Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Systém pro demonstraci vyhodnocování SQL dotazů

Plzeň, 2014 Petr Mazánek

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 24. 6. 2014 Mazánek Petr

Abstract

A System for Demonstration of Evaluation of SQL Queries

This bachelor thesis deals with an introduction to SQL language focusing on all variants of SELECT statement. It also deals with an introduction to web programming and programming of online systems.

The task of the second part of this thesis is to build an online system that will be capable to graphically demonstrate a step-by-step evaluation of specified SQL query. At the very end the system needs to be tested with couple of examples.

Abstrakt

Tato bakalářská práce se zabývá seznámením se s SQL jazykem se zaměřením na všechny varianty příkazu SELECT. Zároveň má za úkol seznámení se s programováním webů a online systémů.

V druhé části práce je úkolem vytvoření online systému, který bude schopen graficky demonstrovat postupné vyhodnocování zadaného SQL dotazu SELECT. Na závěr je nutno celý systém otestovat na několika příkladech.

Obsah

1 Úvod

Na katedře informatiky a výpočetní techniky se v několika předmětech vyučuje dotazovací jazyk SQL. Jedním z nejpoužívanějších příkazů tohoto jazyka je příkaz SELECT umožňující získání informací z tabulek databáze, aniž by došlo ke změně dat. Tento příkaz má také nejsložitější syntaxi, což značně ztěžuje jeho vyhodnocení člověkem. Cílem této práce je vytvořit online systém, pro demonstraci vyhodnocování tohoto příkazu, tak jak by jej postupně vyhodnocoval člověk. Systém by tak měl pomáhat při výuce SQL.

Motivací k tomuto úkolu pro mne byla částečná znalost programování webových aplikací, zejména práce s HTML značkovacím jazykem a programovacím serverovým jazykem PHP, která měla napomoci hladšímu průběhu tvorby práce na straně jedné a hlubšímu proniknutí do této problematiky na straně druhé.

Zbývající část práce je členěna následovně. Kapitola 2 popisuje techniky a metody používané při programování webových stránek a aplikací. Ve třetí kapitole se setkáme s databázovým jazykem SQL se zaměřením na příkaz SELECT. Následně ve čtvrté kapitole provedeme analýzu dané problematiky. Kapitola 5 uvede řešení zadané práce se zaměřením na použité technologie.

2 Programování webu

Programování webu je široká disciplína zabývající se vývojem a kódováním webových stránek, internetových obchodů, online aplikací až po moderní online informační systémy.

2.1 Apache HTTP server

2.1.1 O serveru

Apache, celým názvem Apache http server, je volně dostupný http (webový) server s otevřeným kódem, určený pro většinu dnes používaných platforem (Windows, Linux, Apple, Solaris, atd.) [1]. V dnešní době je považován za nejpoužívanější webový server [2]. Byl vytvořen a je spravován otevřenou komunitou a za dobu své působnosti se stal standardem pro vývoj dalších web-serverových platforem [2].

2.1.2 Historie a vývoj

Jeho vývoj započal mezi lety 1994 a 1995, poté co Rob McCool, zakladatel do té doby nejpopulárnějšího webového softwaru HTTPd (HTTP démon), odešel z NCSA (National Center for Supercomputer Applications). Na základech tohoto softwaru začali ostatní programátoři a webmasteři pomocí záplatování upravovat původní kód, zlepšovat funkčnost a zabezpečení a v dubnu 1995 spatřila světlo světa první verze serveru Apache. Dodnes díky velkému množství příznivců a podpůrných programátorů po celém světě, kteří ho spravují, aktualizují a hledají různé bezpečnostní díry, zůstává aktuální a bezpečný vzhledem k pravidelnému vydávání updatů a záplat [2,3].

2.1.3 Původ názvu

Existují dvě verze názorů, odkud získal Apache své jméno. První a naprosto zřejmý původ odkazuje na domorodý indiánský kmen Apačů, jimž je tento název vyjádřením úcty. Druhý původ názvu pochází z analogie ke spojení anglických slov „a patchy server“, česky by se dalo přeložit jako „záplatový server“. Přívlastek záplatový poukazuje na fakt napojení kódových záplat k základnímu kódu NCSA HTTPd 1.3, jak již bylo zmíněno v předchozím odstavci [1,2].

2.1.4 Důvod použití

Myšlenka použití právě webového serveru Apache pro tuto práci se opírá o statistiku použití různých webových serverů po celém světě, kde si tato varianta drží naprosto

dominantní pozici. Dle statistik uvedených na stránkách trends.builtwith.com používá Apache přes 55% všech serverů na světě [4].

2.1.5 Funkce webového serveru

Hlavní a nejdůležitější funkcí webového serveru je příjem a následná reakce na HTTP (Hypertextový Transportní Protokol) požadavky od klienta. Reakcí bývá většinou odpověď ve tvaru HTML (Hypertext Markup Language – Hypertextový značkovací jazyk) obsahu, případně jiný text, obrázek, či jiný soubor. Samotná odpověď má také tvar HTTP. Na začátku je hlavička obsahující trojciferný návratový stavový kód, následovaná samotným obsahem. Dle stavového kódu rozlišujeme typ odpovědi, nejčastější kódy jsou 200 (Vše v pořádku) a 404 (Stránka nenalezena). Kódy jsou rozděleny do skupin podle uvozující cifry: 2xx pro úspěch, 3xx pro problém spojený s přesměrováním, 4xx pro chybu spojenou s vyřízením požadavku a 5xx pro interní chybu serveru[5,6].

S reakcí na požadavek klienta je také spjat způsob přípravy respektive získání požadovaného návratového obsahu. Toto se týká především stavů, kdy návratový obsah existuje, čili vztahujeme se nyní k návratovým kódům skupiny 2xx. Jsou v zásadě dva typy obsahu, který je vrácen klientovi. Buď se jedná o obsah statický, jinými slovy předem připravené soubory např. ve formě HTML stránek, nebo o obsah dynamický, který je v podstatě připravován a kompilován až ve chvíli kdy je potřeba jeho zobrazení. Dynamičnost v tu chvíli spočívá v reakci na určité podněty. Může se jednat o zadání dat do formuláře a jeho odeslání na server se zpětnou reakcí, či vyžádání dat z databáze a jejich zobrazení klientovi. Tuto úlohu již ovšem nehraje sám server, nýbrž jiná podpůrná technologie, kterou v praxi vesměs nazýváme serverovým skriptovacím jazykem. Čili server obdrží požadavek, připraví požadovaný soubor, a pokud v jeho těle narazí na část skriptu, kterou je potřeba vyhodnotit, podrobí ji příslušnému skriptovacímu jazyku. Ten skript vyhodnotí a výsledek vrátí zpět serveru. Takto se pokračuje, dokud není dosaženo konce souboru a takto vyhodnocený obsah je poté vrácen klientovi jako plnohodnotný HTML kód [7].

2.1.6 Ostatní webové servery

Pro završení kapitoly ještě uvedu ostatní webové servery, které stojí za zmínku. Kromě Apache existuje ještě Internet Information Services od společnosti Microsoft [8], Nginx

od Igora Sysoeva [9], nebo GWS od firmy Google. Servery jsou uvedeny v pořadí nejčastějšího použití od nejčastějšího po nejméně častý [10].

2.2 PHP

2.2.1 Úvod do PHP

PHP je široce užívaný, otevřený, volně dostupný skriptovací jazyk. Jeho použití je zaměřeno na tvorbu internetových aplikací a dynamických webových stránek. V dnešní době je velice často používán v kombinaci se serverovým softwarem Apache a databázovým systémem MySQL. V této sestavě tvoří tzv. LAMP, nebo WAMP balíky určené pro programátory a vývojáře internetových, nebo lokálních aplikací, kde LAMP znamená Linux-Apache-MySQL-PHP konfiguraci, oproti tomu WAMP se liší pouze tím, že je zprovozněn pod Windows. Nutno dodat, že tento programovací jazyk je serverově orientovaný, čili pro jeho použití je bezpodmínečně nutná přítomnost webového serveru, zároveň je multiplatformní. Serverově orientovaný znamená, že to, co je napsáno v PHP, je na serveru vyhodnoceno a výsledek zaslán zpět klientovi většinou ve formě HTML kódu. Multiplatformita zajišťuje možnost použití na prakticky všech dnes běžně používaných operačních systémech. Mezi hlavní klady PHP jazyka patří jednoduchost pro začátečníky, ale zároveň široká škála možností pro pokročilé uživatele a profesionály [11, 12, 13].

2.2.2 Vývoj PHP

Předchůdcem PHP by se dala nazvat sada skriptů vytvořených v roce 1994 Rasmem Lerdorfem, který skrze tyto skripty chtěl zaznamenávat návštěvy svého online životopisu a tuto sadu skriptů nazval „Personal Home Page Tools“, často zkracováno na „PHP Tools“, odtud zkratka PHP. Dnes se nesetkáme s jiným názvem než PHP: Hypertext Preprocessor. Vzhledem k tomu, že se tento projekt mezi vývojáři uchytil, začala další etapa vývoje, zdokonalování a rozšiřování funkčnosti projektu. Byla zavedena možnost propojení s databází a prostředí, díky němuž mohli vývojáři začít programovat dynamické webové aplikace např. návštěvní knihy. Krátce nato upustil tvůrce od názvu PHP a přejmenoval svůj projekt na „Forms Interpreter“ zkráceně FI, neboli „Formulářový interpret“, kde již bylo možné se setkat se základními funkcionalitami, které známe v PHP dnes. V dubnu roku 1996 zakladatel sloučil oba dosavadní názvy a dal za vznik PHP/FI. Tato druhá generace implementací měla zároveň povznést dosavadní „pouhou“ sadu nástrojů mezi programovací jazyky

v pravém slova smyslu. Tento jazyk začal nabývat na popularitě a průzkum společnosti Netcraft z května roku 1998 ukázal, že téměř 60 000 domén se vykazovalo hlavičkou obsahující „PHP“, čímž dávalo najevo přítomnost PHP na hostujícím serveru. Na přelomu let 1997 a 1998 kontaktovali dva programátoři z univerzity v Tel Avivu původního zakladatele a tvůrce PHP majíc potřebu zefektivnit engine původního PHP/FI. Během své online komunikace přišli s návrhy na zlepšení současné implementace a tím se zrodila nová verze PHP 3.0, která již velmi silně připomíná dnešní generaci jazyka. Ti samí pánové začali v zimě 1998 pracovat na nové verzi jádra PHP. Jejich hlavním cílem bylo zlepšit výkon komplexních aplikací. Nová verze PHP 4.0 byla oficiálně vydána v květnu roku 2000 a vydržela až do června 2004, kdy byla nahrazena současnou verzí PHP 5.0. Dnes je PHP nainstalováno na desítkách až stovkách miliónů domén po celém světě [11, 14].

2.2.3 Funkce PHP

Jak již bylo řečeno PHP je serverovým skriptovacím jazykem. Aby bylo možné rozlišit, který kód je určený pro PHP interpret a který ne, jsou vyhrazeny speciální oddělovače. Kód PHP začíná `<?php` a končí `?>`. Cokoliv leží mimo oblast ohraničenou těmito znaky, není PHP interpretem vyhodnocováno. Jsou možné i další varianty, většinou zkrácené varianty, které také uvozují PHP kód, ovšem absolutně správně jsou první zmíněné. Co se zkrácených variant týče, může se člověk setkat například se zápisem `<? zde leží nějaký php kód ?>`, či se zápisem převzatým z ASP (Active Server Pages - serverový skriptovací jazyk vyvíjený společností Microsoft) `<% php kód %>`. Skripty ohraničené těmito značkami se, pokud budou značky na serveru povoleny, vyhodnotí stejně, jako kdyby byly ohraničeny prvně zmíněnými. Výjimečně však může nastat situace, kdy budou na serveru zakázány, a pak může nastat problém nefunkčnosti. Je proto doporučeno používat syntaxi zmíněnou jako první. Zároveň je třeba mít na paměti, že aby mohl PHP interpret vůbec k souboru obsahujícímu skripty přistoupit musí se takový soubor vyznačovat příponou `.php`. Jinými slovy příponou řekneme, že by soubor mohl obsahovat PHP skript, příslušnými znaky poté vyznačíme, odkud a kam skript sahá. Jeden soubor může obsahovat více míst se skripty [11, 12].

2.2.4 Deklarace proměnných

Jako každý programovací jazyk ani PHP se neobejde bez proměnných, které uchovávají potřebné informace, a je s nimi vhodným způsobem manipulováno. Co se typů

proměnných týče, PHP je dynamicky typovaný (někdy též označováno jako slabě typovaný) jazyk. To znamená, že při deklaraci proměnných se neurčuje typ proměnné. Typ je určen až ve chvíli, kdy je do proměnné uložena hodnota, např. textový řetězec, znak, číslo, pravdivostní hodnota nebo například pole hodnot. Deklarace proměnných stejně jako jejich volání se provádí napsáním znaku `$` před název proměnné. Chceme-li do proměnné `cislo` uložit například hodnotu `11`, provedeme následující příkaz:

```
$cislo = 11;
```

Pokud naopak chceme přečíst hodnotu proměnné a vypsat ji použijeme:

```
echo ($cislo);
```

Jak je z příkladu patrné, každý příkaz musí být zakončen středníkem. Jeho absence způsobí fatální chybu. Zde byl použit příkaz `echo ($promenna)`. Tento příkaz je velice často využívaným, neboť umožňuje vypsání proměnné, nebo jakéhokoliv textu do těla HTML dokumentu. Toho je využíváno v situacích dynamického vytváření webové stránky, kdy je možné na základě různých podmínek a návratových hodnot vypisovat odlišný HTML kód pro různé situace. PHP jazyk je case-senzitive (tj. citlivý na velikost písmen) u proměnných, nikoliv však u funkcí a tříd [11].

PHP tak jako ostatní jazyky obsahuje velké množství knihovních funkcí, které může programátor využívat. Všechny jsou velmi dobře popsány v dokumentaci k jazyku s uvedením příkladů užití. Dále je možné vytváření vlastních funkcí. Stačí napsat deklaraci funkce následovanou jejím názvem a potřebnými argumenty v kulatých závorkách oddělenými čárkami. Za argumenty následuje tělo funkce ohraničené složenými závorkami. Funkce mohou, ale nemusí obsahovat návratovou hodnotu, záleží, jak jsou použity. Příklad deklarace funkce:

```
function nazevFunkce (argumenty, oddělené, čárkami){

     tělo funkce;

}
```

Funkce je poté volána pomocí jejího názvu a předáním potřebných argumentů v kulatých závorkách. PHP od verze 3.0 také podporuje objektově orientované programování a od verze 5.0 se značně podobá programování v Javě [11, 14].

2.3 HTML

HTML (Hypertext Markup Language – hypertextový značkovací jazyk) je jazyk dlouhodobě používaný pro strukturování obsahu webových stránek.

2.3.1 Vznik a vývoj

Vznik HTML jako samostatného jazyka se datuje do roku 1991, kdy byla vyjádřena potřeba nějak rozumně sdílet informace o výsledcích výzkumu. Jazyk vznikl pod taktovkou Tima Bernerse-Leeho jako výsledek dvouletého projektu v CERNu (Evropské centrum jaderného výzkumu) zaměřeného právě na sdílení informací ve velké instituci. Software pro první verzi HTML byl napsán pro operační systém NextStep a obsahoval nejen prohlížeč, ale také editor webových stránek. Tím odpadla povinnost znát jazyk HTML. Vzhledem k tomu, že ne každý provozoval NextStep na svém PC a pro programátory prohlížečů bylo příliš složité skloubení prohlížeče a editoru do jednoho software, byla zavedena nutnost jazyk znát, která přetrvává dodnes. Jak vzrůstaly požadavky na rozšiřování možností webových stránek, vyvíjel se i jazyk HTML. Několik málo verzí bylo označeno za slepé uličky, hlavně pro svou přílišnou nekompatibilitu s předchozími verzemi. Jako první standard byl prohlášen HTML 2.0 a poté na jaře 1995 vznikl návrh standardu HTML 3.0, který byl však přílišným skokem pro programátory prohlížečů, a proto bylo původní 2.0 rozšířeno o některé prvky z 3.0 a vznikla tentokrát už pod dohledem W3C (World Wide Web Consorcium) verze 3.2, která se stala dalším standardem HTML. Jak vývoj webů pokračoval, a rostly potřeby vývojářů, vznikaly nové standardy W3C jako HTML 4.0, poté (cca dvouletý odstup) 4.01. Krátce po vydání verze 4.0 byl zaveden ještě jeden standard a to XML (eXtensible Markup Language), který slouží dodnes jako formát pro výměnu a uchovávání dat. XML ovšem ovlivnilo i další vývoj HTML. Odvozením od syntaxe XML tak vznikla specifikace XHTML 1.0. Nejednalo se ovšem o nijak zásadní změnu, zůstaly stejné značky i význam, jen se drobně odlišila syntaxe. Zpočátku se zdálo, že XHTML je ten správný směr pro další vývoj, ovšem zanedlouho po přelomu tisíciletí se ukázalo, že přechod od HTML k XHTML nemá žádný pragmatický důvod. Došlo dokonce k rozpolcení W3C konsorcia, kdy část začala vývoj XHTML 2.0, které přinášelo velmi zajímavé novinky, ovšem za cenu nekompatibility s předchozími verzemi jak HTML, tak XHTML. V roce 2007 se skupiny shodly na nepokračování ve vývoji XHTML 2.0 a začal tak vznikat základ pro HTML 5, které navazuje na verzi 4.01. V současnosti jsou již některé funkce HTML 5 dostupné a na jeho vývoji se dále pracuje [15, 16, 17].

2.3.2 Struktura HTML

Jazyk má pevně danou základní strukturu. Skládá se z deklarace `doctype`, která se uvádí na úplný začátek dokumentu a slouží k identifikaci typu dokumentu. Není úplně povinná, nicméně je vhodné ji používat hlavně kvůli faktu, že se prohlížeče na základě `doctype` rozhodují, do kterého z režimů vykreslování se přepnou. Absence `doctype` pak může znamenat nepředvídatelné chování a „rozhození" designu v různých prohlížečích. Převážně MS Internet Explorer v dřívějších verzích se dokázal velmi lišit v jednotlivých režimech, což způsobovalo rozpad vzhledu webových stránek. Za deklarací `doctype` následují tzv. *tagy* (tagem se v HTML nazývá značka v ostrých závorkách „<" a „>" nesoucí informaci o tom, jaký prvek uvozuje). Tagy mohou být buď párové, to znamená, že jedna značka tag otevírá a další tag zavírá, nebo nepárové, kdy značka sama sebe otvírá a zároveň zavírá. O tom, které tagy jsou párové a které nepárové se člověk dozví z dokumentace. U párových značek je důležité, že se nesmí křížit, neboli musí se zavřít v opačném pořadí, než v jakém byly otevřeny. Za deklarací `doctype` se otevírá tag `<html>`, který udává, že následující dokument je html-dokument. Veškerý obsah tohoto dokumentu se nachází uvnitř tohoto párového tagu, který je standardně uzavřen pomocí `</html>`. Zavření symbolizuje lomítko před názvem tagu. Obsah dokumentu typicky začíná párovým tagem `<head>`, uvozujícím hlavičku dokumentu a obsahujícím údaje o webové stránce, metadata, popis obsahu, klíčová slova, titulek stránky, odkazy na soubory kaskádových stylů a soubory klientských skriptů. Hlavička s výjimkou názvu stránky (obsaženém v párovém tagu `<title>` a zobrazeném v liště prohlížeče) neobsahuje žádnou část dokumentu, kterou by prohlížeč zobrazil. Tělo dokumentu, tedy to, co prohlížeč zobrazí uživateli, se nachází uvnitř párového tagu `<body>` [15, 17, 18]. Příklad viz Obrázek 1.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Titulek stránky</title>
  </head>
  <body>
    <h1>Nadpis první úrovně</h1>
  </body>
</html>
```

Obrázek 1: Základní struktura HTML dokumentu

Značky se dělí do skupin podle významu na strukturální, sémantické a stylistické. Strukturální značky určují strukturu dokumentu a jeho rozvržení. Jedná se například o značku odstavce, tabulky, nadpisu, odkazu, blokového prvku, řádkového prvku a podobně. Další skupinou značek jsou značky sémantické neboli významové. Ty dodávají prvku a jeho obsahu určitý význam. Umožňují zdůraznit, nebo naopak ubrat na významu určitého obsahu. Typickým příkladem je `strong` pro zdůraznění, nebo `em` pro zeslabení významu určitého úseku dokumentu (převážně textu), ale i již zmíněný nadpis z první skupiny má sémantický význam. Třetí a poslední skupinou jsou značky stylistické, které ovšem postupně vymizely, neboť byly uznány zastaralými a nahrazeny validními ekvivalenty v podobě kaskádových stylů, také proto, že současným trendem (respektive trendem poslední dekády) v tvorbě webů se stalo oddělení obsahu od vzhledu stránky, a tudíž se tyto značky staly nežádoucími. V návaznosti na předchozí rozdělení nutno podotknout, že existují značky, které jednak určují strukturu a jednak význam obsahu jako již zmíněný nadpis. Ten totiž nejenže určuje strukturu obsahu, který je mu přidružen, ale zároveň přináší význam tomuto obsahu. Tím se již lehce dotýkám sémantiky tvorby webů a seo-optimalizace (optimalizace pro vyhledávače), která s použitím tagů a zejména se správným použitím strukturálních a sémantických tagů úzce souvisí. Správné použití sémantiky, stejně jako vymezení struktury dokumentu totiž mají zásadní dopad na prohledávání webu roboty vyhledávačů (Seznam, Google, Yahoo!, Bing a další). Klíčovou roli v optimalizaci hraje jednak validita (správnost, bezchybnost) dokumentu, jednak logická návaznost strukturálních značek (použití titulku stránky se správnými klíčovými slovy, návaznost nadpisů jednotlivých úrovní), a také výskyt klíčových slov uvnitř dokumentu s použitím doplňujících sémantických značek a alternativních textů. Čili není prioritní pouze vytvářet graficky hezký design stránky, ale i významově a strukturálně „hezký“ kód stránky [15, 19].

2.3.3 Syntaxe jazyka

Něco málo k syntaxi již bylo řečeno. Tagy se označují ostrými závorkami, většina tagů je párových, některé nepárové, je třeba dodržovat strukturu tagů a nekřížit je. Další důležitou vlastností tagů je, že mohou obsahovat dodatečné informace. Říkáme jim atributy a každý atribut má přidruženou hodnotu uzavřenou do uvozovek bez ohledu na její typ (číslo, řetězec apod.). Známým příkladem je tag odkazu. Pokud na stránce specifikujeme část textu jako odkaz, musíme zároveň použít atribut reference, aby

odkaz věděl, kam vlastně odkazuje, a tedy kam webový prohlížeč přejde po kliknutí na tento odkaz. Značkou `<a>` specifikujeme, že se jedná o odkaz a atributem `href`, kam odkaz směřuje. Následuje obsah zobrazený jako odkaz a ukončení tagu `</a>`.Na Obrázku 2 je názorně předveden tento zápis.

```
<a href="www.odkaz-na-stranku.cz">Zde bude text odkazu zobrazený
na stránce</a>
```

Obrázek 2: Příklad zápisu odkazu

Různé tagy mají různé atributy, které mohou nabývat vícero hodnot. Atributů může být v jednom tagu i více, přičemž některé jsou nepovinné a tag plní svou funkci i bez nich. Například tag odkazu `<a>` zmíněný výše může navíc obsahovat atribut `title` s popisem odkazu, který se zobrazí jako bublinová nápověda [15].

Každý HTML dokument se vyznačuje příslušnou příponou `.html`. Pro jeho zobrazení je potřeba speciálního interpretu nazývaného prohlížeč, který ví, jakým způsobem má daný dokument a jeho jednotlivé prvky zobrazit. Ve skutečnosti tedy uživatel nevidí jednotlivé tagy, ale graficky interpretovanou webovou stránku. To jakým způsobem se budou jednotlivé prvky vykreslovat, záleží na povaze prvku a také na použití příslušných atributů. Rozhodující vlastnosti vzhledu poté již řídí kaskádové styly (umístěné odděleně od obsahu převážně v externích CSS souborech) buď na základě definice vzhledu jednotlivých prvků, nebo použitím tříd a identifikátorů [15]. O tom však více v následující kapitole.

2.4 CSS

CSS (Cascade Style Sheet - Kaskádové styly) je rozšíření HTML jazyka o grafické formátování HTML-elementů. Potřeba grafického formátování vznikla, když bylo HTML rozšířeno z původně vědecké do komerční sféry. Před příchodem CSS se formátování provádělo pomocí několika atributů přímo v HTML značkách, čímž se kód stával nepřehledným, poněvadž se místo struktury a sémantiky kódu do HTML vkrádalo grafické upravování vzhledu. Zároveň bylo toto použití velmi neflexibilní a nepřenosné, stejně jako pracné na úpravu. Kaskádové styly přinesly vyřešení těchto nešvarů, poněvadž umožňovaly stylovat prvky mimo standartní HTML kód a s jejich příchodem se rovněž zásadně rozšířily možnosti grafických úprav. Tento stylopis se stal zároveň přenosným na ostatní stránky v rámci celého webu[15, 20].

2.4.1 Syntaxe kaskádových stylů

Zápis stylu se provádí v závislosti na použitém druhu stylování (viz následující podkapitola). V zásadě se jedná o výběr selektoru následovaný blokem atributů s hodnotami. Selektorem specifikujeme, jaký HTML prvek budeme stylovat a atributy s hodnotami definujeme požadované formátování. Konkrétnější příklady budou uvedeny u jednotlivých možností stylování (viz kapitola 2.4.2) [21].

2.4.2 Možnosti stylování

Možnosti, jak nastylovat HTML dokument nebo jednotlivé prvky, jsou celkem tři. V následujících odstavcích budou shrnuty a pomocí příkladů bude demonstrováno použití jednotlivých možností k dosažení stejného výsledku.

První způsob se týká formátování pouze jediného prvku řádkovým zápisem přímo v HTML tagu, kde použijeme HTML atribut `style` následovaný CSS atributy s hodnotami pro požadovaný vzhled prvku [21]. V příkladu na Obrázku 3 by byl text odkazu, pomocí atributu `text-decoration` a hodnoty `none`, zbaven dekorace podtržení textu a byl by zobrazen jako běžný text, zároveň by mu byla nastavena barva na černou použitím atributu `color`. Atributy jsou mezi sebou odděleny středníkem.

```
<a href=“www.stranka.cz“ style=“text-decoration:none;
color:black;“>Odkaz</a>
```

Obrázek 3: Příklad řádkového zápisu stylu

Další možností, jak nastylovat HTML dokument, je specifikování stylů v hlavičce HTML dokumentu (viz Obrázek 4). Tímto zápisem dosáhneme stejného výsledku jako v prvním případě s tím rozdílem, že se styl vztáhne na všechny prvky v HTML dokumentu.

```
<html>
  <head>
    <style type=“text/css“>
      a {text-decoration:none; color:black}
    </style>
  </head>
```

Obrázek 4: Zápis stylu v hlavičce

V tomto zápisu se již setkáme s použitím hlavních funkcí a možností „Kaskádových stylů“. Můžeme totiž kaskádovitě definovat posloupnost prvků jako selektorů a tím určit jejich vzhled. Zároveň se tento zápis dá rozšířit o použití identifikátorů a tříd. Identifikátor i třídu je nutné přiřadit HTML prvku skrze atributy `id`, nebo `class` a CSS selektor poté doplnit o patřičnou notaci. Jako selektor identifikátoru se používá znak # a pro třídu znak `.`(tečka). Rozdíl mezi identifikátorem a třídou je v rozsahu použití. Identifikátor musí být v rámci jedné stránky jedinečný (může být použit pouze u jednoho HTML elementu), zatímco třídu je možno opakovat [21]. V příkladu na Obrázku 5 je demonstrována kombinace užití všech tří zmíněných možností zápisu selektoru. Použitý selektor by platil pro všechny položky (`li`) nečíslovaného seznamu (`ul`) třídy `list1` nacházející se uvnitř prvku s identifikátorem `main`.

```
#main ul.list1 li{float:left;}
```

Obrázek 5: Kombinace různých možností zápisu selektoru

Třetí způsob a zároveň i doporučený a nejčastěji používaný je připojení externího CSS souboru v hlavičce dokumentu. Obsah souboru vypadá stejně jako deklarace stylů v hlavičce dokumentu (viz předchozí odstavec) a jsou zachovány všechny vlastnosti selektorů, ale na rozdíl od deklarace stylů v hlavičce je toto použití mnohem přenosnější, neboť všechny webové stránky obsahují pouze odkaz ne tentýž CSS soubor a tudíž následná úprava stylů se provede pouze jednou v daném CSS souboru, ale aplikuje se ve všech stránkách s tímto připojeným souborem. Zároveň je zde zachována možnost stylovat pomocí kaskád, tříd a identifikátorů. Napojení souboru k dokumentu se provádí pomocí tagu `<link>` [21] (viz Obrázek 6) a obsah souboru demonstruje Obrázek 7.

```
<html>
  <head>
    <link href=“style.css“ type=“text/css“ rel=“stylesheet“ />
  </head>
```

Obrázek 6: Zápis odkazu na soubor kaskádových stylů

```
a{
  text-decoration:none;
  color:black;
}
```

Obrázek 7: Obsah CSS souboru

2.5 JavaScript

JavaScript je, objektově orientovaný, skriptovací jazyk nezávislý na platformě interpretovaný na straně klienta [22].

2.5.1 Specifikace

Jazyk se používá převážně ve webových stránkách. Často se vkládá přímo do HTML kódu, kde jím bývají ovládány různé prvky grafického prostředí, tlačítka, formulářové prvky apod. Slouží také k animacím a různým grafickým efektům. Dá se jeho použitím také upravovat obsah HTML prvků, neboť umožňuje přistupovat k objektovému modelu webové stránky (tzv. DOM – Document Object Model). Svou syntaxí se podobá jazykům C/C#/Java, na rozdíl od nich se však jedná o jazyk dynamicky (též slabě) typovaný, čili typ proměnné je určen až přiřazenou hodnotou a nikoliv při deklaraci [22, 23].

2.5.2 Použití

JavaScript, oproti serverovým jazykům (např. PHP, ASP), je prováděn výhradně na straně klienta, tudíž až ve webovém prohlížeči po vygenerování obsahu stránky serverem. Zápis skriptů se provádí mezi párové tagy `<script>`, případně do externího souboru a odkázání na tento soubor v hlavičce. Možnosti jazyka se neustále rozšiřují a zjednodušuje se jeho užívání se vznikem nových knihoven a frameworků. Velkým rozšířením možností bylo vydání knihovny jQuery na přelomu roků 2005/2006 [22, 24, 25, 26].

2.5.3 Knihovna jQuery

S vydáním knihovny jQuery se podstatně zjednodušila a zrychlila práce se samotným JavaScriptem, stejně jako rozšířily jeho možnosti. Zároveň tato knihovna umožňuje, podobně jako kaskádové styly, oddělit funkční JavaScript kód od HTML kódu [24, 25].

2.5.4 Použití jQuery

Funkce jQuery jsou dostupné po stažení souboru do adresářové struktury webu a připojení tohoto souboru k HTML dokumentu. Toto připojení se provádí stejně jako připojení jakéhokoliv jiného externího skriptového souboru. Funkce knihovny jsou velice dobře popsány v dokumentaci s uvedenými příklady pro lepší pochopení [24, 25].

3 SQL

SQL (Structured Query Language - strukturovaný dotazovací jazyk) se používá jako prostředek a standard pro manipulaci s relačními databázemi.

3.1 Historie databází

S pojmem databáze se začínáme setkávat v 60. až 70. letech dvacátého století. V té době se začala rodit potřeba nějak „rozumně“ ukládat, uchovávat a spravovat data. Zprvu byly navrženy dva databázové modely a to hierarchický a síťový.

V hierarchickém modelu byly vztahy mezi jednotlivými entitami založeny na nadřízenosti a podřízenosti. V síťovém modelu pak databáze tvořila graf, kde uzly reprezentovaly entity a hrany tvořily relace mezi jednotlivými entitami. Oba dva modely se jistou dobu používaly, nicméně měly jistá negativa zejména při potřebě implementace vazby M:N a daly tak podnět ke vzniku nového modelu (relačního), který se úspěšně používá dodnes. Kromě relačních databází však dnes známe ještě další druhy databází. Patří mezi ně objektově orientované databáze, deduktivní databáze, temporální databáze [27, 28, 29].

3.1.1 Objektově orientované databáze

Základem těchto databází je objekt, který má své atributy/vlastnosti a metody pro práci s hodnotami. Jednotlivé záznamy jsou instancemi objektu. Objektové databáze jsou vhodné pro vytváření složitých modelů. Složitost se týká zejména vztahů mezi objekty a práce s jejich metodami a daty.

Oproti relačním databázím (momentálně nejpoužívanějším) jsou náročnější při vytváření kompletní struktury objektů, zatímco pokládání dotazů je jednodušší [29].

3.1.2 Deduktivní databáze

Jsou založeny na principu logického programování. Základem jsou relace (data, základní informace a vztahy mezi entitami) a odvozovací pravidla (soubor pravidel platný pro relace). Dotazování je poté odvozováním dat a relací na základě použitých pravidel.

S tímto typem databází se běžně nesetkáme, používají se zejména v aplikacích umělé inteligence a dolování dat a pro vědecké účely [29].

3.1.3 Temporální databáze

Tyto databáze jsou v podstatě rozšířením klasických databází o časová razítka, aby bylo možné odlišovat záznamy podle stáří a porovnávat je. Jsou použitelné tam, kde je třeba sledovat vývoj dat v čase, například v bankovních systémech, pojišťovnictví na burzách apod.

Používají tzv. jazyk TSQL, jehož syntaxe je velmi podobná klasickému jazyku SQL, je ovšem doplněná o klauzule pro sledování časových informací (např. WHEN, FIRST, LAST) [29].

3.1.4 Relační databáze

Každá databáze se dá představit jako soubor dat (informací), popisujících nějakou skutečnost v reálném světě. Relační databáze se skládá z entit a relací mezi nimi. Každá entita reprezentuje určitý objekt, který chceme popsat a k popisu použijeme atributy jednotlivých entit. Relace se používají pro spojení spolu souvisejících entit. Tento spoj je zprostředkován skrze přidružené atributy (tzv. cizí klíče). Relací máme vícero typů v závislosti na tom, jaký vztah potřebujeme zachytit [28, 29].

První je relace 1:1, která jednomu záznamu v jedné tabulce přidružuje právě jeden záznam v tabulce druhé (nepoužívá se moc často). Další, velmi často užívanou, relací je 1:M, která jednomu záznamu přidružuje více záznamů z jiné tabulky. Poslední je vazba M:N, která je v podstatě spojením dvou vazeb (1:M a N:1). Jejím smyslem je zachycení stavu, kdy více záznamů jedné tabulky odpovídá více záznamům jiné (výsledkem je přechodová tabulka zprostředkovávající vazbu) [28, 29].

Aby bylo možné data ukládat a přistupovat k nim, bylo nutné vytvořit mechanizmus, který by to umožnil. Tím mechanizmem se stal právě dotazovací jazyk SQL, který je používán v různých databázových systémech (MySQL, Oracle, PostgreSQL,…) viz kapitola 3.2.

3.2 Úvod do SQL

SQL (Structured Query Language - strukturovaný dotazovací jazyk) je jazyk používaný pro manipulaci s daty v relačních databázích. Na jeho vývoji se začalo v 70. letech 20. století a za tuto dobu doznal jistých změn hlavně v oblasti bezpečnosti a integrity dat a stal se standardem pro SŘBD (Systém řízení báze dat) pracující nad relačními databázemi.

SQL patří mezi deklarativní jazyky. V praxi to znamená, že při programování aplikací využívající tento jazyk zpravidla používáme další jazyk (tzv. procedurální jazyk) jako prostředníka. Bylo uvedeno „zpravidla“, poněvadž existuje možnost obsluhovat SQL přímo z terminálu připojením na SQL-server [27]. V online systémech a webových aplikacích je však nejvhodnější, a prakticky jediné funkční řešení, použít procedurální jazyk (např. PHP, ASP,…).

3.3 Rozdělení jazyka na funkční podjazyky

Jazyk se dále dělí na několik funkčních podjazyků, z nichž každý má typické příkazy pro práci s daty. Zároveň je pro každý definovaná určitá skupina uživatelů, kteří jej nejčastěji využívají.

3.3.1 Jazyk pro definici dat - Data definition language (DDL)

Jazyk pro definici dat se používá zejména pro tvorbu a návrh databázových modelů, tabulek, atributů, relací mezi tabulkami. Nejčastějšími uživateli jsou návrháři a tvůrci datových modelů. Obsahuje následující příkazy:

- CREATE – vytváří objekty v databázi (celou databázi, tabulky a další)
- DROP – odstraňuje objekty databáze
- ALTER – umožňuje modifikovat již vytvořené objekty databáze
- TRUNCATE – vymaže všechny záznamy tabulky bez logování, rychlejší
- COMMENT – umožňuje komentovat prvky databáze, není moc běžné
- RENAME – přejmenování tabulky [27]

3.3.2 Jazyk pro manipulaci s daty - Data manipulation language (DML)

Jazyk pro manipulaci s daty je nejběžněji používaným mezi koncovými uživateli a programátory databázových aplikací, kde vytváří prostředí pro vkládání, výběr, úpravu a mazání dat v databázi. Typické jsou následující příkazy:

- SELECT – vybírá záznamy z tabulek, nemodifikuje záznamy
- INSERT – vkládá záznamy do tabulky
- UPDATE – upravuje již vytvořené záznamy v tabulce
- DELETE – maže záznamy z tabulek [27]

3.3.3 Jazyk pro řízení dat - Data control language (DCL)

Tato část jazyka nejčastěji využívaná administrátory a sloužící pro správu oprávnění přístupu, přiděluje nebo naopak odebírá práva uživatelům. Následující příkazy:

- GRANT – přiděluje uživateli práva k databázi
- REVOKE – odvolává přístupová práva udělená příkazem GRANT [27]

3.4 Příkaz SELECT

3.4.1 Úvod do příkazu SELECT

Příkaz SELECT patří mezi DML (Data manipulation language) a jak již název vypovídá, provádí se pomocí něho selekce, neboli výběr, prvků z tabulky/tabulek databáze, na základě kritérií uvedených v těle příkazu. Příkaz SELECT (též označován jako dotaz) nijak nemění obsah databáze, pouze vrací data získaná z databáze na základě zadaných kritérií.

3.4.2 Syntaxe příkazu SELECT

Kompletní syntaxe příkazu SELECT je uvedena na Obrázku 8. Význam jednotlivých symbolů je následující. Pokud není část konstrukce v závorce, je povinná. Pokud se nachází v hranatých závorkách, je možno ji použít volitelně pouze odstraněním závorek. Pokud je mezi hodnotami svislá čára, vybíráme jednu z hodnot. Části textu velkými písmeny používáme přesně v uvedeném tvaru, části malými písmeny nahrazujeme aktuálními hodnotami [30].

```
SELECT [ DISTINCT | ALL ] sloupec1 [AS alias1], sloupec2 [AS
alias2], ....
FROM specifikace_tabulek
[ WHERE podminkova_konstrukce [ AND podminkova_konstrukce1 | OR
podminkova_konstrukce2 ] ]
[ GROUP BY nazev_sloupce ]
[ HAVING podminkovy_vyraz ]
[ ORDER BY nazev_sloupce1, nazev_sloupce2,...  [ASC | DESC] ]
[ LIMIT pocet_zaznamu [ OFFSET offset ] ]
```

Obrázek 8: Syntaxe příkazu SELECT

V syntaxi příkazu jsou povinné dvě části konstrukce. První částí je SELECT, za kterou následují sloupce (atributy) tabulek, které chceme vybrat. Jednotlivé názvy sloupců oddělujeme čárkami a je zde možnost dočasného přejmenování sloupců pomocí aliasů za použití klauzule AS (viz Obrázek 8). Přímo za výraz SELECT je možno umístit klíčové slovo DISTINCT, nebo ALL. DISTINCT způsobí, že záznamy budou vybrány bez duplicit, ALL vybere všechny záznamy. Implicitně je nastaveno ALL.

Druhou povinnou částí konstrukce je FROM, který specifikuje, z jakých tabulek budou záznamy extrahovány. Tabulky je možné dočasně přejmenovat pomocí zkratky, která

musí bezpodmínečně následovat za názvem tabulky. Názvy tabulek s případnými zkratkami se oddělují čárkami.

Další části příkazu SELECT jsou volitelné. Umožňují blíže specifikovat, které záznamy budou zahrnuty do výsledku. Klauzule WHERE umožňuje zadat jednu či více podmínek (kritérií). Pouze záznamy (řádky) splňující specifikovaná kritéria se mohou ve výsledku objevit. Jednotlivá kritéria se spojují použitím logických spojek AND (logický součin – „a zároveň“) a OR (logický součet – „nebo“).

Za klauzulí WHERE může následovat klauzule GROUP BY, která umožní seskupení záznamů podle vybraného sloupce/sloupců. Jednotlivé sloupce jsou opět odděleny čárkami. Za GROUP BY může následovat klauzule HAVING, která umožňuje zadat další podmínky (podobně jako klauzule WHERE). Tyto podmínky se však vztahují na již seskupené záznamy. V klauzuli HAVING je na rozdíl od WHERE možno využívat agregační funkce. Agregační funkce jsou v SQL matematické či statistické funkce nad seskupovanými daty, proto se používají výhradně s klauzulí GROUP BY [30].

Předposlední zmíněnou klauzulí je ORDER BY. Pomocí této klauzule můžeme výsledek seřadit podle uvedeného sloupce/sloupců (v případě většího počtu sloupců než 1 se sloupce oddělují čárkami) vzestupně ASC, nebo sestupně DESC. Výchozí řazení je ASC. Nakonec výraz LIMIT umožní specifikovat maximální počet vrácených záznamů a navíc pomocí OFFSET určit od jakého indexu začít vybírat záznamy [30].

Jak je patrné, variabilita tohoto příkazu je veliká v závislosti na tom, jaké konstrukce použijeme [30].

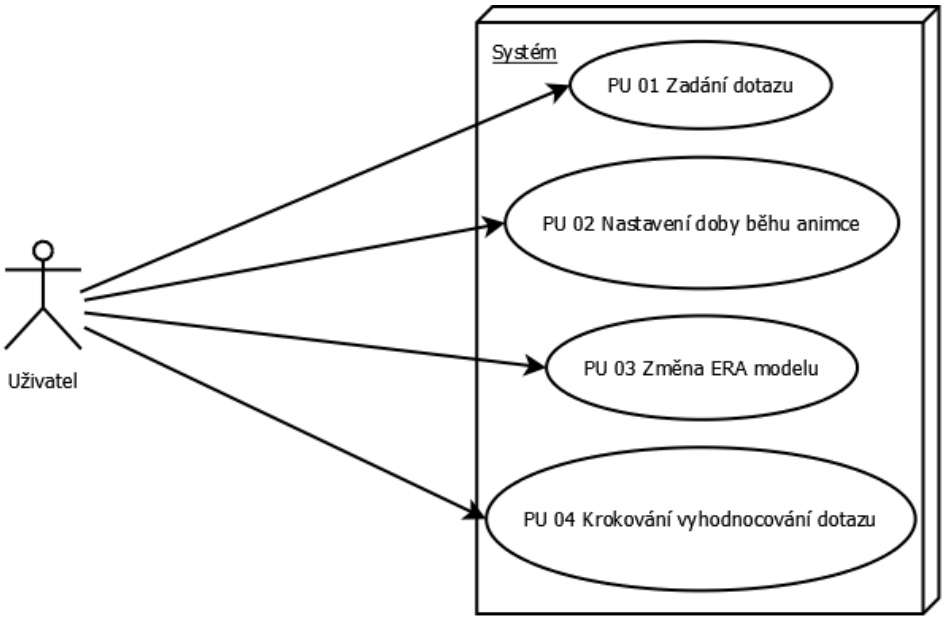
4 Analýza dané problematiky s návrhem řešení

4.1 Specifikace požadavků

Úkolem této práce bylo naprogramování on-line systému pro grafickou demonstraci postupného vyhodnocování SQL dotazů. Systém má běžet na webovém serveru, být přístupný z internetu a zobrazitelný ve webovém prohlížeči jako www stránka umožňující postupné vyhodnocení libovolného SQL dotazu SELECT se zaměřením na klauzule GROUP BY a HAVING. Systém by měl obsahovat více databázových modelů, mezi nimiž bude možné přepínat, aby bylo možné vyhodnocování dotazů demonstrovat v různých situacích a odlišném kontextu.

4.2 Případy užití

V následujících kapitolách budou stručně popsány případy užití systému a UML model těchto případů (viz Obrázek 9).



Obrázek 9: Diagram případů užití

4.2.1 PU 01

Uživatel zadává SQL dotaz typu SELECT. Systém odpovídá vrácením příslušného obsahu webové stránky, obsahující výsledek dotazu a všechna data potřebná k demonstraci jeho vyhodnocení.

4.2.2 PU 02

Uživatel nastaví prodlevu mezi jednotlivými kroky postupného vyhodnocování SQL dotazu. Má na výběr z několika možností.

4.2.3 PU 03

Uživatel vybere datový model (ve formě zobrazeného ERA modelu), pro který bude později zadávat dotaz.

4.2.4 PU 04

Uživatel bude pomocí tlačítka postupně krokovat části vyhodnocování doplněné o adekvátní animace.

4.3 Použité technologie

Jelikož se má jednat o on-line systém neboli webovou aplikaci/systém, bude k jeho tvorbě, testování a samotnému spuštění nutný webový server. Systém by měl vyhodnocovat SQL dotazy, tedy se neobejde bez systému řízení báze dat (SŘBD). Další nutností bude přítomnost serverového skriptovacího jazyka, který dokáže pracovat s databázovými nástroji, konkrétně s SQL jazykem, a bude jej schopen interpretovat do podoby vhodné pro výstup do webové aplikace/stránky. Kandidátů je více a bude potřeba vybrat nejvhodnější variantu. V souvislosti s weby všeobecně vyplývá nutnost použití značkovacího jazyka webových stránek HTML případně XHTML. HTML/XHTML se dnes již neobejde bez stylování pomocí kaskádových stylů a proto bude tuto technologii zapotřebí použít.

Zároveň bude třeba webovou stránku pro účely grafické demonstrace animovat. Grafická úprava a práce s animacemi ve vlastní webové stránce je již záležitostí klientské strany (internetového prohlížeče) a k tomu je nutné použít klientský skriptovací jazyk.

Proto bude i systém nutno rozdělit na část serverovou a část klientskou. Obě dvě části spolu budou úzce souviset, respektive klientská část bude přímo navazovat na část serverovou, a bude nutno obě dvě části dobře sesynchronizovat a kompozici webové stránky na serveru připravit tak, aby na ni klientská část programu mohla plynule navázat a vznikla tak požadovaná souhra celého systému. Následující podkapitoly mají za úkol představit model kompletní architektury aplikace a hlavní myšlenky pro realizaci úkolu.

4.4 Serverová část systému

Primárním úkolem pro serverovou část bylo zvolit vhodnou konfiguraci a prostředky. Tím je myšleno zvolení platformy, softwarové výbavy pro server a jazyka pro interpretaci SQL a přípravu webové stránky.

4.4.1 Konfigurace serveru

Pro účely této práce byla jako serverová výbava, zvolen balík nazývaný LAMP. Jedná se o balík technologií používaný pro tvorbu webových stránek a aplikací. Jako platforma je použita distribuce Linuxu, v tomto případě Debian. Jako serverový software slouží osvědčený Apache. Třetí písmeno názvu balíku znamená MySQL. Jedná se o SŘBD užívající pro komunikaci standardní SQL jazyk. Poslední písmeno stojí za názvem skriptovacího serverového jazyka PHP.

Tato konfigurace byla zvolena jako osvědčená a na serverech v praxi běžně používaná, nicméně výsledný systém může běžet i na jiných konfiguracích serverů schopných interpretovat PHP a pracovat s MySQL.

4.4.2 Databáze

Nadstavba systému MySQL s názvem phpMyAdmin umožňuje ve webovém prohlížeči připojení k databázi a v grafickém prostředí navržení celé struktury databáze, tabulek, atributů a naplnění databáze potřebnými daty. Pro potřeby práce budou vytvořeny dva až tři databázové modely pro demonstraci různých spojení tabulek (relací). Na nich bude potom možné demonstrovat různé použití dotazů.

Pro zlepšení orientace v jednotlivých modelech budou zhotoveny ERA diagramy pro každý model. Tyto diagramy budou dostupné ve webovém rozhraní, kde budou zároveň zadávány dotazy. Na základě zobrazených diagramů bude uživatel moci vytvářet dotazy, protože bude mít přehled o tabulkách a jejich atributech a vazbách.

4.4.3 Vymezení pojmů

V následujících kapitolách budou používány dva výrazy znějící podobně ovšem odlišných významů. Jedná se o výrazy *poddotaz* a *parciální dotaz*. Poddotazy jsou míněny části vzniklé rozdělením původního dotazu pomocí dělících výrazů (zmíněny později v kapitole 4.4.4), které ovšem samy o sobě nejsou SQL dotazy vhodné pro databázi. Naproti tomu parciální dotazy jsou již plnohodnotné SQL dotazy postupně vznikající zpětnou kompozicí právě z poddotazů a příslušných dělících výrazů.

4.4.4 Postupné vyhodnocování dotazů

Základ celého systému bude postupné vyhodnocování dotazů, které je potřebné pro demonstraci vyhodnocování dotazu. Samotná myšlenka postupného vyhodnocování dotazů se opírá o rozdělení zadaného dotazu pomocí určitých dělících výrazů (např. WHERE, AND/OR, GROUP BY) na poddotazy se zachováním pořadí jednotlivých poddotazů tak, jak byly nalezeny v původním dotazu. Následně by měla proběhnout postupná kompozice parciálních dotazů právě ze zmíněných poddotazů. Každý parciální dotaz bude vracet určitá data.

V podstatě by se mělo jednat o postupné napojování částí původního dotazu a tím ke zpřísňování kritérií pro návratová data. Jinými slovy by se měla postupně selektovat data od nejobecnějšího dotazu (výtah všech záznamů ze zadaných tabulek dotazu) po úplný, původně zadaný, dotaz. Postup je nastíněn ukázkou (viz Obrázek 10).

Původní dotaz: `SELECT jmeno,prijmeni,id FROM zamestnanec WHERE jmeno=“Karel“ OR id<10`

Rozdělení na poddotazy: `SELECT jmeno,prijmeni,id FROM zamestnanec; jmeno=“Karel“; id<10`

První parciální dotaz: `SELECT jmeno,prijmeni,id FROM zamestnanec`

Druhý parciální dotaz: `SELECT jmeno,prijmeni,id FROM zamestnanec WHERE jmeno=“Karel“`

Třetí parciální dotaz: `SELECT jmeno,prijmeni,id FROM zamestnanec WHERE jmeno=“Karel“ OR id<10`

Obrázek 10: Nástin techniky dělení dotazů a zpětné kompozice

Z ukázky by měla být patrná myšlenka, kam bude práce směřovat. Z takto postupně získaných dat bude nutné jednotlivé záznamy jednoznačně identifikovat, a tím umožnit další práci s nimi. Vzhledem k uvedenému postupu dělení dotazů bude nutné nalézt konečnou množinu dělících výrazů (dále je budu nazývat jako *spojky*), dle kterých bude dělení probíhat. Zároveň vyvstává nutnost uchování poddotazů a použitých dělících spojek ve správném pořadí tak, jak se v dotazu vyskytly.

Po dokončení dělení dotazu se bude provádět zpětná postupná kompozice parciálních dotazů. Každý parciální dotaz bude postoupen databázi a vrácená data budou zachycena

do připraveného HTML výstupního objektu (nebráno objektu z hlediska objektového programování, nýbrž objektu jako elementu struktury HTML objektového modelu). Všechny záznamy, jak již bylo řečeno, budou muset být jednoznačně identifikovatelné. Po dokončení celé kompozice, čili po dosažení poslední části původního dotazu a jejím vyhodnocení databází bude takto vzniklá struktura obalená potřebnými HTML tagy odeslána jako výstup klientovi.

4.5 Klientská část systému

Pro grafické zpracování a ovládání webových prvků a grafické zobrazení daných výstupů se neobejdeme bez použití dalšího skriptovacího jazyka webových aplikací, tentokrát na straně klienta. Jako nevhodnější adept se jeví skriptovací jazyk JavaScript se svou podpůrnou knihovnou jQuery. Jako jazyk si prošel jistým vývojem a mezi programátory webů je celkem populární [24,25,26]. Má širokou škálu použitelnosti a funkcí, které se budou pro tento projekt hodit.

4.5.1 Demonstrace vyhodnocování dotazů

Nyní již plně na klientské straně s výstupem připraveným serverem bude začínat grafické názorné postupné vyhodnocování parciálních dotazů. Dotaz by měl být vidět na obrazovce se zvýrazněným textovým řetězcem právě vyhodnocované části dotazu. Toto zvýraznění se bude postupně přesouvat s přibývajícími částmi a zrovna tak se budou měnit data zobrazující momentální záznamy z databáze, platné pro současný parciální dotaz. Změna by měla být doprovázena adekvátní animací, aby bylo patrné, co se se záznamy právě děje.

4.5.2 Animace vyhodnocování

K animování bude sloužit skriptovací jazyk JavaScript. Animovat by se mělo skrývání nepotřebných prvků a přepozicování na správné pozice. Další animace by měla sloužit k seskupování podle GROUP BY klauzule (jeden z hlavních bodů práce). Animace seskupení by měla probíhat tím způsobem, že se nejprve prvky patřící do téže skupiny podbarví stejnou barvou, poté přepozicují (přemístí) ke kořenovému prvku a následně skryjí, neboť do výstupu již nadále nepatří. S animacemi bude také souviset funkce nastavení časování, kde si uživatel bude moci nastavit dobu běhu každé animace v milisekundách. Na výběr bude cca z pěti hodnot.

4.5.3 Ostatní funkce klientské části

Další funkcí klientské části systému by měla být možnost volby jednoho z ERA diagramů popsaných v kapitole 4.1.2 a sloužících k lepší orientaci v navržených modelech databáze.

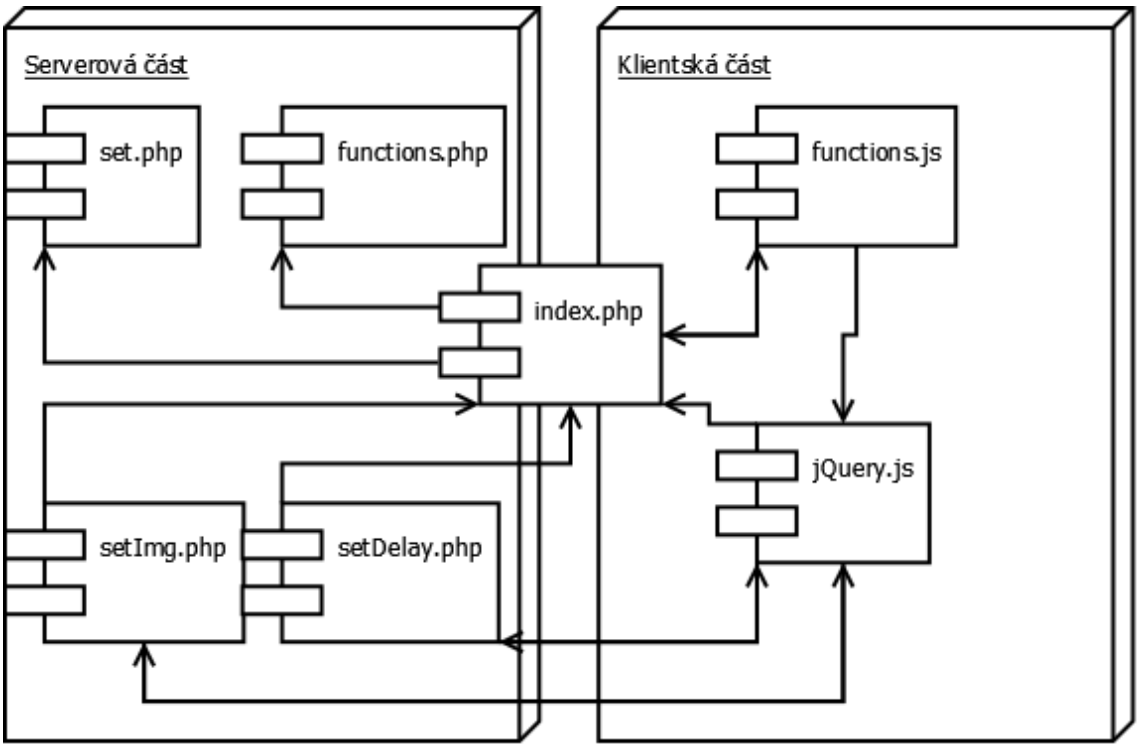
5 Implementace zadaného systému

5.1 Serverová část

Serverová část je programována v jazyce PHP. Její hlavní funkcí je příprava a rozdělení zadaného dotazu na poddotazy a následná kompozice parciálních dotazů a jejich odesílání na databázi se zachycením vrácených dat do struktury HTML stránky.

5.1.1 Struktura serverové části aplikace

Serverová část je komponována z hlavního souboru `index.php` umístěného v kořenovém adresáři webové aplikace a čtyř pomocných skriptových souborů umístěných ve složce `php_scripts`. Jedná se o soubory `functions.php`, `set.php`, `setDelay.php` a `setImg.php`. Struktura celé aplikace je vidět na Obrázku 11.



Obrázek 11: Struktura aplikace

5.1.2 Struktura webové stránky

Soubor `index.php` obsahuje standardní HTML hlavičku, kde jsou nalinkovány (připojeny) JavaScriptové soubory a soubor kaskádových stylů. Následuje tělo HTML dokumentu, ve kterém se jako první vyskytuje kontejner na obrázky ERA diagramů. Tyto diagramy je možné přepínat v klientském okně prohlížeče pomocí rozbalovacího menu. Pokud skript zjistí, že má v session proměnné uloženou hodnotu označující

poslední použitý diagram, tzn., že byl použit skript `setImg.php`, použije ji jako atribut `src` u obrázku a nastaví atribut `selected` u rozbalovacího menu, pokud nebude uložena, bude u obou použita výchozí hodnota. Na stejném principu pracuje i další rozbalovací menu ovládající dobu běhu animací s tím rozdílem, že tuto hodnotu nastavuje skript `setDelay.php`. Dalším prvkem stránky je formulářový vstup na zadávání dotazů a standartní odesílací tlačítko s hodnotou `Vyhodnotit`. Formulář je odesílán na server metodou POST. Posledním tlačítkem `Další krok` se ovládá již samotné vyhodnocování dotazu respektive jeho krokování vpřed.

5.1.3 Příprava datových struktur

O práci s daty se stará hlavní skriptový soubor `index.php`, kde jsou připojeny pomocné soubory `set.php` a `functions.php`. Soubor `functions.php` obsahuje externí funkce využívané systémem. Soubor `set.php` obsahuje nezbytná nastavení pro chod celého systému, jako nastavení interního kódování pro PHP, připojení na databázový server a při úspěchu nastavení kódování pro databázi. V opačném případě vypsání hlášky o neúspěchu navázání spojení.

Jako hlavní datové struktury slouží pole se spojkami, pole pro zachycení poddotazů a pole použitých spojek.

5.1.4 Příprava dotazu

Přípravou dotazu je míněna úprava samotného textového řetězce dotazu o kterou se stará funkce `queryChecker()` obsažená v souboru `functions.php`. Přestože jazyk SQL v rámci syntaxe nerozlišuje velká a malá písmena (rozlišuje je pouze u názvů tabulek a atributů), je zvyklostí používat u SQL výrazů všechna velká písmena. Vzhledem k tomu, že ne každý tyto zvyklosti dodržuje, je v systému přítomna funkce, která je schopna tyto nedostatky napravit. Tato funkce projde řetězec zadaný v parametru a nahradí vybrané SQL výrazy jejich ekvivalenty s velkými písmeny. Je přitom využito PHP funkce `str_ireplace()`, která dokáže najít a nahradit v řetězci hledaný podřetězec jiným bez závislosti na velikosti písmen. Opět je možné tuto funkci rozšířit o další výrazy přidáním řádků do těla funkce. Zároveň tím připravíme dotaz k následujícímu dělení, kde již u výrazů na velikosti písmen záleží.

5.1.5 Původní myšlenka dělení dotazu

První verze funkce pro dělení pracovala následovně. V cyklu se procházely všechny spojky v poli. V každém kroku byla snaha rozdělit dotaz podle aktuální spojky. Byla k tomu použita PHP funkce `explode()`, která umožňuje rozdělení jednoho řetězce podle druhého. Výstupem je pole, které má tolik záznamů, kolik řetězců vzniklo dělením.

Myšlenka byla tedy taková, že pokud bude vzniklé pole mít velikost 1 (zjištěno funkcí `count()`), znamená to, že se daná spojka v dotazu nevyskytuje, neboť se podle ní nic nerozdělilo. Pokud by tedy výsledek dělení byl pole o velikosti 1, pokračovalo by se s další spojkou na původním výrazu. Ovšem byla-li by velikost pole rovna 2, daná spojka by se v dotazu nacházela, do pole s poddotazy by se uložila první část dotazu před spojkou, do pole spojek by se uložila použitá spojka a dělení by pokračovalo už jen se zbývající částí dotazu.

5.1.6 Zjištěné problémy dělení dotazu

Tento postup se zpočátku zdál jako vyhovující, nicméně po několika testováních byl odhalen jeho hrubý nedostatek. Nepočítal totiž s tím, že první část dotazu po rozdělení může rovněž obsahovat spojku, ke které se zatím cyklus nedostal, neboť spojky v dotazu nemusí nutně být v pořadí, v jakém jsou uloženy v poli spojek. Tím mohlo dojít k přeskočení jednoho, či více poddotazů, což bylo uznáno nevyhovujícím.

5.1.7 Výsledné řešení dělení dotazu

Předchozí stav věci dal podnět ke vzniku nové myšlenky na úpravu postupu dělení. Pokud totiž může nastat situace popsaná výše, je nutné postoupit obě rozdělené části dotazu dalšímu dělení. Předchozí postup byl přesunut do těla externí funkce `querySplit()` do souboru `functions.php`, aby bylo možné celý postup volat rekurzivně. Původní konstrukce na porovnání hodnot byla přepracována na mnohem flexibilnější. Pro velikost pole po dělení rovnu 1 je zbytek přeskočen a pokračuje se další obrátkou cyklu. Pro ostatní hodnoty velikostí pole je spuštěn nový cyklus (důvodem je fakt, že původní dotaz může obsahovat více než jeden výskyt téže spojky a tudíž test na jeden výskyt je nedostačující), který projde všechny, kromě poslední, části rozděleného dotazu a na tyto části je znovu aplikována funkce dělení rekurzivně. Pokud k rozdělení nedojde, jsou spojky přeskakovány, pokud dojde je spuštěna další rekurze. Takto se pokračuje, dokud není docíleno jednotlivých atomických a dále nedělitelných

částí dotazu, přičemž vždy při návratu z rekurze je zapsán příslušný dotaz do pole pro dotazy a příslušná spojka do pole pro spojky. Poslední část dotazu pokračuje v původním cyklu, poté se uloží v poli.

Jelikož tento postup již plně vyhovuje funkčním požadavkům, byl uznán za ucházející a později byl ještě rozšířen. Byla totiž vznesena podmínka pro nedělení částí dotazu ve chvíli, kdy je používáno spojování tabulek pomocí klauzule WHERE/AND. V takovém případě totiž rozdělení není žádoucí, nýbrž se nejedná o selekci záznamů jako takovou, ale o spojení dvou či více tabulek. V každé obrátce cyklu je tedy sledováno, jestli rozdělený výraz neobsahuje následující vzorec spojení:

```
nazev_tabulky1.atribut1 = nazev_tabulky2.atribut2
```

Takový vzorec totiž odpovídá přidružení tabulek a v tu chvíli nebude dělení provedeno, respektive současný výraz bude připojen k předchozímu s použitím předchozí spojky. Stejného postupu je použito také pro situaci, v níž se vyskytuje klauzule BETWEEN/AND, pro niž je rovněž nevyhovující rozdělení, neboť použití klauzule BETWEEN bez AND vede k chybě. Princip celého algoritmu dělení dotazu je nastíněn v pseudokódu v Obrázku 12.

```
funkce dělení dotazu (řetězec)
  pro všechny záznamy v poli spojek;
    rozděl dotaz podle současné spojky do pole;
    pokud velikost pole = 1
      pokračuj další spojkou;
    jinak
      pro všechny záznamy v poli
        rekurzivní dělení dotazu (souč. výraz);
      po návratu z rekurze zapiš spojku do pole;
      zvyš pointer o 1;
    pokud výraz obsahuje WHERE/AND jako spoj tabulek, nebo
BETWEEN/AND klauzuli
      spoj poslední dva poddotazy pomocí spojky;
      sniž pointer o 1;
    jinak
      zapiš výraz do pole poddotazů;
```

Obrázek 12: Algoritmus dělení dotazu

Ve chvíli kdy jsme prošli všechny spojky, je dělení ukončeno a výsledná pole jsou naplněna potřebnými daty pro následnou kompozici.

5.1.8 Zpětná kompozice dotazu a strukturování obsahu webové stránky

Kompozice dotazu probíhá v těle dokumentu `index.php` po rozdělení dotazu. Samotná kompozice probíhá pomocí smyčky přes všechny prvky v poli poddotazů. Dotaz je postupně skládán z prvků v poli poddotazů a v poli spojek. Dochází k postupné kumulaci částí dotazu a dotazování na databázi. Po každém dotazu je vytvořena struktura HTML odstavce s párovým tagem `<p></p>` a identifikátorem s pořadím momentálního kroku cyklu. Do tohoto odstavce je umístěn ještě formátovací párový tag `<span></span>`, kterému je v prvním kroku cyklu zároveň přiřazena třída `class="active"`. Tato třída slouží později k vizuálnímu oddělování právě vyhodnocované části dotazu. Uvnitř tohoto tagu je již vypsána příslušná spojka a aktuální poddotaz.

Poté je třeba otestovat momentální parciálni dotaz na přítomnost agregačních funkcí a přítomnost klauzule GROUP BY. Pokud by totiž byla použita agregační funkce bez použití klauzule GROUP BY (klauzule může být použita později v dotazu), vedla by tato skutečnost k chybě. Aby systém této chybě předcházel, jsou zde opatření, která dovolí agregační funkci zůstat, pouze pokud již v parciálním dotazu je použita zároveň i klauzule GROUP BY. Pokud použita není, systém zjistí počáteční a koncový index agregační funkce v řetězci dotazu a tento řetězec ořízne o danou část podřetězce. Bere přitom v úvahu skutečnost možného výskytu mezery za čárkou oddělující atributy v řetězci a vhodně se adaptuje, aby výsledek byl korektní.

Následně je potřeba připravit pro jednotlivé budoucí záznamy z databáze jejich identifikátory pro použití v HTML/JavaScriptu. Jelikož každý záznam v databázi musí být jednoznačně identifikován primárním klíčem, avšak systém nemůže vědět který atribut klíčem je, je nutné použít všechny atributy pro každý záznam, mezi nimiž se primární klíč nevyhnutelně nachází. Proto je původně zadaný dotaz pro tyto účely přeformulován tak, aby místo výpisu požadovaných atributů měl znak `*`, a tím byly vybrány všechny atributy. Zbytek dotazu od klauzule FROM je zachován a tudíž bude mít původní i přeformulovaný dotaz stejný počet záznamů. Upravený dotaz je vyhodnocen databází a všechny atributy jednotlivých záznamů zřetězeny, hashovány a

poté doplněny o číselné označení momentálního kroku smyčky (resp. čísla parciálního dotazu), čímž dojde ke vzniku unikátních identifikátorů pro jednotlivé záznamy.

Následuje připojení k databázi a odeslání parciálního dotazu k vyhodnocení. Dalším krokem je vytvoření struktury tabulky s párovými tagy `<table></table>` s patřičným identifikátorem. Tento identifikátor umožňuje oddělit tabulky jednotlivých parciálních dotazů. Tato tabulka slouží k zachycení dat vrácených databází na momentální parciální dotaz. Zde může nastat několik variant.

Jelikož dotaz může být zadaný špatně, nebo mu nemusí odpovídat žádný záznam v tabulce, je nutno tyto případy ošetřit. Pokud bude dotaz zadán chybně, bude návratová hodnota dotazu `false`. Zároveň zjištěný počet sloupců vrácených databází, bude obsahovat nulovou hodnotu. V tomto případě se vygeneruje tabulka, která nebude obsahovat žádné hlavičkové buňky, ani řádky v těle tabulky a tím bude pro systém patrné, že došlo při vyhodnocování dotazu k chybě. V případě, že na dotaz nebudou vráceny žádné záznamy databáze, bude návratová hodnota dotazování rovna `0`, což není `false`, a tedy k chybě nedošlo a můžeme určit a vypsat požadované názvy sloupců do hlavičky tabulky mezi tagy `<thead></thead>`. Použitím funkce `mysqli_num_fields()` zjistíme počet sloupců a cyklem všechny projdeme a vypíšeme pomocí funkcí `mysqli_field_seek()` (nastavíme pointer na požadovanou pozici pole názvů) a `mysqli_fetch_field()` (odchytíme název pole). V případě nulového počtu vrácených záznamů je do těla tabulky vložena řádka s buňkou obsahující informaci o tom, že dotazu neodpovídá žádný řádek v databázi.

Pokud je počet vrácených záznamů větší než nula, je spuštěn cyklus přes všechny záznamy. Každý záznam tvoří jednu řádku tabulky. Řádky se zapisují mezi tagy `<tr></tr>`. Každý řádek je pro potřeby systému nutné opatřit identifikátorem řádku, který již byl zmíněn výše v této kapitole. Po zapsání identifikátoru řádku je spuštěn cyklus přes všechny atributy daného záznamu, který zapíše hodnoty atributů do buněk tabulky mezi tagy `<td><div>atribut</div></td>`. Atributy jsou ještě obaleny tagy `<div>` z důvodu jejich pozicování, jelikož některé prohlížeče jsou schopné pozicovat buňky tabulek a jiné ne. Tímto obalením se vyhneme vzájemné nekompatibilitě různých prohlížečů, protože pozicování blokových elementů je prováděno všemi prohlížeči stejně. Po doběhnutí cyklu je celá kompozice hotova a webová stránka je připravena k použití.

5.2 Klientská část

V této části aplikace se již všechny děje odehrávají v okně webového prohlížeče za použití klientského skriptovacího jazyka JavaScript s nadstavbovou knihovnou jQuery.

5.2.1 Nastylování dokumentu

Dokument je stylován pomocí stylopisu v externím souboru s názvem `stylesheet.css`. Ten se nachází v kořenové složce webu a je připojen k dokumentu pomocí tagu `<link>`. V zásadě se jedná o drobné úpravy jednotlivých HTML tagů jako nastavení vnitřního a vnějšího okraje, vykreslení rámečku kolem hlavičkových buněk tabulky, centrování textu, různé varianty podbarvení, nastavení zaoblení rámečku nebo barvy písma.

Zmíněné atributy nejsou nikterak zásadně důležité. Jsou zde však atributy tagů, bez kterých by systém nevypadal tak, jak má. Jedná se o skrytí (nastavením atributu `display` na hodnotu `none`) všech tabulek vyjma první s identifikátorem `t0` a stejně tak o skrytí všech odstavců kromě `q0` (u těchto dvou je nastaven atribut `display` na hodnotu `block`). Princip je v tom, že výše zmíněná tabulka a odstavec slouží jako výstupní. V nich se odehrává veškeré dění, které má být vidět a ostatní tabulky a odstavce (`t1`…`tn` a `q1`…`qn`) jsou pouze datovými zdroji, a tudíž by byla jejich viditelnost nežádoucí.

5.2.2 Projekce parciálních dotazů

Samotné jádro klientské části aplikace je psáno v jazyce JavaScript. Obdobně jako stylopis i JavaScriptové funkce jsou k dokumentu připojeny v hlavičce jako externí soubor s názvem `functions.js`. V hlavičce je umístěn ještě jeden tag `<script>`, který připojuje knihovnu jQuery také jako externí soubor s názvem `jquery-1.8.3.js`. Níže popsané konstrukce se vztahují pouze k souboru `functions.js`.

Jelikož jazyk JavaScript pracuje asynchronně (to znamená, že kód není zpracováván přesně v pořadí, jak je zapsán ve zdrojovém souboru) nebylo možné systém doladit do té úrovně, že by jednotlivé animace prvků byly odděleny a prováděny postupně, a tak jsou animace v každém kroku provedeny současně (míněno pro pozorovatele, poněvadž nějaké zpoždění v rámci milisekund přítomno je, nicméně je okem nepostřehnutelné a nerozeznatelné). Byl učiněn pokus, jak toto chování obejít pomocí smyčky aktivního čekání nad proměnnou, která by byla přenastavena po dokončení animace a tím

uvolněna smyčka k dalšímu běhu programu. Bohužel pokus skončil nezdarem, jelikož program zůstal viset ve smyčce a k přenastavení proměnné nikdy nedošlo.

Na začátku dokumentu jsou deklarovány dvě funkce pro ovládání roletkových menu v těle HTML stránky. První menu ovládá obrázek zobrazený jako ERA model databáze a druhé ovládá změnu rychlosti prováděných animací. Princip použití obou funkcí spočívá v přečtení právě zvolené hodnoty v menu a její aplikace do webové stránky, bez nutnosti znovunačtení stránky. Obsah stránky je tak dynamicky měněn bez nutnosti zbytečného načítaní ostatních prvků. Obě funkce zároveň využívají možnosti komunikace se serverem na pozadí, a tak nově nastavené hodnoty uloží do serverových proměnných $SESSION skrze skripty `setImg.php` a `setDelay.php` (zmíněné v kapitole 5.2.3 Příprava datových struktur) volané funkcí `$.post()` s potřebnými argumenty. Mimo tyto dvě funkce je zde, v případě že neexistuje, ještě deklarována proměnná `delay` a je jí přiřazena příslušná hodnota z roletkového menu. Dále jsou zde deklarovány proměnné použité v následujících částech skriptu, včetně proměnné `krok`, která je obzvláště důležitá, poněvadž řídí jednotlivé kroky demonstrování vyhodnocování dotazu. Pokud tato proměnná ještě neexistuje, je jí přiřazena hodnota `0`, v opačném případě se s ní nic neděje.

Systém demonstrování je řízen tlačítkem `Další krok` v těle webové stránky. Po jeho stisknutí se spustí funkce demonstrace následujícího kroku vyhodnocování. Funkce pracuje tak, že porovnává identifikátory prvků v množině aktuálně zobrazovaných záznamů (pro aktuální parciální dotaz) s identifikátory záznamů dalšího parciálního dotazu.

Na začátku každého kroku jsou nastaveny potřebné proměnné na aktuální hodnoty. K aktuálnímu parciálnímu dotazu je připojen následující použitím funkce `append()`. Pokud tabulka s daty pro následující parciální dotaz neobsahuje žádné řádky, ale následující parciální dotaz existuje, je tento stav detekcí chyby v uvedeném dotazu a tato chyba je oznámena uživateli.

V ostatních případech, pokud tabulka s následujícími daty obsahuje záznamy, přechází systém k zobrazování dalšího kroku vyhodnocování. Ať v případě chyby nebo v případě pokračování je odebrána třída `active` původnímu tagu `<span>` v zobrazovacím

odstavci a je přidělena připojenému tagu `<span>` v témže odstavci, což simuluje posun v dotazu o krok dále. Zde se systém dále větví podle různých kritérií.

5.2.3 Selekce a řazení prvků

Tato část podmínkové konstrukce nastává, pokud není použita klauzule GROUP BY a je zde s prvky manipulováno ve smyslu jejich třídění a řazení k obrazu výstupu daného parciálního dotazu.

V této části je nejprve spuštěn vnější cyklus procházející prvky momentálně zobrazené v tabulce `t0`. Zároveň je v každé obrátce cyklu spuštěn vnitřní cyklus přes prvky v tabulce příslušící momentálnímu parciálnímu dotazu. Systém zkouší porovnávat identifikátor prvku vnějšího cyklu s identifikátory všech prvků vnitřního cyklu, pokud narazí na shodu, nastaví příznakovou proměnnou `node_found` (indikuje nalezení shodného prvku), původně nastavenou na hodnotu `false`, na hodnotu `true`.

Dále pokud by prvek byl momentálně skryt, je tento znovu zobrazen animací za použití funkce `fadeIn(delay)` (argument v závorce značí dobu animace v milisekundách, která je určena hodnotou proměnné `delay`), je mu nastavena příslušná barva pozadí a je nastavena hodnota proměnné `animate` na `false`, aby nedocházelo k animování přesunu prvku na pozici (souběh těchto dvou animací nevypadal hezky). Pokud prvek nebyl skryt je `animate` nastaveno na `true`.

Vzápětí dochází k přesunu prvku na správnou pozici, pokud na ní prvek již není. Je zde rozdíl, pokud je prvek přesouván směrem nahoru nebo dolů. Roli zde hrají proměnné symbolizující současnou a požadovanou pozici prvku. Před samotným přesunem prvku ve struktuře tabulky jsou ještě spočítány souřadnice, pomocí nichž bude prvek animován na svou pozici. Animace přesunu je zprostředkována skrze pozicování v CSS prostřednictvím jQuery. Rozdíl hodnot souřadnic udává počet pixelů, o něž se prvek posune ve svislé ose.

Po přemístění prvku je nutno aktualizovat proměnnou ukazující na pole s prvky a nastavit hodnotu iterátoru vnějšího cyklu na -1, aby s další obrátkou byla procházena celá struktura od začátku. Absence tohoto nastavení zpočátku způsobovala chyby v pořadí, nýbrž některé prvky byly přeskočeny, nebo přepozicovány jinými. Na konci obrátky vnějšího cyklu je zkontrolována proměnná `node_found` a pokud obsahuje

hodnotu `false`, tedy daný prvek nebyl mezi novými nalezen, je skryt použitím funkce `fadeOut(delay)`.

5.2.4 Seskupování podle GROUP BY

Je-li systémem detekována přítomnost klauzule GROUP BY v současně vyhodnocovaném dotazu, je spuštěna větev programu specializovaná na animaci seskupování podle vypsaných atributů.

Algoritmus musí nejprve rozpoznat, podle jakých atributů bude seskupovat. Toho je dosaženo oddělením části řetězce obsahující podřetězec „ GROUP BY „ a následným rozdělením zbylého řetězce pomocí znaku „`,`“ (čárka). Po rozdělení by měly zbýt názvy atributů, podle nichž se má seskupovat, které nám ovšem samy nestačí. Potřebujeme zjistit indexy pozic, které odpovídají příslušným atributům v hlavičkových buňkách tabulky a podle těchto pozic se budou hledat prvky pro seskupení. Do pole si tedy uložíme všechny hlavičkové buňky tabulky a cyklem je projdeme a zjišťujeme, jestli jejich obsah je roven obsahu pole s výrazy pro seskupení. Při rovnosti přepíšeme v poli název atributu jeho indexem, posuneme pointer na další prvek a nastavíme pointer cyklu na -1, abychom vyřešili situaci, kdy nebudou výrazy v poli ve stejném pořadí jako názvy atributů v tabulce, což by způsobilo fatální chybu skriptu. Po skončení cyklu máme připravené pole indexů pro seskupování.

Následně je spuštěn cyklus, který řídí seskupování. Tento cyklus projde postupně všechny prvky tabulky, která je výsledkem dotazu po seskupení, čili jsou v ní přítomny prvky, které nám nakonec musí zbýt v tabulce pro zobrazení. Pro každý prvek je zde náhodným generátorem barev specifikována barva pro případné seskupované prvky i pro sebe sama. Jedná se o tři náhodná čísla od 51 do 255 (čísla od 0 byla vynechána, protože obarvení bylo příliš tmavé a obsah prvků nečitelný) symbolizující složky RGB modelu barev.

Poté je spuštěn další cyklus procházející prvky v zobrazující tabulce `t0`. Cyklus zkouší pro každý prvek porovnat hodnoty na příslušných indexech (na indexech prvků pro seskupování) s hodnotami prvku vnějšího cyklu. Toho je dosaženo použitím třetího cyklu přes pole indexů pro seskupování. Před spuštěním tohoto cyklu je nastavena proměnná `node_found` na `true`, a pokud se všechny příslušné atributy budou sobě rovnat, zůstane na hodnotě `true`. V případě byť jen jediné nerovnosti bude nastavena

na `false`. Pokud hodnota zůstane `true`, je podle `id` nalezen ekvivalent prvku v tabulce `t0` (mezi zobrazenými) jako kořenový prvek skupiny, zjištěna jeho pozice a prvek nalezen funkcí pro seskupení je podbarven příslušnou barvou, přepozicován na pozici kořenového prvku v tabulce `t0` a následně skryt funkcí `fadeOut()`. Takto se projdou všechny prvky, nalezené jsou podbarveny, přepozicovány a skryty. Pokud prvek narazí sám na sebe, obarví se příslušnou barvou a je pokračováno další obrátkou cyklu.

Po ukončení vnějšího cyklu ovšem nemusí být zbylé prvky na svých správných pozicích. Poněvadž seskupování pomocí databázového stroje prvky nejen seskupí, ale i seřadí podle příslušných atributů, je i v této části nutno provést řazení. Toto řazení je spuštěno s prodlevou rovnou dvojnásobku aktuálně nastavené doby pro animace, neboť v předchozích cyklech byly provedeny vždy dvě animace po sobě (přepozicování a skrytí). Algoritmus řazení je převzat z větve programu popsané v předchozí kapitole.

5.2.5 Úprava pro agregační funkce

Jelikož byla souhra agregačních funkcí a výskytu klauzule GROUP BY ošetřena již na serveru, je zde jen pár úprav, aby vše vypadalo tak jak má. Protože při absenci klauzule byly agregační funkce v dotazu vynechány, avšak při jejím výskytu již vynechány nejsou, je patrné, že výstup s agregační funkcí bude mít více atributů, než kolik jich měl výstup předchozí. Proto je nutné po seskupení doplnit hlavičkovou řádku o vynechané atributy a tělo tabulky o hodnoty těchto atributů. Vše se děje v návaznosti na předchozí události a toto doplnění atributů a hodnot je včleněno před řazení prvků po uplynutí dané prodlevy animace (viz konec posledního odstavce předchozí kapitoly). Toto doplnění je realizováno přepsáním hodnot uvnitř buněk tabulky a pro hezčí efekt doplněno animací vplynutí prvků. Poté jsou prvky seřazeny, jak již bylo řečeno. Celý postup seskupování stručně popisuje Obrázek 13.

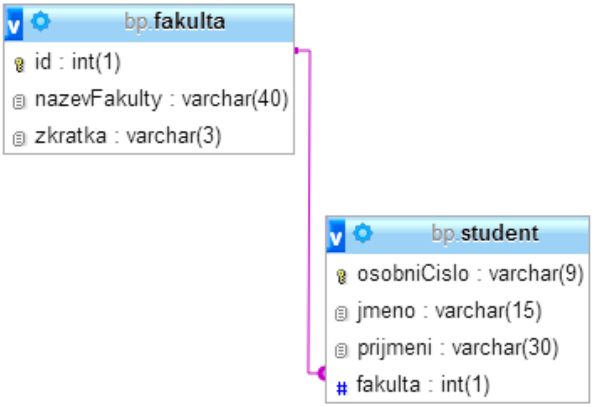
```
bez seskupování:
pro všechny prvky v současné množině výsledků
  hledej ekvivalent v koncové množině výsledků podle id;
  pokud prvek nalezen
    zkontroluj pozici, případně přepozicuj s animací;
  pokud prvek nalezen ale současný je skryt
    zobraz prvek na správnou pozici bez animace;
  jinak
    skryj prvek;
pro seskupování:
najdi všechny atributy, podle kterých se bude seskupovat, a
převeď je na jejich indexy v poli;
pro všechny prvky koncové množiny výsledků
  hledej prvky se shodnými atributy v současné množině;
  pokud shoda nalezena
    najdi kořenový prvek skupiny v současné množině použitím id
prvku koncové množiny;
    nastav barvu a přepozicuj prvek ke kořenovému s animací a
skryj ho;
  pokud shoda nenalezena pokračuj;
  nastav timeout pro následné řazení;
funkce po timeoutu:
  pokud je třeba doplň hlavičku a tělo tabulky o vynechané
atributy (agregační fce) a nech je vplynout animací;
  seřaď prvky po seskupení podle vzoru koncové množiny výsledků;
```

Obrázek 13: Algoritmus seskupování

5.3 Databázové modely pro systém

5.3.1 První model

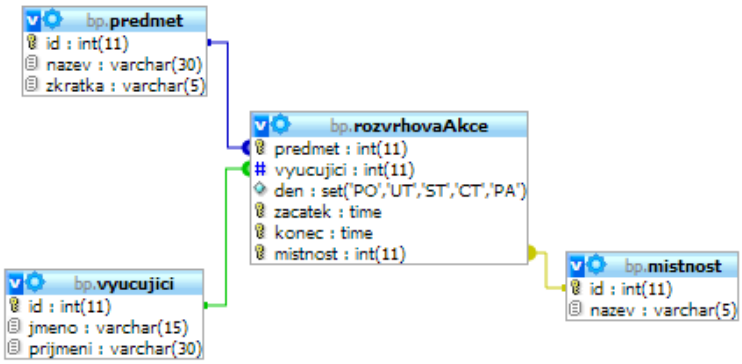
Prvním modelem v databázi je model se dvěma tabulkami, a sice `student` a `fakulta`. Mezi těmito tabulkami a jedna vazba 1:M (viz Obrázek 2: ERA diagram prvního modelu). Jedná se o velmi jednoduchý model na procvičení základních dotazů s podmínkami a eventuálně napojením dvou tabulek k sobě. Tabulka student obsahuje atributy `osobniCislo, jmeno, prijmeni, fakulta`. Identifikátorem/klíčem tabulky je atribut `osobniCislo` a `fakulta` je cizím klíčem. V tabulce fakulta jsou obsaženy atributy `id, nazevFakulty, zkratka`. Primárním klíčem je atribut `id`. ERA diagram modelu zachycen na Obrázku 14.



Obrázek 14 : ERA diagram prvního modelu

5.3.2 Druhý model

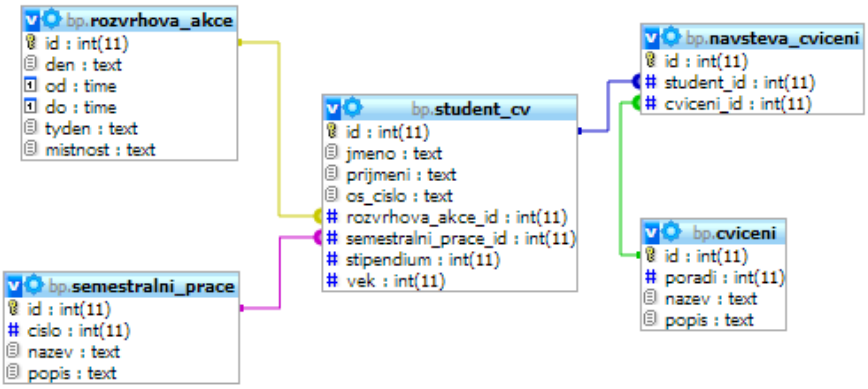
Dalším model je už složitější skladby, obsahuje čtyři tabulky pojmenované `predmet`, `vyucujici`, `rozvrhovaAkce`, `mistnost`. Tento model obsahuje vazby typu M:N. Propojovací tabulkou je `rozvrhovaAkce` obsahující atributy `predmet`, `vyucujici`, `den`, `zacatek`, `konec`, `mistnost`. Primární klíč propojovací tabulky tvoří všechny atributy vyjma atributu `vyucujici`. Tabulka `vyucujici` je připojena svým primárním klíčem `id` na cizí klíč tabulky `rozvrhovaAkce` `vyucujici`. Stejně tak jsou připojeny zbylé dvě tabulky `predmet` a `mistnost` svými primárními klíči `id` na cizí klíče `predmet` a `mistnost` tabulky `rozvrhovaAkce`. Era diagram modelu je na Obrázku 15.



Obrázek 15: ERA diagram druhého modelu

5.3.3 Třetí model

Třetí a poslední model má nejsložitější strukturu. Skládá se z 5 tabulek a reprezentuje vztahy mezi studenty na cvičeních, rozvrhovými akcemi a semestrálními pracemi studentů. Strukturu modelu zobrazuje Obrázek 16.



Obrázek 16: ERA diagram třetího modelu

6 Testování funkčnosti aplikace

Posledním bodem zadání bylo otestování funkčnosti vytvořené aplikace na několika vhodných příkladech. Pro testování budou použity nejrůznější dotazy obsahující různé typy vazeb, spojení tabulek, kritérií, seskupování a kritérii s agregačními funkcemi. Výsledky testování budou shrnuty a vhodně prezentovány v následujících kapitolách. Tabulky s výsledky budou obsahovat testovaný dotaz, počet kroků, na něž byl dotaz kompletně vyhodnocen a výsledek dotazu v porovnání s výsledkem standardního SŘBD (v tomto případě MySQL).

6.1 Testování použitím prvního modelu

V prvním modelu můžeme, pro jeho jednoduchou strukturu, otestovat základní funkce jako výběr záznamů z jedné tabulky s kritérii, spojení dvou tabulek použitím různých metod, nebo výběr ze dvou tabulek s kritérii.

6.1.1 Výsledek testování

Tabulka 1: Výsledky testování dotazů na prvním modelu

Dotaz	Počet kroků	Výsledek
SELECT * FROM student	1	OK
SELECT * FROM student WHERE fakulta = ‘3‘	2	OK
SELECT osobniCislo, jmeno, prijmeni, nazevFakulty FROM student,fakulta WHERE student.fakulta = fakulta.id	1	OK
SELECT osobniCislo, jmeno, prijmeni, nazevFakulty FROM student,fakulta WHERE student.fakulta = fakulta.id AND zkratka NOT LIKE ‘FEL‘	2	OK

6.2 Testování použitím druhého modelu

Druhý model je o něco složitější. Je zde výskyt vazby M:N a tudíž je možnost otestovat spojování více tabulek dohromady a skládání kritérií přes více tabulek. Zároveň zde budeme testovat seskupování a následnou filtraci použitím klauzule HAVING a agregačních funkcí.

6.2.1 Výsledek testování

Tabulka 2: Výsledky testování na druhém modelu

Dotaz	Počet kroků	Výsledek
SELECT p.nazev, v.jmeno, v.prijmeni, den, zacatek, konec, m.nazev FROM predmet p, rozvrhovaAkce r, vyucujici v, mistnost m WHERE r.predmet=p.id AND r.vyucujici=v.id AND r.mistnost=m.id	1	OK
SELECT p.nazev, v.jmeno, v.prijmeni, den, zacatek, konec, m.nazev FROM predmet p, rozvrhovaAkce r, vyucujici v, mistnost m WHERE r.predmet=p.id AND r.vyucujici=v.id AND r.mistnost=m.id AND p.zkratka='ZIS'	2	OK
SELECT nazev, zkratka, den, začátek, konec FROM rozvrhovaAkce r INNER JOIN předmět p ON r.predmet=p.id WHERE den='PO' OR nazev LIKE 'Základy%'	3	OK
SELECT jmeno, COUNT(jmeno) FROM vyucujici WHERE id<8 GROUP BY jmeno HAVING COUNT(jmeno)>1	4	OK

6.3 Testování použitím třetího modelu

Třetí a poslední model je ze všech nejsložitější. Jeho strukturu tvoří 5 tabulek. Dají se zde velmi dobře testovat agregační funkce, seskupování a řazení.

6.3.1 Výsledek testování

Tabulka 3: Výsledky testování na třetím modelu

Dotaz	Počet kroků	Výsledek
SELECT stipendium, COUNT(os_cislo) FROM student_cv WHERE stipendium > 0 GROUP BY stipendium	3	OK
SELECT stipendium, COUNT(os_cislo) FROM student_cv WHERE stipendium > 0 GROUP BY stipendium HAVING COUNT(os_cislo) >= 2	4	OK

SELECT jmeno, prijmeni, os_cislo, vek from student_cv ORDER BY vek	2	OK
SELECT jmeno, prijmeni, os_cislo, den, tyden, od, do FROM student_cv, rozvrhova_akce WHERE student_cv.rozvrhova_akce_id = rozvrhova_akce.id	1	OK

6.4 Shrnutí testování

Každý ze tří modelů byl otestován čtveřicí různorodých dotazů pro zjištění funkčnosti, kdy bylo zaznamenáno, na kolik kroků byl dotaz zcela vyhodnocen a jak dopadlo testování v porovnání s výstupem standardního systému řízení báze dat (zde MySQL). Všechny testované dotazy byly vyhodnoceny správně.

7 Závěr

Cílem této práce bylo vytvořit online systém, který bude schopen demonstrovat vyhodnocování SQL dotazů se zaměřením na dotaz SELECT, zvláště potom na klauzule GROUP BY a HAVING.

Po důkladné analýze daného problému byl vymyšlen hlavní algoritmus celého systému, kdy postupné vyhodnocování je zajištěno rozdělením dotazu na poddotazy, které jsou následně odeslány na databázový server a výsledky vráceny zpět. Následnou zpětnou kompozicí vznikne soubor datových struktur, které zachycují potřebné mezivýsledky. Na základě nich je poté možno sledovat změnu dat v závislosti na použitých konstrukcích dotazu. Vše je po vyhodnocení zobrazeno ve webovém prohlížeči a dané mezivýsledky jsou graficky prezentovány.

Vytvořený systém dokáže reagovat na většinu běžně používaných konstrukcí dotazu SELECT a použitím různých animací názorně demonstrovat postupné vyhodnocování dotazu. Všechny body zadání tak byly splněny.

Zdroje

[1] *Apache Server Definition*, http://www.modulehosting.com/apache.html (27. 4. 2014)

[2] *Information about the Apache Web Server*, http://www.ntchosting.com/apache-web-server.html (27. 4. 2014)

[3] *About the Apache HTTP Server Project - The Apache HTTP Server Project*, http://httpd.apache.org/ABOUT_APACHE.html (27. 4. 2014)

[4] *Web Server technologies Web Usage Statistics*, http://trends.builtwith.com/web-server# (27. 4. 2014)

[5] *What is web server?*, http://www.webdevelopersnotes.com/basics/what_is_web_server.php (27. 4. 2014)

[6] *RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1*, http://tools.ietf.org/html/rfc2616 (27. 4. 2014)

[7] *Web server - Wikipedia, the free encyclopedia*, http://en.wikipedia.org/wiki/Web_server (27. 4. 2014)

[8] *Home : The Official Microsoft IIS Site*, http://www.iis.net/ (22. 6. 2014)

[9] *nginx*, http://nginx.org/en/ (22. 6. 2014)

[10] *May 2014 Web Server Survey | Netcraft*, http://news.netcraft.com/archives/2014/05/07/may-2014-web-server-survey.html (22. 6. 2014)

[11] BRÁZA, Jiří *PHP 5: začínáme programovat*

Vydavatel: Grada Publishing a.s., 2005, ISBN: 978-80-247-1146-1

[12] *PHP: What is PHP? – Manual*, http://www.php.net/manual/en/intro-whatis.php (28. 4. 2014)

[13] *PHP 5 Introduction*, http://www.w3schools.com/php/php_intro.asp (28. 4. 2014)

[14] *PHP: History of PHP – Manual*, http://www.php.net/manual/en/history.php.php (28. 4. 2014)

[15] PROCHÁZKA, David *CSS a XHTML: tvorba dokonalých WWW stránek krok za krokem* - 2., aktualizované vydání
Vydavatel: Grada Publishing a.s., 2011, ISBN: 978-80-247-3897-0

[16] *Historie a vývoj HTML - - HTML5*, http://htmlguru.cz/uvod-historie.html (29. 4. 2014)

[17] *What Is HTML - History of Web Page Markup*, http://webdesign.about.com/od/htmlxhtmltutorials/a/what-is-html.htm (29. 4. 2014)

[18] *Introduction to HTML*, http://www.w3schools.com/html/html_intro.asp (28. 4. 2014)

[19] *Sémantika - Úvod do sémantiky*, http://www.semantika.name/uvod-semantika.html (29. 4. 2014)

[20] *Kaskádové styly dokumentů - - HTML5*, http://htmlguru.cz/css.html (29. 4. 2014)

[21] *CSS prakticky, použití CSS v html stránkách*, http://www.jakpsatweb.cz/css/css-prakticky.html (6. 5. 2014)

[22] *Javascript – úvod*, http://www.jakpsatweb.cz/javascript/javascript-uvod.html (6. 5. 2014)

[23] *JavaScript Introduction*, http://www.w3schools.com/js/js_intro.asp (6. 5. 2014)

[24] *jQuery návod - vše okolo jQuery*, http://jquery-navod.cz/kategorie-ostatni-clanky/1-uvodni-clanek (2. 5. 2014)

[25] *jQuery*, http://jquery.com/ (16. 6. 2014)

[26] *History | jQuery Foundation*, https://jquery.org/history/ (16. 6. 2014)

[27] *Databáze a jazyk SQL | Interval.cz*, http://interval.cz/clanky/databaze-a-jazyk-sql/ (1. 5. 2014)

[28] *SQL Introduction*, http://www.w3schools.com/sql/sql_intro.asp (1. 5. 2014)

[29] *Databáze nejsou jen MySQL | Interval.cz*, http://interval.cz/clanky/databaze-nejsou-jen-mysql/ (1. 5. 2014)

[30] *SQL Syntax*, http://www.mckoi.com/database/SQLSyntax.html#15 (2. 5. 2014)

Přílohy

Příloha 1: Uživatelská příručka

Příloha 2: Obsah CD

Příloha 1: Uživatelská příručka

Postup instalace a zprovoznění systému

Základním prvkem je funkční webový server, na kterém běží PHP a databázový systém (nejlépe MySQL). Pro naplnění databáze daty je nutné provést import do databáze ze souboru `bp.sql`, který se nachází v adresáři `SQL` přiloženého CD.

Po úspěšném importu je třeba zkopírovat celý obsah adresáře `SOURCE` přiloženého CD do kořenového adresáře webového serveru (případně do nějakého podadresáře v závislosti na struktuře serveru). Zároveň je potřeba po zkopírování obsahu upravit soubor `set.php` v adresáři `php_scripts`, aby se mohl systém připojit k databázi na serveru (nastavit funkce `mysqli_connect()` a `mysqli_select_db()`).

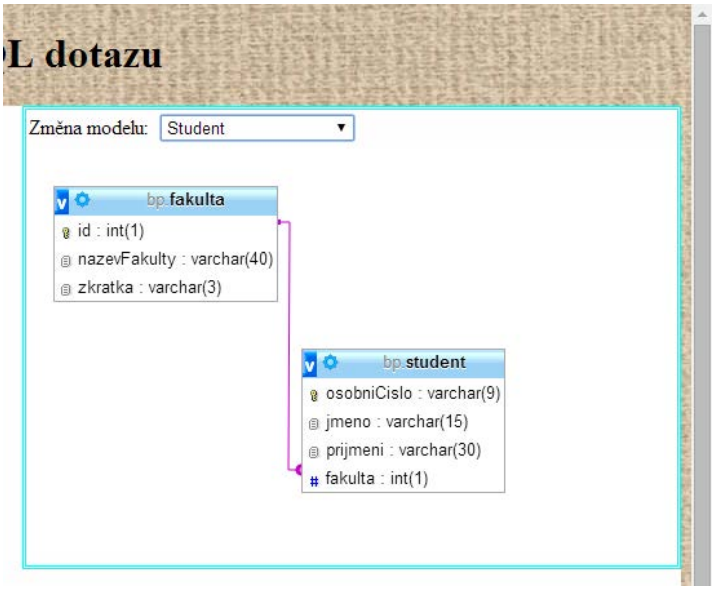
Tím je systém připraven k použití.

Práce se systémem

Do systému se dostaneme použitím internetového prohlížeče a zadáním adresy webového serveru, kde je systém umístěn. Po načtení se zobrazí hlavní stránka, ve které bude probíhat veškeré dění a práce se systémem.

V levé části obrazovky je situováno ovládání zadávání dotazů a nastavení rychlosti běhu animací. Do prázdné řádky je možné zadat SQL dotaz a kliknutím na tlačítko [Vyhodnotit] (nebo stisknutím klávesy <Enter>) nechat systém dotaz vyhodnotit. Pokud se SQL dotaz bude skládat z více částí (tzn. bude obsahovat spojku WHERE, AND, OR, GROUP BY, HAVING), je možné dotaz postupně krokovat klikáním na tlačítko [Dalsi krok] (zde je velmi důležité nechat vždy systém dokončit předchozí animace, v opačném případě se setkáme s chybou). V případě krokování je ještě možné zde nastavit rychlost resp. dobu běhu animací v milisekundách. K tomu slouží roletkové menu napravo od tlačítka [Vyhodnotit]. Na výběr je z pěti možností (200ms, 500ms, 1000ms, 2500ms, 5000ms).

Je možné zadávat standardní SQL dotazy. Při použití klauzule WHERE/AND pro spojení dvou a více tabulek je nutno uvádět vždy celý zápis (název_tabulky.atribut).



V pravé části obrazovky je možné přepínat pomocí roletkového menu ERA diagramy databázových modelů použitých v systému. Je zde na výběr ze tří modelů (Student, Rozvrhová akce, Studenti na cvičeních). Každý model má pevně danou strukturu, která je zachycená v použitých ERA diagsramech.

Příloha 2: Obsah CD

Kořenový adresář obsahuje soubor `readme.txt`, v němž je popsán obsah celého CD. Dále obsahuje pdf soubor s textem bakalářské práce a adresáře `SOURCE` a `SQL`.

Adresář `SOURCE` obsahuje kompletní zdrojové kódy celého systému.

Adresář `SQL` obsahuje exportovaný soubor databáze určený pro import do nové databáze.