikací. To by mělo dopomoci k pochopení funkcí a možnosti navrhnutých aplikací. První příklad je ze stránky internetového obchodu http://www.mangaya.cz/ a týká se práce s rozděleným menu na stránce.

Obsah stránek je strukturován následovně: Menu je sice jedno, ale je rozděleno do dvou částí. První část menu, která obsahuje nabídku knih k prodeji, je umístěna v levé části stránky. Druhá část menu, která obsahuje informace o firmě, je umístěna nenápadněji v horní pravé části. Je toho dosaženo tak, že je nastavena jen jedna proměnná obsahující informaci o tom, které menu je zvoleno; a je také nastavena jedna proměnná určující výchozí položku menu. Jsou však dvě menu, a tedy i dvě sady položek menu, které se načítají a od kterých se odvíjí všechny položky menu ve stromové struktuře pod nimi. Realizace v kódu je zcela jednoduchá:

```
$menu1  =  $this->C_menu->load_menu(2);

$menu2  =  $this->C_menu->load_menu(1);
```

Načtou se dvě menu a ty se následně v *template* vypíší na dvě určená místa. Při vlastním vkládání položek menu je pak jen vždy třeba správně vyplnit parent a tím určit, zda bude

daná položka menu ve stromu pod položkou s indexem 1 či 2 – podle toho se bude zobrazovat na stránce.

Druhý praktický příklad se týká složitějšího načítání obsahu z databáze na stránce Centra pro teoretická studia http://www.cts.cuni.cz/. Konkrétně se následující odstavce budou zabývat fungováním načítání obsahu pro detailní informace o jednotlivých zaměstnancích.

Struktura načítaných dat je následující. V tabulce „cts_c_lide“ jsou informace o jednotlivých osobách. Každá osoba může k sobě mít přiřazené fotky z tabulky obrázků „cts_c_img“. Každá osoba může mít svoje granty, které jsou zaznamenané v tabulce „cts_c_granty“. Protože grant může mít více lidí najednou, jsou tyto granty napojeny na tabulku osob asociační tabulkou „cts_c_granty_lide“. Každá osoba může mít svoje publikace v tabulce „cts_c_publikace“. Opět se na nich může autorsky podílet s dalšími osobami, a je tedy vytvořena asociační tabulka „cts_c_publikace_autori“.

Cílem je vypsání detailu stránky, na kterém by kromě běžných údajů o osobě byly vypsány všechny granty a publikace, na nichž se dotyčný podílí, stejně jako jeho fotogalerie. Vytvoření takovéhoto složitého výpisu lze provézt díky popisované aplikaci následovně.

Vytvoří se obsah typu sql a napojí se na příslušné menu. V našem případě to je menu nazvané „Detail osoby“. Toto menu není viditelné běžnému uživateli, ale dá se na něj dostat kliknutím na odkaz „detail osoby“ v seznamu stálých členů. V daném obsahu se vytvoří obsahová část s názvem (atribut „name“) „sql“. Její obsah bude následující:

```
select * from cts_c_lide where index_lide =
'.(is_numeric($_GET["osoba"])?$_GET["osoba"]:"").'
```

PHP skript vložený do dotazu zajistí, že se načte ta osoba, na jejíž obsah se na předchozí stránce kliknulo. Zároveň použití funkce „is_numeric“ zabrání možnému nebezpečí příchodu nevhodného parametru, a tím SQL injection, protože připouští pouze číselný obsah. Dotaz tedy načte informaci o daném člověku, či v případě zadání špatného parametru selže a nevypíše nic.

Dále se vytvoří obsahová část s názvem „sql_more“, ve které budou dotazy na připojené tabulky.

```
x SELECT * FROM cts_c_lide_img, cts_c_img WHERE fkey_lide =
'.$db["index_lide"].' and fkey_img = index_img ORDER BY
img_category
```

```
x SELECT * FROM cts_c_granty,  cts_c_granty_lide WHERE
fkey_lide = '.$db["index_lide"].' and index_granty =
fkey_granty and grant_od <= "'.date("Y").'" and grant_do >=
"'.date("Y").'" ORDER BY nazev

xx SELECT * FROM cts_c_granty_lide, cts_c_lide WHERE
fkey_lide = index_lide and fkey_granty =
'.$dbin["index_granty"].' ORDER BY poradi

x SELECT * FROM cts_c_publikace, cts_c_publikace_autori
WHERE index_publikace NOT IN (SELECT fkey_publikace FROM
cts_c_publikace_kategorie_join WHERE
fkey_publikace_kategorie =10) and fkey_publikace =
index_publikace and fkey_lide = '.$db["index_lide"].' ORDER
BY rok desc, nazev

xx SELECT * FROM cts_c_publikace_autori, cts_c_lide WHERE
fkey_lide = index_lide and fkey_publikace =
'.$dbin["index_publikace"].' ORDER BY sort
```

Jak bylo řečeno, pro správné zpracování obsahu „sql_more“ je nutné, aby byly dotazy na jedné řádce a začínaly znakem „x“, který se opakuje podle počtu zanoření. Vidíme tedy, že první poddotaz nám vybírá data z tabulky obsahující obrázky. Protože chceme vybrat jen obrázky týkající se jedné dané osoby, tak má omezující podmínku, která závisí na původním dotazu. Následují poddotazy na granty a publikace. Ty jsou složitější: oba mají ještě vlastní poddotaz zpět na tabulku lidí. Je tomu tak proto, že prvním dotazem zjistíme informace o osobě, která nás zajímá. Druhým dotazem zjistíme, které má daná osoba publikace (či analogicky granty). Avšak na to, abychom vypsali seznam publikací, stále ještě nemáme dost údajů. Je totiž možné, že naše osoba na daných publikacích nepracovala sama. Proto je zapotřebí ještě třetí dotaz, resp. podpoddotaz. Ten míří zpět na tabulku lidí, aby se zjistili všichni autoři dané publikace. Tento dotaz probíhá pro každou z publikací, které jsou vráceny nadřazeným dotazem. Na stejném principu fungují výše uvedené dotazy na granty.

Po napsání dotazů již je třeba jen přidělit tomuto obsahu šablonu, přes kterou by se jejich obsah mohl vypisovat. Protože jsou použity složité poddotazy, je zde pro jejich výpis použita metoda „pa2s“. Níže je uvedena část šablony, kde se metoda volá:

```
.($db["db2"][0]["nazev"]?'<h2 class="clear">Aktuální
granty:</h2>':'').
```

```
$this->pa2s($db["db2"], 7, 'default').
($db["db3"][0]["nazev"]?'<h2 class="clear">Vybrané
publikace:</h2>':'').
$this->pa2s($db["db3"], '8', 'default').'
```

Zajímavé je použití přímého přístupu k datům, které jsou vráceny poddotazem. Aby se ověřilo, že poddotaz vrátil nějaké granty, a je tedy třeba vypsat nadpis <h2 class="clear">Aktuální granty:</h2>, musí podmínka testovat existenci názvu prvního vráceného grantu přímo skrze proměnnou $db["db2"][0]["nazev"]. Protože zde byla použita pro výpis poddotazů metoda „pa2s“, je třeba ještě vytvořit šablony pro jednotlivé položky výpisu publikací a grantů. V nich je pak možno udělat zanořené poddotazy už pouze metodou „pa“ – tak, jak byla popsána výše v práci – neboť z vnořených poddotazů je třeba vypisovat pouze jména a primární klíče. Vlastní stránka pak může vypadat například následovně:

Závěr

Hlavním cílem této práce bylo navrhnout dva funkční univerzální systémy, resp. součásti komplexního *content management* systému, které by sloužily k načítání menu a obsahu na webovou stránku. Systémy jejichž návrhem a funkcemi se zabývá tato práce byly během psaní této práce využity při tvorbě řady webových prezentací. Za zmínku stojí například webová stránka Centra pro teoretická studia (společného pracoviště Univerzity Karlovy v Praze a Akademie věd České republiky), internetový obchod knihkupectví Mangaya, nebo webové stránky Petanque klubu 1293 Vojnův Městec. Obsah a menu všech těchto stránek je kompletně řešen pomocí vypracovaných systémů. Z toho lze soudit, že navržené systémy jsou funkční a umožňují snadné využití pro řadu různorodých úloh. Pro navrhování těchto systémů, a také pro následné popsání jeho tvorby a funkčnosti, byla samozřejmě použita řada odborných přístupů. Jde v prvé řadě o principy datového modelování a normalizace, které byly použity pro návrh struktury databáze, kterou aplikace používají. A také o objektový přístup pro návrh a programování vlastních aplikací, které se starají o data. Tvorbě systémů předcházela příslušná literární rešerše. Nejprve byly prozkoumány možné postupy a nástroje, které by mohly být použity pro tvorbu systémů. A

následně z nich byly použity nejvhodnější a nejpoužívanější. Základní postupy a nástroje, které byly nakonec použité pro navrhování vytvářených systému, jsou popsány v rešeršní části této práce. Díky využití navržených systémů v praxi, bylo možné nejen systémy dodatečně odladit a doplnit jejich funkčnost,ale i obohatit tuto práci reálnými příklady a postřehy z praxe. Tím vším bylo přispěno ke splnění druhého hlavního cíle, k seznámení čtenáře s problematikou navrhování pokročilých objektových *web content management* aplikací. Protože vytvořené systémy jsou navrženy objektově a velmi univerzálně, lze je nejen snadno napojit do různých *content management* systémů, ale jde pro ně vytvořit řadu specializovanějších rozšíření. Jako příklady takovýchto rozšíření může být například doplněk pro snadné načítání galerie obrázků, nadstavba pro usnadnění práce s geografickými informačními systémy, či doplněk pro tvorbu systémů pro podporu rozhodování.

DOC. DAVID STORCH

E-mail: storch@cts.cuni.cz

Osobní webové stránky: http://www.cts.cuni.cz/~storch/

Telefon: +420 221 183 535

David Storch, Ph.D. (1970), se v Centru pro teoretická studia UK zabývá obecnou a evoluční ekologií a ekologií společenstev lesních a vodních ptáků. Přednáší na Přírodovědecké fakultě UK v Praze a Biologické fakultě JU v Českých Budějovicích. Je autorem dvou knih, řady odborných článků a častým přispěvatelem do časopisu Vesmír.

Aktuální granty:

FP7: SCALES - Securing the Conservation of biodiversity across Administrative Levels and spatial, temporal and Ecological Scales [detail]
David Storch
2009 - 2014, Evropský projekt v rámci 7RP

Limity druhového bohatství: makroekologická analýza evolučních a ekologických procesů podmiňujících diverzitu na povrchu Země [detail]
David Storch, Arnošt L. Šizling
2011 - 2014, GAČR P505/11/2387

Vybrané publikace:

Storch, David; **Zrzavý, Jan; Mihulka, Stanislav** (2009): Evolution: Ein Lese-Lehrbuch
Spektrum Akademischer Verlag, Heidelberg. [detail]

Storch, David; Šizling, Arnošt L.; Reif, Jiří; **Gaston, Kevin** (2009): Invariance in species-abundance distributions
Theoretical Ecology, 2: 89-103, doi 10.1007/s12080-008-0031-3.

Storch, David; Šizling, Arnošt L.; **Keil, Petr** (2009): Rapoport's rule, species tolerances, and the latitudinal diversity gradient: geometric considerations
Ecology, 90: 3575-3586.

Storch, David; Šizling, Arnošt L.; **Šizlingová, Eva;** Reif, Jiří; **Gaston, Kevin** (2009): Rarity, commonness and the contribution of individual species to species richness patterns
American Naturalist 174: 82-93.

Storch, David; Šizling, Arnošt L.; **Šizlingová, Eva;** Reif, Jiří; **Gaston, Kevin** (2009): Species-abundance distribution results from a spatial analogy of central limit theorem
Proceedings of the National Academy of Sciences of the U.S.A., 106: 6691-6695.

Storch, David; Šizling, Arnošt L.; **Nekola, J.C.; Boyer, A.G.** (2008): Artifactions in the Log-transformation of Species Abundance Distributions
Folia Geobotanica, 43:259-268, doi: 10.1007/s12224-008-9020-y.

Storch, David; Šizling, Arnošt L. (2008): The concept of taxon invariance in ecology: Do diversity patterns vary with changes in taxonomic resolution?
Folia Geobotanica, 43:329-344, doi: 10.1007/s12224-008-9015-8.

Storch, David; Šizling, Arnošt L.; Reif, Jiří; **Polechová, Jitka; Šizlingová, Eva; Gaston, Kevin** (2008): The quest for a null model for macroecological patterns: geometry of species distributions at multiple spatial scales
Ecology Letters 11: 771-784.

Storch, David; Šizling, Arnošt L. (2007): Geometry of species distributions: Random clustering and scale invariance
In: *Scaling Biodiversity* (eds. Storch D., Marquet P.A.& Brown J.H.], Cambridge University Press, Cambridge.

Storch, David; **Marquet, Pablo A.; Brown, J.H.** (2007): Scaling biodiversity
Cambridge University Press, Cambridge. [detail]

Storch, David; Šizling, Arnošt L.; **Gaston, Kevin** (2007): Scaling species richness and distribution: Uniting the species-area and species-energy relationships
In: *Scaling Biodiversity* (eds. Storch D., Marquet P.A.& Brown J.H.], Cambridge University Press, Cambridge.

Seznam odborné literatury

[1] GOSS INTERACTIVE. 8 Things to consider when starting a WCM project (paper) [online]. Datum poslední revize 2009 [Cit. 27 .2. 2011]. Dostupné na adrese http://www.gossinteractive.com/CHttpHandler.ashx?id=254&p=0

[2] ANTHONY, Paul. 15 free CMS options for Web Design Professionals Reviewed [online]. Datum poslední revize 17. 5. 2008 [Cit. 27. 2. 2011]. Dostupné na adrese http://blog.webdistortion.com/2008/05/17/13-free-cms-options-for-web-design-professionals/

[3] MCCLURE, Steve. Object Database vs. Object-Relational Databases (paper for IDC Bulletin) [online]. Datum poslední revize 08. 1997 [Cit. 27. 2. 2011]. Dostupné na adrese http://ce.sharif.ir/courses/84-85/1/ce384/resources/root/Object%20Database%20vs_%20Object-Relational%20Databases.pdf

[4] VOSTROVSKÝ, Václav. *Vytváření databází v Oracle*. 1. vyd. Praha: Česká zemědělská universita v Praze, Provozně ekonomická fakulta, 2006.

[5] Oracle® Database Object-Relational Developer's Guide [online]. Datum poslední revize 9. 2008 [Cit. 27. 2. 2011]. Dostupné na adrese http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28371/toc.htm.

[6] Database eLearning [online]. [Cit. 27. 2. 2011]. Dostupné na adrese http://db.grussell.org/.

[7] Introduction to Data Modeling [online]. ©1994-2008, Datum poslední revize 29. 2. 2004 [Cit. 27. 2. 2011]. Dostupné na adrese http://www.utexas.edu/its/archive/windows/database/datamodeling/index.html.

[8] AMBLER, Scott W. Data Modeling 101 [online]. ©2002-2006, Datum poslední revize 19. 11. 2006 [Cit. 27. 2. 2011]. Dostupné na adrese http://www.agiledata.org/essays/dataModeling101.html.

[9] Relational Database (Wikipedia) [online]. Datum poslední revize 27. 2. 2011 [Cit. 27. 2. 2011]. Dostupné na adrese: http://en.wikipedia.org/wiki/Relational_database.

[10] WELLING, Luke a THOMSON, Laura. *PHP a MySQL - rozvoj webových aplikací*. 1.vyd. Praha: SoftPress, 2003.

[11] MARSTON, Tony. The Relational Data Model, Normalisation and Effective Database Design [online]. ©2004, Datum poslední revize 12. 8. 2005 [Cit. 27. 2. 2011]. Dostupné na adrese http://www.tonymarston.co.uk/php-mysql/database-design.html.

[12] Codd's 12 Rules (Wikipedia) [online]. Datum poslední revize 27. 2. 2011 [Cit. 27. 2. 2011]. Dostupné na adrese http://en.wikipedia.org/wiki/Codd's_12_rules.

[13] Data Management Reference - Concepts and Techniques [online]. ©1998-2005, Datum poslední revize 2. 5. 2005 [Cit. 27. 2. 2011]. Dostupné na adrese: http://www.datamodel.org/reference.php.

[14] Database Normalization (Wikipedia) [online]. Datum poslední revize 37. 2. 2011 [Cit. 27. 2. 2011]. Dostupné na adrese http://en.wikipedia.org/wiki/Normal_forms.

[15] MySQL Developer Zone [online]. ©1995-2008 [Cit. 27. 2. 2011]. Dostupné na adrese http://dev.mysql.com/.

[16] WILLIAMS, Hugh E a LANE, David. PHP a MySQL: *Vytváříme webové databázové aplikace*. 1. vyd. Praha: Computer Press, 2002.

[17] DUBOIS, Paul. *MySQL profesionálně*. 1. vyd. Praha: Mobil Media a.s., 2003.

[18] KOSEK, Jiří. *PHP, tvorba interaktivních internetových aplikací*. 1. vyd. Praha: GRADA Publishing, 1999.

[19] CASTAGNETTO, Jesus, RAWAT, Harish, SCHUMANN, Sascha, SCOLLO, Chris a VELIATH, Deepak. *Programujeme PHP profesionálně*. 1. vyd. Praha: Computer press, 2001.

[20] MACH, Jakub. *PHP pro úplné začátečníky*. 1. vyd. Praha: Computer Press, 2003.

[21] PHP Hypertext preprocessor [online]. ©2001-2008, Datum poslední revize 27. 2. 2011 [Cit. 27. 2. 2011]. Dostupné na adrese: http://www.php.net/.

[22] BRÁZA, Jiří. *PHP 4 - praktické příklady*. 1. vyd. Praha: GRADA Publishing, 2003.

[23] KOFLER, Michael, ÖGGL, Bernd. *PHP 5 a MySQL 5 - Průvodce webového programátora*. Computer Press, 2007.

[24] COLEMAN, Derece. A Use Case Template [online]. Datum poslední revize 19. 6.1998 [Cit. 27.2. 2011]. Dostupné na adrese http://www.bredemeyer.com/pdf_files/use_case.pdf

[25] SOMMERVILLE, Ian. *Software Engineering 6th edition*. 6. vyd.: Addison Wesley, 2000.

[26] COLTER, Angela. User Mental Models of Breadcrumbs [online]. Datum poslední revize 13.1.2010 [Cit. 27.3.2011. Dostupné na adrese]http://www.angelacolter.com/breadcrumbs/.

[27] KONOPÁSEK, Jakub. Databáze publikací a použití PHP. Praha: Česká zemědelská univerzita,

Katedra informačních technologií, 2008. 53 s. Vedoucí diplomové práce Ing.Marek Čandík.

Přílohy

Navržené PHP aplikace jsou dostupné v nejaktuálnější verzi ke stažení na těchto adresách:

http://dp.mangaya.cz/C_menu.phps

http://dp.mangaya.cz/C_CMS.phps

Zdrojový kód třídy C_MySQL na kterou je používána ve výše uvedených aplikacích používána pro přistupování k databázi.

http://dp.jossss.zvyk.com/sources/C_MySQL2.phps

Následující webové stránky používali navržené systémy v době dokončení této práce k načítání menu a obsahu. Řazeny jsou podle doby kdy začali systémy používat.

Webové stránky petanque klubu 1293 Vojnův Městec (http://pk1293.vojnuvmestec.cz/).

Webové stránky Centra pro teoretická studia, společné pracoviště Univerzity Karlovy v Praze a Akademie věd České republiky (http://www.cts.cuni.cz/).

Internetový obchod knihkupectví Mangaya (http://mangaya.cz/).

Webové stránky firmy Akční tisk (http://akcnitisk.cz/).

Webové stránky firmy Eko-consult (http://eko-consult.cz/).

Internetový portál auta v akci (http://www.autavakci.cz/).

SQL dotaz pro vložení navržených tabulek do databáze:

```
--

-- Table structure for table `dp_menu`

--


CREATE TABLE IF NOT EXISTS `dp_menu` (

`index_menu` int(4) NOT NULL,

`parent` int(4) DEFAULT NULL,

`sort` int(3) DEFAULT NULL,
```

```
`invisible` varchar(3) COLLATE utf8_czech_ci DEFAULT NULL,
`inactive` varchar(3) COLLATE utf8_czech_ci DEFAULT NULL,
`skip` varchar(3) COLLATE utf8_czech_ci DEFAULT NULL,
`auth_keys` varchar(300) COLLATE utf8_czech_ci NOT NULL,
PRIMARY KEY (`index_menu`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;

-- --------------------------------------------------------

--
-- Table structure for table `dp_menu_cm`
--

CREATE TABLE IF NOT EXISTS `dp_menu_cm` (
`fkey_menu` int(4) NOT NULL,
`fkey_menu_content` int(7) NOT NULL,
`sablona` varchar(200) COLLATE utf8_czech_ci NOT NULL,
`category` varchar(30) COLLATE utf8_czech_ci NOT NULL,
`sort` int(3) DEFAULT "1",
PRIMARY KEY (`fkey_menu`,`fkey_menu_content`),
KEY `fkey_menu` (`fkey_menu`),
KEY `fkey_menu_content` (`fkey_menu_content`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;

-- --------------------------------------------------------

--
-- Table structure for table `dp_menu_content`
--
```

```
CREATE TABLE IF NOT EXISTS `dp_menu_content` (
`index_menu_content` int(7) NOT NULL,
`type` varchar(50) COLLATE utf8_czech_ci NOT NULL,
PRIMARY KEY (`index_menu_content`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;

-- --------------------------------------------------------

--
-- Table structure for table `dp_menu_content_parts`
--

CREATE TABLE IF NOT EXISTS `dp_menu_content_parts` (
`fkey_menu_content` int(5) NOT NULL,
`lang` varchar(3) COLLATE utf8_czech_ci NOT NULL,
`name` varchar(200) COLLATE utf8_czech_ci NOT NULL,
`part` text COLLATE utf8_czech_ci,
PRIMARY KEY (`fkey_menu_content`,`lang`,`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;

-- --------------------------------------------------------

--
-- Table structure for table `dp_menu_lang`
--

CREATE TABLE IF NOT EXISTS `dp_menu_lang` (
`fkey_menu` int(4) NOT NULL,
```

```
`lang` varchar(5) COLLATE utf8_czech_ci NOT NULL,
`name` varchar(100) COLLATE utf8_czech_ci NOT NULL,
`title` varchar(2000) COLLATE utf8_czech_ci DEFAULT NULL,
`description` varchar(500) COLLATE utf8_czech_ci NOT NULL,
`keywords` varchar(500) COLLATE utf8_czech_ci NOT NULL,
PRIMARY KEY (`fkey_menu`,`lang`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;

-- --------------------------------------------------------

--
-- Table structure for table `dp_sablony`
--

CREATE TABLE IF NOT EXISTS `dp_sablony` (
`index_sablony` int(6) NOT NULL,
`popisek` varchar(200) COLLATE utf8_czech_ci DEFAULT NULL,
PRIMARY KEY (`index_sablony`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;

-- --------------------------------------------------------

--
-- Table structure for table `dp_sablony_join`
--

CREATE TABLE IF NOT EXISTS `dp_sablony_join` (
`fkey_sablony` int(5) NOT NULL,
`fkey_table` int(6) NOT NULL,
```

```
`for_tbl` varchar(100) COLLATE utf8_czech_ci NOT NULL,
PRIMARY KEY (`fkey_sablony`,`fkey_table`,`for_tbl`),
KEY `fkey_table` (`fkey_table`,`for_tbl`),
KEY `fkey_sablony` (`fkey_sablony`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;

-- --------------------------------------------------------

--
-- Table structure for table `dp_sablony_parts`
--

CREATE TABLE IF NOT EXISTS `dp_sablony_parts` (
`fkey_sablony` int(6) NOT NULL,
`nazev` varchar(200) COLLATE utf8_czech_ci NOT NULL,
`lang` varchar(3) COLLATE utf8_czech_ci NOT NULL DEFAULT
"all",
`sablona` text COLLATE utf8_czech_ci,
PRIMARY KEY (`fkey_sablony`,`nazev`,`lang`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;
```