

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2010

Josef Kaláb

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Kalendář akcí a rezervace techniky HZS ČR

Josef Kaláb

Bakalářská práce

2010

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Josef KALÁB**
Osobní číslo: **I070036**
Studijní program: **B2646 Informační technologie**
Studijní obor: **Informační technologie**
Název tématu: **Kalendář akcí a rezervace techniky HZS**
Zadávající katedra: **Katedra informačních technologií**

Z á s a d y p r o v y p r a c o v á n í :

Tato práce má za úkol vytvořit aplikaci pro HZS, která bude obsahovat kalendář sportovních a jiných akcí. Dále bude obsahovat modul pro rezervaci techniky a administraci uživatelů (přidávání, změnu a odstranění uživatel). Pro jednotky požární ochrany bude vytvořen modul na jejich evidenci. Aplikace bude vytvořena jako internetová napsaná v PHP a bude využívat databáze MySQL.

V teoretické části budou rozebrány základy databázového jazyka MySQL a programovacího jazyka PHP.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

GUTMANS, Andi; BAKKEN, Stig Sather; RETHANS, Derick. Mistrovství v PHP 5. 2. vydání. Praha : Computer Press, 2007. 656 s. ISBN 97880251115190.

Kolektiv autorů. PHP6, MySQL, Apache : Kolektiv autorů. Praha : Computer Press, 2006. 816 s. ISBN 978-80-251-2767-4.

WELLING, Luke; THOMSON, Laura. MySQL Průvodce základy databázového systému. Vyd. 1. Brno : CP Books, 2005. 255 s. ISBN 80-251-0671-3.

Vedoucí bakalářské práce:

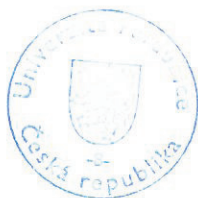
Ing. Pavel Škrabánek
Katedra řízení procesů

Datum zadání bakalářské práce: **15. ledna 2010**

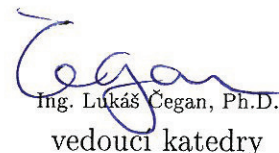
Termín odevzdání bakalářské práce: **14. května 2010**



prof. Ing. Simeon Karamazov, Dr.
děkan



L.S.



Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2010

Prohlašuji:

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

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Ústí nad Orlicí dne 12.5.2010

Josef Kaláb

Anotace

V teoretické části je popsána historie a vlastnosti databázového systému MySQL včetně vysvětlení základních příkazů. Dále je popsán vznik skriptovacího jazyka PHP, základní datové typy a základy programování v PHP. Výše uvedené technologie jsem využil při tvorbě aplikace, která si vzala za úkol vytvořit program kalendář akcí a rezervace techniky HZS ČR.

Klíčová slova

MySQL, PHP, CSS, Rezervace, HZS, Kalendář

Title

Event calendar and technology reservation HZS

Annotation

Theoretical part describes history and properties of database system MySQL including an explanation of basic orders. Further, the thesis focuses on formation of scripting language PHP, basic data types and elements of programming in PHP. I employed above mentioned technologies at creation of computer application, which task was to form a program of calendar of events and reservation of technics HSZ CR.

Keywords

MySQL, PHP, Reservation, HZS, Calendar

Obsah

1	Úvod.....	11
2	MySQL	12
2.1	Historie.....	12
2.2	Úvod do MySQL.....	13
2.2.1	Sloupec.....	13
2.2.2	Řádek	13
2.3	Výhody MySQL.....	14
2.4	Nevýhody MySQL	14
2.5	Typy tabulek.....	14
2.5.1	ISAM	14
2.5.2	MyISAM.....	15
2.5.3	InnoDB.....	15
2.5.4	HEAP	15
2.5.5	CVS.....	16
2.5.6	Dočasné tabulky.....	16
2.6	Normální formy.....	16
2.6.1	První normální forma.....	17
2.6.2	Druhá normální forma	18
2.6.3	Třetí normální forma.....	19
2.7	Relace.....	19
2.7.1	1:1	19
2.7.2	1:n	20
2.7.3	m:n	20
2.8	Klíče	20
2.8.1	Primární klíč (primary key)	20
2.8.2	Cizí klíč (foreign key).....	21

2.9	Základy jazyka SQL.....	21
2.9.1	Datové typy.....	21
2.9.2	Vytvoření databáze	23
2.9.3	Vytvoření Tabulek	24
2.9.4	Změna tabulky	24
2.9.5	Vložení záznamů do tabulek.....	25
2.9.6	Změna obsahu záznamů.....	25
2.9.7	Odstranění záznamů.....	26
2.9.8	Vyprázdnění tabulky.....	26
2.9.9	Odstranění databáze, tabulky.....	26
2.9.10	Jednoduché dotazy SELECT	26
2.9.11	Omezení výpisu na několik sloupců	27
2.9.12	Zjištění počtu řádků	27
2.9.13	Počet odlišných hodnot.....	28
2.9.14	Omezení počtu vypsaných záznamů.....	28
2.9.15	Třídění dotazů	29
2.9.16	Výběr záznamů na základě podmínky	29
2.9.17	Spojení dat z více tabulek	30
3	PHP	32
3.1	Co je PHP?.....	32
3.2	Historie PHP.....	32
3.3	Vkládání PHP do HTML	33
3.4	PHP značky	33
3.5	Vkládání HTML do PHP	34
3.6	Datové typy	34
3.7	Proměnné.....	34
3.7.1	Práce s proměnnými.....	34

3.7.2	Předem nadefinované proměnné.....	35
3.8	Konstanty	35
3.9	Operátory v PHP	36
3.10	Řídící struktury	36
3.10.1	Příkaz if.....	37
3.10.2	Konstrukce switch.....	37
3.11	PHP cykly.....	38
3.11.1	Cyklus typu while	38
3.11.2	Cyklus typu do	38
3.11.3	Cykly typu for.....	38
3.11.4	Cyklus foreach	39
3.11.5	Break a continue	40
4	Popis vytvořené aplikace	41
4.1	Návrh databáze.....	41
4.1.1	Aktualizace dat	43
4.1.2	Zajištění bezpečnosti.....	44
4.2	Aplikace z pohledu uživatelů.....	45
4.2.1	CSS	47
5	Závěr	49
5.1	Použité zdroje.....	50

Seznam obrázků

obr 1. Vytvoření databáze filmů	17
obr 2. Databáze filmů v 1NF	18
obr 3. Tabulka titulů a osob	18
obr 4. 2NF	18
obr 5. 3NF	19
obr 6 Výpis tabulky příslušník.....	27
obr 7 Omezený výpis tabulky	27
obr 8 Počet záznamů	27
obr 9 Omezení DISTINCT	28
obr 10 Výpis tabulky pomocí LIMIT	28
obr 11 Seřazený výpis.....	29
obr 12 Výpis pomocí LIKE	29
obr 13 Výpis dotazu s podmínkou	30
obr 14 Spojení dvou tabulek	31
obr 15 Tabulka matematických operátorů	36
obr 16 Tabulka logických operátorů.....	36
obr 17 E-R diagram vytvořené databáze.....	43
obr 18 Ošetření data.....	45
obr 19 Hlavní stránka aplikace	46
obr 20 Editace techniky	47
obr 21 Ukázka CSS.....	48

Seznam zkratek

CSS	Cascade Style Sheets
GPL	General Public License
HTML	Hypertext Markup Language
HZS ČR	Hasičský záchranný sbor České republiky
ID	Identification
ISAM	Indexed Sequential Access Method
JPO	Jednotka požární ochrany
OEČ	Osobní evidenční číslo
OOP	Object-oriented programming
PHP	Hypertext Preprocessor
SQL	Structured Query Language
TFA	Toughest Firefighter Alive
TSV	Tab-separated values
URL	Uniform Resource Locator

1 Úvod

První profesionální hasičský sbor byl založen už v roce 1853 v Praze. Od té doby prošel sbor mnoha změnami. Největší změny nastaly především v oblasti technického vybavení a připravenosti příslušníků. HZS ČR prezentuje svojí techniku a dovednosti na sportovních akcích jako je například v poslední době hodně rozšířená soutěž TFA nejtvrďší hasič přežívá, ale také předvádí ukázky zásahu pro města a obce. Dále provádí školení a výcviky pro JPO. Jelikož technika, různé akce a školení neustále přibývají je potřeba vytvořit program na rezervaci techniky a kalendář akcí pro HZS ČR. Toto byl impulz k zadání bakalářské práce.

V první teoretické kapitole této práce bude rozebrán systém pro správu relačních databází MySQL. Popsána bude historie, vlastnosti, výhody a nevýhody MySQL. Dále bude následovat popis příkazů pro správu databáze. Rozebrány budou také příkazy pro tvorbu a úpravu databázových tabulek, triggerů. Popsány budou SQL dotazy pro získání dat z tabulek. Také se zde zaměřím na typ útoku na tak zvaný SQL injection a obranu proti ně-mu.

Druhá kapitola bude věnovaná skriptovacímu jazyku PHP. Popsána bude historie, vlastnosti. Dále se budeme věnovat základním programovacím technikám, jako jsou základní datové typy, větvení programu a využívání cyklu.

V další kapitole bude popsána samotná aplikace, která tvoří praktickou část bakalářské práce. Jejím cílem bylo navrhnout aplikaci pro rezervaci techniky a evidenci jízd. Schvalování jízd a uzavírání příkazů. Dále byl vytvořen modul pro přehled a zadávání akcí a to at' už sportovních, nebo pro ukázky činnosti. Byla také přidána sekce pro spravování a editaci JPO a jejich členu.

V závěru se nachází celkové hodnocení a shrnutí práce. Popis nasazení aplikace do reálného provozu a připomínky k ní. Dále návrh jak by se mohla aplikace do budoucnosti vylepšit a rozšířit.

2 MySQL

2.1 Historie

MySQL je databázový systém, vytvořený švédskou firmou AB, nyní vlastněný společností Sun Microsystems, dceřinou společností Oracle Corporation. Jeho hlavními autory jsou Michael Widenius a David Axmark. MySQL je považován za úspěšného průkopníka dvojího licencování – je k dispozici jak pod bezplatnou licenci GPL, tak pod komerční placenou licenci.

MySQL je multiplatformní databáze. Komunikace s ní probíhá, jak už název napovídá, pomocí jazyka SQL. Podobně jako u ostatních SQL databází se jedná o dialekt tohoto jazyka s některými rozšířeními.

Pro svou snadnou implementovatelnost jej lze instalovat na Linux, MS Windows, ale i další operační systémy. Výkon a především díky tomu, že se jedná o volně šiřitelný software, má vysoký podíl na v současné době používaných databázích. Velmi oblíbená a často nasazovaná je kombinace MySQL, PHP a Apache jako základní software webového serveru.

MySQL bylo od počátku optimalizováno především na rychlost, a to i za cenu některých zjednodušení: má jen jednoduché způsoby zálohování, a až donedávna nepodporovalo pohledy, trigger, a uložené procedury. Tyto vlastnosti jsou doplňovány teprve v posledních letech.

2.2 Úvod do MySQL

Jako první bychom si měli říci, co jsou vlastně databáze a proč je využíváme. Databáze je velmi zjednodušeně určité úložiště dat. Proč vlastně používat databáze?

- Databáze poskytuje rychlejší přístup k datům než soubory.
- Databáze umožňuje přímý přístup k datům.
- Databáze má zabudovaný mechanismus pro paralelní přístup k datům.
- Databáze má zabudovaný systém uživatelských práv.
- Databáze umožňuje pomocí dotazů snadno extrahovat množiny dat, která vyhovují zadaným kritériím.

Databáze se dají rozdělit ještě do několika druhů. Dále se budeme věnovat pouze relační databázi MySQL. Základem každé relační databáze je tabulka, která obsahuje data. Některé databázové systémy mají pouze jednu tabulku, ale u relačních databází jich můžeme mít více. Tabulka se skládá ze sloupců a řádků.

2.2.1 Sloupec

Každý sloupec musí mít jedinečný název a určitý datový typ podle dat, která chceme ukládat. Datových typů je velmi mnoho a je dobré si řádně rozmyslet, jaký datový typ zvolíte. Určitě je zbytečné využívat datový typ TEXT, když chcete uložit jen jeden znak.

2.2.2 Řádek

Řádky nebo také záznamy. Oba pojmy jsou identické, jelikož jeden řádek reprezentuje jeden záznam. Každý řádek by měl mít určitý *jedinečný identifikátor*, který jednoznačně určí příslušný záznam.

2.3 Výhody MySQL

- Nejdůležitější vlastností databáze je její stabilita. MySQL je velmi stabilní, každá nová verze je vždy vývojáři důkladně otestována.
- Důležitá vlastnost je rychlost. MySQL je téměř ve všech kategoriích nejrychlejší.
- MySQL je možno nainstalovat na většinu operačních systémů.
- MySQL podporuje přístup z mnoha programovacích jazyků (C, C++, Eiffel, Java, Perl, PHP).
- Poměrně dynamický vývoj MySQL.
- Dostupnost a cena – pro nekomerční účely zdarma.
- MySQL je dnes velmi rozšířená. Obrovská výhoda vyplývající z rozšířenosti je uživatelská podpora.

2.4 Nevýhody MySQL

- Nepodporuje složitější programátorské konstrukce
- Nemá dostatečný výkon v opravdu náročných webových aplikacích.

2.5 Typy tabulek

Specialitou MySQL je to, že se při vytvoření nové tabulky musí zadat její typ. MySQL podporuje různé typy tabulek, které se navzájem liší řadou vlastností.

2.5.1 ISAM

Je metoda ukládání dat s možností rychlého vyhledávání pomocí speciálních pomocných struktur (indexů). V takzvaném primárním souboru jsou zapsány úplné datové záznamy pevné délky sekvenčním způsobem, to je v takovém pořadí, v jakém byly přidány, a nebývají nijak setříděny.

Dále je vytvořeno jeden nebo více indexů, což jsou struktury umožňující rychlé hledání záznamů podle některých atributů. Jejich úkolem je minimalizovat počet přístupů do primárního souboru, bývají často velmi malé (oproti primárnímu souboru), protože obsahují jen informace potřebné k hledání podle daného atributu, a pokud hledaný záznam (či více záznamů) existuje, vrátí index ukazatel (nebo více ukazatelů) do primárního souboru na hledaný záznam (záznamy).

2.5.2 MyISAM

Je následovníkem formátu ISAM. MyISAM představuje v MySQL standardní typ tabulek. Jedná se o stabilní, vyspělý a jednoduše upravovatelný typ tabulek. Pokud nemáte žádný zvláštní důvod, proč použít jiný typ tabulek, pak použijte právě tento. Každá tabulka MyISAM je uložena na disku ve třech souborech. MyISAM poskytuje největší podporu pro *AUTO_INCREMENT* na rozdíl od ostatních typů tabulek. Další velkou výhodou tohoto typu je poskytování *FULLTEXT* indexu, pomocí kterého lze provádět fulltextové vyhledávání.

2.5.3 InnoDB

Typ InnoDB je v porovnání s předchozím typem tabulky nový. Podporuje všechny vlastnosti MyISAM, navíc se pyšní dvěma odlišnostmi:

- Databázové operace v tabulkách InnoDB se dají spouštět jako transakce. Transakce jsou v mnoha případech bezpečnější a občas i rychlejší, pokud se můžete vyhnout tzv. operací Locking.
- Tabulky InnoDB podporují používání pravidel integrity (pravidla cizího a primárního klíče)

Bohužel existují i důvody, které hovoří proti používání tabulek typu InnoDB:

- InnoDB ještě nedosáhl vysoké stability tabulek jako je u typu MyISAM.
- U tabulek InnoDB nelze vytvořit fulltextový index.
- Správa tabulek je o něco složitější
- Komerční licence MySQL s podporou InnoDB je dvakrát tak dražší než bez InnoDB.

2.5.4 HEAP

Tabulky HEAP jsou zajímavé tím, že se vytváří pouze v operační paměti RAM (neukládají se na pevný disk) a že používají tzv. index typu hash, který umožňuje obzvláště rychlý přístup k záznamům. Oproti tabulkám typu MyISAM a InnoDB mají tabulky typu HEAP řadu funkčních omezení:

- Záznamy můžete prohlížet pouze pomocí znaků = nebo <=>.
- Není podporován *AUTO_INCREMENT*.
- Indexy lze vytvořit pouze pro sloupce *NOT_NULL*.

Tabulky typu HEAP byste pak měli používat pouze tehdy, pokud chcete maximální rychlostí spravovat malá množství dat. Vzhledem k tomu, že se tento typ tabulek ukládá pouze do paměti RAM, tabulky se po ukončení MySQL odstraní.

2.5.5 CVS

Je jednoduchý souborový formát určený pro výměnu tabulkových dat. Soubor ve formátu CSV sestává z řádků, ve kterých jsou jednotlivé položky odděleny znakem čárka (.). Hodnoty položek mohou být uzavřeny do uvozovek, což umožňuje, aby textové položky obsahovaly čárku. Pokud textové položky obsahují uvozovky, jsou zdvojeny.

Jelikož se v některých jazycích včetně češtiny čárka používá v číslech jako oddělovač desetinných míst, existují varianty, které používají jiný znak pro oddělování položek než čárku, nejčastěji středník, případně tabulátor (taková varianta se pak někdy označuje jako TSV). Variantu se středníkem (ale stále pod názvem CSV) používá např. Microsoft Excel v české verzi Microsoft Windows. Díky jednoduchosti, nenáročnosti a čitelnosti i bez specializovaného software se tento formát používá pro výměnu informací mezi různými systémy. Ke stejnému účelu se dnes používá i modernější a univerzálnější formát XML.

2.5.6 Dočasné tabulky

U většiny z výše uvedených typů tabulek je možné vytvořit tabulky pouze dočasně. Tyto tabulky se automaticky odstraní, jakmile skript PHP ukončí připojení k MySQL. Dočasné tabulky jsou pro ostatní skripty PHP či pro jiné MySQL neviditelné, takže dva skripty PHP mohou používat vedle sebe dvě dočasné tabulky se stejným jménem, aniž by došlo k nějakým konfliktům.

Dočasné tabulky nepředstavují žádný vlastní typ tabulek, ale víceméně se jedná o variantu již zmíněných typů tabulek. Dočasné tabulky si občas vytváří sám MySQL, pokud potřebuje zpracovat složité dotazy obsahující příkaz *SELECT*.

2.6 Normální formy

Normální formy budu prezentovat na ukázkové databázi *Filmy*. V databázi budou uloženy informace o filmech, jako je třeba autor, režie, rok vydání atd. Tyto informace můžeme ukládat i bez databáze, například jako jednoduchý seznam v textovém formátu:

- Robert Zemeckis, Tom Hanks: *Forrest Gump*, USA, 1994
- Frank Darabont, Tom Hanks: *Zelená Mile*, USA, 1999
- Frank Darabont, Morgan Freeman: *Vykoupení z věznice Shawshank*, USA, 1994
- Jan Hřebejk: *Pelíšky*, ČR, 1998

Tento seznam v podstatě obsahuje všechny potřebné informace. Proč se vůbec převodem tohoto textu do databáze zabývat? Několik důvodů se najde:

- Výše uvedený seznam umožňuje jednoduché prohledávání.
- Data v této podobě nelze nijak třídit.
- Například nelze vytvořit seznam všech filmů od daného režiséra.

Možná si tedy řeknete, že není nic jednoduššího, než výše vytvořený seznam umístit do tabulky databáze. Vytvoříme tedy novou tabulku s následujícími sloupci:

Titul	Puvod	Rok	Osoba1	Osoba2
Forrest Gump	USA	1994	Robert Zemeckis	Tom Hanks
Vykoupení z věznice Shawshank	USA	1994	Frank Darabont	Morgan Freeman
Zelená Mile	USA	1999	Frank Darabont	Tom Hanks
Pelisky	CR	1998	Jan Hřebejk	

obr 1. Vytvoření databáze filmů

Tato vytvořená tabulka je na první pohled nedokonalá je krásně vidět, že se v ní pro film nedá zadat více než dvě osoby. Co provedete s filmem na němž se podílely tři a více osoby? Vložíte další sloupce podle vzoru, které potom u dalších filmů zůstanou zcela nevyužité?

2.6.1 První normální forma

Definice první normální formy vypadá takto:

- Musí se odstranit všechny sloupce, které mají stejný obsah.
- Pro každou skupinu navzájem svázaných dat se musí vytvořit zvláštní tabulka.
- Každý záznam se identifikuje pomocí primárního klíče.

ID	Titul	Puvod	Rok	Osoba
1	Forrest Gump	USA	1994	Robert Zemeckis
2	Forrest Gump	USA	1994	Tom Hanks
3	Zelená Mile	USA	1999	Frank Darabont
4	Zelená Mile	USA	1999	Tom Hanks
5	Vykoupeni z veznice Shawshank	USA	1994	Frank Darabont
6	Vykoupeni z veznice Shawshank	USA	1994	Morgan Freeman
7	Pelisky	CR	1998	Jan Hrebejk

obr 2. Databáze filmů v 1NF

Nyní si s počtem osob nemusíme dělat žádné starosti. Do tabulky se nyní vejdou všichni. Zaplatíme za to docela hodně, obsah sloupci *Titul*, *Puvod* a *rok* se opakuje u každé osoby.

2.6.2 Druhá normální forma

Pravidla pro vytvoření tabulky v druhé normální formě jsou následující:

- Pokaždé, když se začne v sloupcích jejich hodnota opakovat, je potřeba vytvořit více menších tabulek.
- Tabulky se musí spojovat pomocí cizích klíčů.

ID	Titul	Puvod	Rok
1	Forrest Gump	USA	1994
2	Zelená Mile	USA	1999
3	Vykoupeni z veznice Shawshank	USA	1994
4	Pelisky	CR	1998

ID	Osoba
1	Robert Zemeckis
2	Tom Hanks
3	Frank Darabont
4	Morgan Freeman
5	Jan Hrebejk

obr 3. Tabulka titulů a osob

titulID	osobaID
1	1
1	2
2	2
2	3
3	3
3	4
4	5

obr 4. 2NF

Tento krok patří mezi nejtěžší a nejabstraktnější, a to pravděpodobně proto, že tabulka *2NF* neodpovídá žádnému lidskému uvažování. Pro ruční správu dat by byla tabulka

2NF naprosto k ničemu. Počítače ale nejsou lidé. Pro počítač nepředstavuje slučování dat žádný problém.

2.6.3 Třetí normální forma

Tření normální forma představuje jedno jediné pravidlo. Sloupce, které nejsou bezprostředně závislé na primárním klíči, je nutno odstranit. Jinými slovy umístit je do jiné tabulky. V ukázkovém příkladu se výše uvedená definice týká sloupce *Puvod* v tabulce *Titul*.

ID	Titul	IDpuvod	Rok	ID	Puvod
1	Forrest Gump	1	1994		
2	Zelená Mile	1	1999	1	USA
3	Vykoupení z veznice Shawshank	1	1994	1	USA
1	Pelisky	2	1998	2	CR

obr 5. 3NF

Celá ukázková databáze filmů tedy bude obsahovat čtyři tabulky.

2.7 Relace

Když se databáze rozdělí na několik tabulek, tak se musí mezi tabulkami vytvořit nějaké vztahy. Tyto vztahy se v případě databází nazývají *relace*. Mezi dvěma tabulkami mohou být v podstatě tři druhy relací.

2.7.1 1:1

Relace 1:1 představuje jednoznačný vztah mezi dvěma tabulkami. Každý záznam v první tabulce odpovídá přesně jednomu záznamu v druhé tabulce. Tato relace se vyskytuje poměrně vzácně.

Dejme tomu že máme tabulku *zaměstnanec*, ve které jsou uloženy záznamy jako například *id*, *jméno*, *příjmení*, *datum narození*, *pozice*, *plat*, atd. Nyní budeme chtít tuto tabulku rozdělit na dvě. V jedné tabulce budeme mít záznamy jako je *id*, *jméno*, *příjmení*. V druhé tabulce bude zbytek záznamů. Existuje několik důvodů proč tabulku rozdělit. První důvod je zabezpečení. Dalším důvodem je rychlost, obsahuje-li tabulka příliš mnoho sloupců a při dotazu využíváme některé z nich. Dělení tabulek má, ale i své nevýhody, jako je třeba potřeba udržovat data v tabulkách synchronizované.

2.7.2 1:n

Dalším druhem je relace 1:N. Tato relace opětovně vzniká mezi dvěma tabulkami, kde jedna hodnota primárního klíče v hlavní tabulce odpovídá několika hodnotám pole v druhé tabulce. Jako příklad uvedu vztah mezi tabulkou se seznamem kupujících a tabulkou objednávky. Každý kupující totiž může učinit několik objednávek, ale naopak každá objednávka náleží jednomu kupujícímu.

2.7.3 m:n

Posledním typem relace je $m:n$. Tato relace vzniká mezi dvěma tabulkami, kde každý záznam může odpovídat několika záznamům z druhé tabulky. V případě relací $m:n$ je třeba vytvořit mezi oběma tabulkami další tabulku, pomocí níž relaci $m:n$ převedeme na dvě relace typu $1:n$.

Například evidence studentů na univerzitě a jejich seznamy zkoušek. Chceme-li vytvořit databázi, která bude obsahovat údaje o tom, který student složil jakou zkoušku, budeme potřebovat další tabulku, která bude zprostředkovávat vztah mezi tabulkou obsahující seznam studentů a tabulkou obsahující seznam všech zkoušek.

2.8 Klíče

Relace úzce souvisí s primárním a cizím klíčem. Tato kapitola se bude věnovat pojmům a využití klíčů. Bohužel se zde zcela neobejdeme bez příkazů jazyka SQL, s nimiž se budeme seznamovat až v nadcházejících kapitolách.

2.8.1 Primární klíč (primary key)

Úkolem primárního klíče je co možná nejrychlejší nalezení určitého záznamu v tabulce. Tato operace se musí provést pokaždé, když potřebujeme získat data z více tabulek a tato situace nastává velmi často. V MySQL můžeme použít rovněž primární klíč, který se skládá z více položek tabulky. Na druhou stranu v každém případě musí mít primární klíč tyto vlastnosti:

- Pole, které obsahuje primární klíč se musí nastavit jako index (primární index) aby vyhledávání záznamů probíhalo co možná nejrychleji.
- Na pole pro primární klíč se daleko více hodí to pole, které obsahuje celé číslo, než pole, které obsahuje řetězec s proměnou délkou.
- Dále platí, že se obsah pole s primárním klíčem použije v ostatních tabulkách jako cizí klíč a i zde je daleko účinnější, pokud je cizí klíč co nejvíce celiství.

V případě MySQL se pro políčko s primárním klíčem zpravidla používá pole s datovým typem *INT* nebo *BIGINT* a to s atributy *NOT NULL*, *AUTO_INCREMENT PRIMARY KEY*. Na názvu sloupce s primárním klíčem vůbec nezáleží. Může být prakticky cokoli, co dává smysl. Jako třeba *ID*, *Number*, *NumberID*, *Cislo*.

2.8.2 Cizí klíč (foreign key)

Úkolem cizího klíče je ukazovat na záznam v podřízené tabulce. Tento odkaz se však vytvoří až v okamžiku formulování dotazu na databázi, například v této formě:

- ```
SELECT Titul, Puvod FROM Filmy, Puvody
WHERE Filmy.IDpuvod=Puvody.ID
ORDER BY Titul
```

Tento výše uvedený dotaz vytvoří v prvním sloupci abecední seznam všech titulů a ve druhém sloupci jejich původ.

Při deklaraci tabulky nehraje cizí klíč žádnou významnou roli. Pro MySQL je pole označené jako cizí klíč stejné jako kterékoli jiné pole. Nejsou potřeba ani žádná zvláštní klíčová slova. Neměli byste ale zapomenout na to, aby bylo pole s cizím klíčem stejného datového typu jako pole s primárním klíčem, když se to nesplní tak bude odkazování velmi pomalé.

## 2.9 Základy jazyka SQL

Tato kapitola bude popisovat úvod databázového jazyka SQL. Tento jazyk se používá pro formulování dotazů na databázový systém. Příklady jsou demonstrovány na databázi, která je využita v praktické části této práce. Její struktura je popsána v kapitole 4.3.4.

### 2.9.1 Datové typy

Každá tabulka v MySQL obsahuje několik sloupců. Do každého sloupce se musí zadat příslušný datový typ a nezáleží na tom, jestli je tabulka typu MyISAM, nebo InnoDB.

#### 2.9.1.1 Celá čísla

Standardně se pro všechna celá čísla v MySQL používá znaménko. Například do sloupce *TINYINT* můžeme zadat čísla z intervalu -127 až + 128. Pokud, chceme používat výhradně kladná čísla, musíme u definice sloupce zadat omezení *UNSIGNED*.

- TINYINT(m) – 8-bit integer (1 BYTE)
- SMALLINT(m) – 16-bit integer (2 BYTE)
- MEDIUMINT(m) – 24-bit integer (3 BYTE)
- INT(m), INTEGER(m) – 32-bit integer (4 BYTE)
- BIGINT(m) – 64-bit integer (8 BYTE)

Nepovinný parametr *m* udává šířku sloupce ve výsledcích při použití příkazu *select* (maximum display width). Nijak neovlivňuje povolený rozsah číslíc.

### 2.9.1.2 Čísla s pevnou a s plovoucí desetinnou čárkou

Rozdíl mezi číslem s pevnou a s pohyblivou desetinnou čárkou spočívá v jeho vnitřní interpretaci. Čísla s plovoucí čárkou se v MySQL ukládají v binární podobě, ale čísla s pevnou čárkou se ukládají jako řetězce.

- FLOAT(m,d) – Číslo s plovoucí desetinnou čárkou s přesností na 8 desetinných míst. Nepovinné parametry *m* a *d* udávají počet cifer před a za desetinnou čárkou.
- DOUBLE(m,d), REAL(m,d) – Čísla s plovoucí desetinnou čárkou s přesností na 16 desetinných míst
- DECIMAL(p,s) - Čísla s pevnou desetinnou čárkou. Parametr *p* udává celkový počet cifer(maximálně 64), parametr *s* udává počet míst za desetinnou čárkou(maximálně 30). Jako standard je nastaveno DECIMAL(10,0).

### 2.9.1.3 Datum a čas

Z níže uvedených formátů datumů a časů se nejčastěji používá formát *DATETIME*. Pole s tímto formátem se hodí k ukládání libovolných časových údajů. Datový typ *TIMESTAMP* hraje svojí roli jinde. Sloupce, kde se použije tento formát, spravuje MySQL automaticky. Při každé změně záznamu automaticky ukládá MySQL do tohoto sloupce čas změny. Sloupce s *TIMESTAMP* má smysl používat pouze tam, kde si přejeme automaticky zaznamenávat časy posledních změn. Nejsou tedy v žádném případě určeny k ukládání libovolných časových údajů.

- DATE – datum ve formátu ‘2000-11-30’. Hodnoty z intervalu od 1000-01-01 do 9999-12-31.
- TIME – čas ve formátu ‘22:30:59’. Hodnoty z intervalu od +/- 838:59:59.
- DATETIME – kombinace z údajů DATE a TIME ve formátu ‘2000-11-30 22:30:59’
- YEAR – letopočet 1900-2155
- TIMESTAMP – datum a čas, obsah sloupce se při každé změně v MySQL automaticky aktualizuje.

#### **2.9.1.4 Řetězce znaků**

Pro ukládání řetězců znaků se zpravidla používá datový typ *VARCHAR(n)*, kde *n* určuje maximální počet znaků v řetězci.

- CHAR(n) – znakové řetězce s předem určenou délkou (maximálně 255 znaků)
- VARCHAR(n) – znakové řetězce s proměnou délkou
- TINYTEXT – znakové řetězce s proměnou délkou (maximálně 255 znaků)
- TEXT - znakové řetězce s proměnou délkou (maximálně  $2^{16}-1$  znaků)
- MEDIUMTEXT - znakové řetězce s proměnou délkou (maximálně  $2^{24}-1$  znaků)
- LONGTEXT - znakové řetězce s proměnou délkou (maximálně  $2^{32}-1$  znaků)

#### **2.9.2 Vytvoření databáze**

Pro vytváření databází a tabulek se většinou používá nějaký program na správu databáze například phpMyAdmin. Výhoda je, že nemusíme znát určité příkazy a zamezíme tím chybám ve vytváření tabulek a databází. Před vytvoření první tabulky musíme nejdříve vytvořit databázi. Pro vytvoření databáze musíme mít přidělená práva v MySQL a poté databázi vytvoříme příkazem:

- CREATE DATABASE *test* ;

Výše uvedeným příkazem vytvoříme databázi, která se bude jmenovat *test*. Jest-li máme vytvořenou databázi musíme jí nastavit jako aktivní, abychom s ní mohli nadále pracovat a to uděláme příkazem *use*. To je dobré pro to, když máme více databází, aby jsme nemuseli při vyhledávání neustále psát do dotazu v jaké databázi má hledat data.

- USE *test*;



### 2.9.3 Vytvoření Tabulek

Po té co jsme vytvořili databázi, se můžeme podívat na to, jak se dají vytvářet tabulky. Nové tabulky se vytvoří příkazem `create table`. Nejlepší bude, když si to ukážeme na příkladu.

- `CREATE TABLE prislusnik (  
OEC INT NOT_NULL,  
Jmeno CHAR(20) NOT_NULL,  
Prijmeni CHAR(20) NOT_NULL,  
Hodnost CHAR(50),  
Vek DATE NOT_NULL,  
PRIMARY KEY(OEC)  
);`

Tento příklad vytvoří v databázi tabulku, která se bude jmenovat *prislusnik*. Bude obsahovat pět sloupců. První sloupec, který má název *OEC* a je datového typu *INT* což znamená, že může obsahovat jen celá čísla. Taktéž nesmí být nulový, protože obsahuje omezení *NOT\_NULL*. Sloupce s názvem *Jmeno*, *Prijmeni* jsou typu *CHAR(20)*. Tyto sloupce mohou obsahovat řetězce znaků, které mohou mít maximálně 20 znaků a nesmějí být nulové. Sloupec *Vek* je datového typu *DATE* zde vkládáme datum narození. Poslední řádek s klíčovým slovem *PRIMARY KEY* určuje primární klíč tabulky. Podle takto označeného sloupce se jednoznačně určí právě jeden řádek tabulky. Automaticky je na takovýto sloupec aplikována podmínka, že data v něm musí být jedinečná. Další základní omezení pro sloupce jsou:

- *UNIQUE* – hodnota musí být jedinečná, zakázaná duplicita
- *UNSIGNED* – u celých čísel se vynechává znaménko
- *AUTO\_INCREMENT* – automatické číslování
- *PRIMARY KEY* – primární klíč
- *FOREIGN KEY* – cizí klíč, odkazuje na sloupec jiné tabulky
- *CHECK* – podmínka pro hodnoty ve sloupci
- *DEFAULT* – předdefinovaná hodnota

### 2.9.4 Změna tabulky

Příkazem *ALTER TABLE* se dají provádět různé úpravy tabulek, jako například přidávání nebo odstraňování sloupců, změna vlastností sloupců, změna datového typu, definování nebo odebrání indexů. Následující příklad ukazuje, jak zvýšit maximální počet znaků ve sloupci *Jmeno* v tabulce *prislusnik* z dvaceti na padesát znaků.

- ALTER TABLE *prislusnik* CHANGE *Jmeno* *Jmeno* CHAR(50) NOT NULL

Na první pohled je příkaz trochu matoucí, dvojité použití názvu sloupce *jmeno*. Poprvé se výraz *jmeno* týká stávajícího názvu sloupce, podruhé se týká nového názvu sloupce. Ten, ale v našem případě zůstal stejný. Kdyby se měl název změnit třeba na *JmenoPadesat*, pak by zápis vypadal takto:

- ALTER TABLE *prislusnik* CHANGE *Jmeno* *JmenoPadesat* CHAR(50) NOT NULL

### 2.9.5 Vložení záznamů do tabulek

Příkazem *INSERT INTO* vkládáme do tabulek nové záznamy. Za názvem tabulky musíme nejprve uvést seznam názvů sloupců a poté seznam vkládaných hodnot. Sloupce kde se vyskytují standardní hodnoty, nebo smí obsahovat hodnotu NULL, a sloupce s omezením *AUTO\_INCREMENT* zadávat nemusíme. Zkusíme vytvořit jednoduchý příklad, kde vložíme data do tabulky *prislusnik*.

- INSERT INTO *prislusnik* (OEC,*jmeno*,*prijmeni*,*hodnost*,*vek*) VALUES(793555,'Karel','Formánek','mjr',1960-1-1)

Názvy sloupců můžeme vynechat, když zadáme všechny hodnoty. To je výhoda, u velkých tabulek, které mají třeba osm sloupců. Při využití této vlastnosti pak lze dosáhnout stejného efektu, jako u předchozího příkladu, pomocí následujícího zápisu:

- INSERT INTO *prislusnik* VALUES(793555,'Karel','Formánek','mjr',2000-1-1)

### 2.9.6 Změna obsahu záznamů

Když už nějaký záznam do tabulky vložíme a chceme ho změnit, aniž by jsme ho chtěli celý vymazat a znovu zadat, použijeme příkaz *UPDATE*. Ukážeme si například jak v tabulce *prislusnik* změním jméno u příslušníka s oec 793599. Jméno Josef zkusíme změnit na Jan.

- UPDATE *prislusnik* SET *jmeno*='Jan' WHERE OEC=793599

Ale pozor na jednu záhadnou věc, když uvedeme *UPDATE* bez příkazu *WHERE*, tak příkaz platí pro celou tabulku! Níže uvedený příkaz by udělal neplechu v celé tabulce a to tím, že by všem příslušníkům nastavil jméno na Jan.

- UPDATE *prislusnik* SET *jmeno*='Jan'

### 2.9.7 Odstranění záznamů

Nejjednodušším příkazem, je příkaz pro odstranění záznamu *DELETE*. Všechny záznamy specifikované pomocí příkazu *WHERE* se po zadání *DELETE* odstraní. Není zde nutné udávat sloupce, protože se odstraní celý záznam.

- *DELETE FROM prislusnik WHERE prislusnik.oec=793111*

### 2.9.8 Vyprázdnění tabulky

Když by jsme potřebovali vyprázdnit data z tabulky, ale nechtěli by jsme přijít o celou strukturu tabulky, použijeme příkaz *truncate*. MySQL ve skutečnosti tabulku smaže a podle její definice ji znovu vytvoří. Z toho vyplývá skutečnost, že hodnoty generované pomocí *AUTO\_INCREMENT* se začnou počítat znovu od začátku.

- *TRUNCATE TABLE prislusnik*

Tento příkaz vyprázdní tabulku *prislusnik*.

### 2.9.9 Odstranění databáze, tabulky

Pokud nastane situace, kde nepomůže ani změna tabulky, ani její vyprázdnění přijde na řadu její odstranění. Pro odstranění se používá příkaz *drop*. Stejný příkaz lze použít i na odstranění celé databáze.

- *DROP prislusnik*

Výše uvedený příkaz odstraní z databáze tabulku *prislusnik*.

- *DROP bp*

Tento příkaz vymaže celou databázi, která se jmenuje *bp* a to jak všechny data tak všechny tabulky.

### 2.9.10 Jednoduché dotazy SELECT

Nejjednodušším možným dotazem je *SELECT \* FROM* jméno tabulky. Výsledkem je zobrazení všech záznamů v tabulce. Znak hvězdička znamená, že se mají zobrazit všechny záznamy v tabulce.

- *SELECT \* FROM prislusnik*

Výsledek výše uvedeného dotazu vypadá takto<sup>1</sup>:

| OEC    | Jmeno | Prijmeni | Hodnost | Vek        |
|--------|-------|----------|---------|------------|
| 793599 | Josef | Kaláb    | nrap    | 1985-11-19 |
| 793444 | Petr  | Pirkl    | por     | 1980-06-05 |

obr 6 Výpis tabulky příslušník

### 2.9.11 Omezení výpisu na několik sloupců

Můžou nastat případy, kdy nepotřebujeme vypsat z tabulky všechny sloupce, ale jen některé, které budeme potřebovat. To se dá zařídit, úpravou předchozího dotazu. Místo znaku hvězdička napíšeme jméno sloupce, nebo více sloupců, které chceme vypsat.

- `SELECT OEC,Hodnost FROM prislusnik`

Tento dotaz vypíše obsah sloupců *OEC* a *Hodnost* všech záznamů uložených v tabulce *prislusnik*.

| OEC    | Hodnost |
|--------|---------|
| 793599 | nrap    |
| 793444 | por     |

obr 7 Omezený výpis tabulky

### 2.9.12 Zjištění počtu řádků

Někdy nastane situace, že nás nebude zajímat obsah tabulky, ale budeme pouze chtít zjistit, kolik záznamů se v tabulce vyskytuje. S tímto problémem nám pomůže příkaz *COUNT*:

- `SELECT COUNT(id) as Pocet_zaznamu FROM clenjpo`

Tento příkaz nám vrátí počet záznamů v tabulce *clenjpo*. Kdybychom místo jména sloupce v našem případě *id* zadali jakýkoli jiný sloupec, nebo znak hvězdička (pro všechny sloupce), tak se výsledek nijak nezmění. V příkazu se objevilo pojmenování sloupce výpisu *as Počet\_zaznamu*. Výsledek bude vypadat takto.

| Pocet_zaznamu |
|---------------|
| 1488          |

obr 8 Počet záznamů

---

<sup>1</sup> Tabulka příslušník obsahuje více záznamů, byla oříznuta pro zjednodušení výpisu.

### 2.9.13 Počet odlišných hodnot

Pokud se ve sloupci tabulky několikrát opakují stejné hodnoty a my chceme zjistit, kolik různých hodnot se ve sloupci vyskytuje, pak použijeme klíčové slovo *DISTINCT*.

- `SELECT COUNT( DISTINCT Prijmeni ) as Pocet_zaznamu FROM clenjpo`
- `SELECT COUNT( DISTINCT id ) as Pocet_zaznamu FROM clenjpo`

Výsledkem nemusí být nutně to samé číslo. V prvním případě zjistíme počet příjmení, ale nesmí se opakovat (být stejné). Ve druhém případě zjistíme počet všech různých id.

| Pocet_prijmeni | Pocet_id |
|----------------|----------|
| 794            | 1488     |

obr 9 Omezení *DISTINCT*

### 2.9.14 Omezení počtu vypsaných záznamů

Kromě omezování výpisu počtu sloupců můžeme omezovat i počet vypsaných záznamů. Ukažme si to na konkrétním příkladu. Máme tabulku *ClenJpo*, která obsahuje přes tisíc záznamů, ale nás by zajímalo prvních 5 záznamů (například pro zobrazení v dokumentu HTML). Při zpracování zbývajících záznamů by se jen zbytečně plýtvalo výpočetním časem, operační pamětí i kapacitou sítě. K tomu využijeme příkaz *LIMIT n*, který zajistí zobrazení pouze *n* prvních záznamů. Následující příklad vrací z tabulky *JPO* pět záznamů.

- `SELECT NazevJpo FROM jpo LIMIT 5`

Výsledek výše uvedeného příkazu:

| NazevJpo             |
|----------------------|
| Bestovice            |
| Bohousová - Záchlumí |
| Bošín                |
| Brandýs nad Orlicí   |
| Bulina               |

obr 10 Výpis tabulky pomocí *LIMIT*

Výše uvedený dotaz můžeme ještě upravit pomocí příkazu *LIMIT offset,n*. Výraz *offset* udává, u kterého záznamu se má začít<sup>2</sup>.

---

<sup>2</sup> Pozor, záznamy se číslují od nuly.

- `SELECT NazevJpo FROM jpo LIMIT 5, 5`

Tento dotaz vypíše názvy jpo od pátého záznamu dalších pět.

### 2.9.15 Třídění dotazů

Příkaz `SELECT` vrací výsledky v prakticky libovolném pořadí, většinou jak data do tabulky vložíme. Pokud chceme mít výsledky dotazu seřazeny, musíme k tomu použít příkaz `ORDER BY`. Níže uvedený dotaz vrací seznam jmen a příjmení příslušníků seřazený podle abecedy a výpis je omezený na pět záznamů.

- `SELECT jmeno,prijmeni FROM clenjpo ORDER BY prijmeni LIMIT 5`

Výsledek pak vypadá takto:

| jmeno    | prijmeni |
|----------|----------|
| Miloslav | Abrahám  |
| Roman    | Abrahám  |
| Roman    | Absolon  |
| Jirka    | Adamec   |
| Petr     | Adamec   |

obr 11 Seřazený výpis

Chceme-li položky setříděny sestupně, pak musíme za příkaz `ORDER BY` doplnit klíčové slovo `DESC`. Příkaz pak bude vypadat takto:

- `SELECT jmeno,prijmeni FROM clenjpo ORDER BY prijmeni DESC LIMIT 5`

### 2.9.16 Výběr záznamů na základě podmínky

Často potřebujeme v tabulkách vyhledávat záznamy, podle námi zvolených kritérií. Podmínky výběru se zadávají pomocí klíčového slova `WHERE`. V prvním příkladu si ukážeme jak zobrazit příslušníky, kteří se jmenují Josef.

- `SELECT jmeno,prijmeni FROM prislusnik WHERE jmeno='Josef'`

Pomocí tohoto příkazu dostaneme výslednou tabulku:

| jmeno | prijmeni |
|-------|----------|
| Josef | Kaláb    |

obr 12 Výpis pomocí LIKE

V druhém příkladu si ukážeme jak porovnávat řetězce pomocí operátoru *LIKE*. Dotaz vypíše všechny příslušníky, v jejichž jméně se vyskytuje řetězec *os*. U operátoru *LIKE* slouží *%* jako zástupný znak pro libovolný řetězec znaků. Výsledek bude stejný jako v předchozím případě.

- `SELECT jmeno,prijmeni FROM prislusnik WHERE jmeno LIKE '%os%'`

V posledním příkazu si ukážeme jak vypsát člena jpo, který mám třeba id 605.

- `SELECT id,jmeno,prijmeni FROM clenjpo WHERE id=605`

| id  | jmeno  | prijmeni |
|-----|--------|----------|
| 605 | Michal | Fuksa    |

obr 13 Výpis dotazu s podmínkou

### 2.9.17 Spojení dat z více tabulek

Dosud jsme zkoušeli vypisovat záznamy pouze z jedné tabulky, v nichž jsme používali příkaz *SELECT*. U relačních databází však jsou často tabulky spolu navzájem svázány. Z toho vyplývá, že musí existovat možnost, při použití příkazu *SELECT* navzájem spojovat data z různých tabulek První pokus pro vytvoření seznamu všech příslušníků a akci těžce ztroskotá.

- `SELECT prislusnik.prijmeni, akce.nazev FROM akce, prislusnik`

Tento nepovedený dotaz totiž vrací seznam všech kombinací, jmen příslušníků a názvů akcí. Pokud mají dotazy přes několik tabulek zobrazit použitelné výsledky, musí se vždy přesně zadat, jakým způsobem mají být data navzájem spojena. Toto spojení dat máte možnost nastavit pomocí příkazu *WHERE*.

Ve výše uvedeném příkazu se nachází jedna nová věc, která je důležitá u spojování tabulek. `SELECT prislusnik.prijmeni,` znamená. Vypiš z tabulky *prislusnik* kde se vybere sloupec *prijmeni*.

Druhý již funkční příklad jak by mělo vypadat správné spojení tabulek.

- `SELECT prislusnik.prijmeni, akce.nazev FROM akce,prislusnik WHERE prislusnik.OEC=akce.OEC`

Existuje řada možností, jak se k výsledku dostat. Jednou z možností je použít příkaz *LEFT JOIN*.

- `SELECT prijmeni, nazev FROM akce LEFT JOIN prislusnik ON prislusnik.OEC=akce.OEC`

V obou předchozích případech je výsledek stejný, takže je jedno, který způsob budeme používat.

| <b>prijmeni</b> | <b>nazev</b> |
|-----------------|--------------|
| Kaláb           | Školení JPO  |
| Kaláb           | TFA          |
| Kaláb           | S-16         |

**obr 14** Spojení dvou tabulek



## **3 PHP**

V následující kapitole si povíme něco málo o historii PHP a podíváme se na úvod programování v PHP.

### **3.1 Co je PHP?**

PHP je serverový skriptovací jazyk navržený pro potřeby webových stránek. To znamená, že vše co PHP provádí, neprobíhá na straně klienta jako například u JavaScriptu, ale zpracovává se na straně serveru. Ten generuje HTML výstup, který je odeslán uživateli. PHP je Open Source, tedy volně šiřitelná technologie. PHP není závislé na platformě a není vázané s žádným konkrétním serverem, může tedy běžet kdekoli

### **3.2 Historie PHP**

Počátky vývoje PHP sahají do roku 1995, kdy Rasmus Lerdorf vytvořil jednoduchý systém evidence přístupů ke svému webu, nejdříve v jazyce Perl, poté v C. Ten se rozšířil mezi další uživatele, kteří přicházeli s požadavky na vylepšení. Vznikl tak systém Personal Home Page Tools, později Personal Home Page Construction Kit.

#### **➤ PHP/FI**

Rasmus Lerdorf vytvořil kromě samotného jazyka i nástroj umožňující začleňování SQL dotazů a tím zpřístupnění databází na serveru Forms Interpreter (FI). V roce 1997 bylo PHP/FI 2.0 oficiálně uvolněno, brzy potom vznikla verze PHP 3.0.

#### **➤ PHP 3**

PHP 3.0 vytvořili Andi Gutmans a Zeev Suraski. Původní zkratka Personal Home Page dostává nový význam – PHP: Hypertext Preprocessor. Což znamená skriptovací programovací jazyk, určený především pro programování dynamických internetových stránek. Nejčastěji se začleňuje přímo do struktury jazyka HTML, což lze využít při tvorbě webových aplikací. PHP lze použít i k tvorbě konzolových a desktopových aplikací.

#### **➤ PHP 4**

PHP 4.0, oficiálně uvolněné v roce 2000 je výkonnější verzí než PHP 3. Přibila například i podpora pro mnoho WWW serverů, HTTP sessions, buffering výstupu a nové jazykové konstrukty. Rovněž způsoby zpracování vstupů uživatele je bezpečnější.

#### ➤ **PHP 5**

V červnu 2003 byla oficiálně uvolněna beta verze PHP 5. Největší změna je v objektovém modelu, PHP se přibližuje ostatním jazykům podporujícím objektově orientované programování (OOP). Snazší je také obsluha chyb.

### **3.3 Vkládání PHP do HTML**

PHP soubory se označují obvykle příponou php, případně php3, php4 nebo phtml. Jazyk PHP byl vytvořen právě pro vytváření webových stránek, takže PHP skripty můžeme volně vkládat do HTML kódu, jen je třeba je oddělit.

### **3.4 PHP značky**

K oddělení PHP kódu od HTML slouží PHP značky („<?“ a „?>“). PHP značky označují, kde PHP kód začíná a kde končí. Všechn text mezi těmito značkami je interpretován jako kód PHP, všechny text mimo značky je považován za HTML kód. Můžeme použít různé styly značek PHP.

#### ➤ **Krátký styl**

Nejjednodušší styl, na většině serverů zřejmě funguje, ale dostupnost už není nijak zaručena, takže se také může stát, že fungovat nemusí.

- <? ... PHP kód ... ?>

#### ➤ **XML styl**

Tento styl je považován za nejlepší. Je doporučený pro PHP3 a PHP4, jeho dostupnost je garantována na všech serverech, navíc se může používat v XML dokumentech.

- <?php ... PHP kód ... ?>

### 3.5 Vkládání HTML do PHP

Pokud potřebujeme vložit naopak HTML do PHP (např. v případě generovaného HTML výstupu), můžeme použít příkaz echo. Níže uvedený příkaz vypíše obyčejný nadpis první úrovně.

- `<?php echo "<h1>nadpis</h1>"; ?>`

### 3.6 Datové typy

U různých programovacích jazyků je nutno při deklarování proměnných uvést jejich typ. Mezi tyto jazyky, patří mimo jiné Java, C, C++, Fortran, Delphi a další. U jazyka PHP tato nepříjemná povinnost odpadá. Samo totiž podle obsahu proměnné pozná, o jaký typ se jedná. PHP rozlišuje osm základních typů.

- BOOLEAN – Hodnota proměnné může nabývat buď true, nebo false.
- INTEGER – Kladné a záporné celočíselné hodnoty.
- FLOAT – Hodnota s plovoucí desetinnou čárkou.
- ARRAY – Datový typ pole, jedná se o složený typ, může obsahovat ostatní datový typy.
- STRING – Řetězce v PHP mohou obsahovat 256 znaků.
- OBJECT – Objektová proměnná může obsahovat různé prvky PHP.
- RESOURCE – Zdroje jsou speciální datové typy. Slouží, například pokud potřebujeme otevřít soubor.
- NULL – Zvláštní datový typ, který nemá žádnou hodnotu.

### 3.7 Proměnné

V PHP začínají názvy proměnné vždy znakem dolaru “\$”. Musíme ovšem dodržovat některé omezení. V názvu se mohou vyskytovat pouze znaky ASCII, dále číslice a znak podtržítka “\_”. Za znakem dolaru nesmí následovat číslice. PHP rozlišuje v názvu malé a velké písmena. Správně zadané proměnné vypadají například takto:

- `$x` , `$Prijmeni` , `$_50`

#### 3.7.1 Práce s proměnnými

Pokud chceme vypsat nebo zjistit datový typ proměnné, použijeme funkci `gettype()`. Další funkcí je nastavení datového typu proměnné uživatelsky, provádí se příkazem `settype(proměnná,typ)`. Dalšími důležitými funkcemi pro práci s proměnnými jsou

isset(), unset() a empty(). Funkce unset() odstraní proměnnou a uvolní tak paměť jiným operacím, isset() zjišťuje, zda byla proměnné přiřazena hodnota, pokud ano, vrací true - naproti tomu empty() je přesný opak isset(), vrací pravdu, když hodnota nebyla proměnné přiřazena.

### 3.7.2 Předem nadefinované proměnné

Když spustíme skript PHP, celá řada dat je uložena v předem nadefinovaných proměnných. Přesný obsah proměnných však závisí na několika faktorech. Nejdůležitějšími proměnnými jsou:

- `$_SERVER` – Pole s informacemi o webovém serveru
- `$_GET` – Všechna data z metody *method=“get“*
- `$_POST` - Všechna data z metody *method=“post“*
- `$_COOKIE` – Cookies, která prohlížeč nabízí pro tuto internetovou relaci
- `$_SESSION` – Proměnné, které jsou zaregistrovány v aktuální relaci
- `$GLOBALS` - Všechny globální proměnné

## 3.8 Konstanty

Konstanty, jak již napovídá jejich název, obsahují hodnoty, které se během činnosti programu nemění. Je to například hodnota  $\pi$  (asi 3,14), či bod varu vody při obvyklém atmosférickém tlaku (100 stupňů celsia). U názvu konstant se na rozdíl od proměnných nepoužívá znak dolaru. Pro jejich zápis se ve zdrojovém kódu kvůli lepší čitelnosti a přehlednosti používají velká písmena. Tabulka takzvaných magických konstant obsahuje:

- `_LINE_` - Aktuální číslo řádku
- `_FILE_` - Úplná cesta a název právě otevřeného souboru
- `_FUNCTION_` - Název funkce
- `_CLASS_` - Název třídy
- `_METHOD_` - Název metody

Vlastní konstanty vytvoříme pomocí funkce *define*.

- `define(“MY_ERROR“, 1);`
- `define(“MY_VERSION“, “0.9.2“)`

Konstantu `my_version` nelze nalézt, proto se odkaz na konstantu vyhodnotí jako řetězec a ten se vypíše. Jak jsme už výše řekli, PHP rozlišuje rozdíl mezi malými a velkými písmeny. Správný výsledek vrátí až příkaz zapsaný ve čtvrtém řádku.

### 3.9 Operátory v PHP

V PHP můžeme pracovat s proměnnými i jinak než je jen deklarovat a vypisovat. Můžeme například provádět jednoduché matematické operace, ale i porovnávat proměnné. Důležitou věcí, než se začneme bavit o samotných operátorech, je ujasnit si rozdíl mezi přirovnáním (v PHP se značí "=") a porovnáváním ("=="). Přirovnání jednoduše nastaví hodnotu jednoho prvku druhému, porovnávání je dotaz, zda se dva prvky rovnají či ne. Matematické operátory podporované v PHP jsou:

| Operátor              | Význam         | Příklad                       | Je pravda když:                                          |
|-----------------------|----------------|-------------------------------|----------------------------------------------------------|
| <code>==</code>       | rovnost        | <code>\$x == \$y</code>       | <code>\$x</code> a <code>\$y</code> mají stejnou hodnotu |
| <code>&lt;</code>     | menší než      | <code>\$x &lt; \$y</code>     | <code>\$x</code> je menší než <code>\$y</code>           |
| <code>&gt;</code>     | větší než      | <code>\$x &gt; \$y</code>     | <code>\$x</code> je větší než <code>\$y</code>           |
| <code>&lt;=</code>    | menší či rovno | <code>\$x &lt;= \$y</code>    | <code>\$x</code> je menší či rovno <code>\$y</code>      |
| <code>&gt;=</code>    | větší či rovno | <code>\$x &gt;= \$y</code>    | <code>\$x</code> je větší či rovno <code>\$y</code>      |
| <code>!=</code>       | nerovnost      | <code>\$x != \$y</code>       | <code>\$x</code> se nerovná <code>\$y</code>             |
| <code>&lt;&gt;</code> | nerovnost      | <code>\$x &lt;&gt; \$y</code> | <code>\$x</code> se nerovná <code>\$y</code>             |

obr 15 Tabulka matematických operátorů

Zvláštní skupinou proměnných, jsou logické proměnné. Pro práci s nimi je definována celá řada operátorů. Ty pracují se dvěma proměnnými. Vyjímkou je jen negace.

| Operátor                | Název                   | Příklad                         | Je pravda když:                                                |
|-------------------------|-------------------------|---------------------------------|----------------------------------------------------------------|
| <code>&amp;&amp;</code> | logický součin (a)      | <code>\$a &amp;&amp; \$b</code> | <code>\$a</code> a <code>\$b</code> bude pravda                |
| <code>and</code>        | logický součin (a)      | <code>\$a and \$b</code>        | <code>\$a</code> a <code>\$b</code> bude pravda                |
| <code>  </code>         | logický součet (nebo)   | <code>\$a    \$b</code>         | <code>\$a</code> nebo <code>\$b</code> bude pravda             |
| <code>or</code>         | logický součet (nebo)   | <code>\$a or \$b</code>         | <code>\$a</code> nebo <code>\$b</code> bude pravda             |
| <code>xor</code>        | negovaný logický součet | <code>\$a xor \$b</code>        | <code>\$a</code> nebo <code>\$b</code> bude pravda, ale ne obě |
| <code>!</code>          | negace                  | <code>!\$a</code>               | <code>\$a</code> je nepravda                                   |

obr 16 Tabulka logických operátorů

### 3.10 Řídicí struktury

Skript může díky podmínkám reagovat na vstupní data a zařídit se podle nich. Pokud například budeme mít stránku přístupnou jak pro registrované, tak i neregistrované uživatele, lze zajistit, že registrovaní uživatelé budou mít přístupných více funkcí. Pomocí podmínek můžeme například rozlišit, jestli se jedná o správce aplikace, nebo o obyčejného uživatele.

### 3.10.1 Příkaz if

Nejčastěji používanými konstrukcemi jsou v programovém kódu dotazy na stav proměnných či konstant. V PHP se k tomuto účelu používá příkaz `if`. Syntaxe se velice podobá té z jazyka C++.

```
○ if (($_POST['user']==") or ($_POST['pass']=="))
 {
 ...
 }else{
 ...
 }
```

Výše uvedený skript hlídá, aby při přihlášení ve formuláři byla všechna pole vyplněna, jak pole uživatel tak pole heslo. Je-li podmínka uvedená v kulatých závorkách splněna, pak se pokračuje prvním blokem příkazů ve složených závorkách. V opačném případě se vykoná druhý blok za slovem *else*. V PHP lze použít také příkaz *elseif* nebo *else if*. Tím můžeme zajistit další větvení programu. Dále je možnost využít modifikaci příkazu `if` takzvaný krátký zápis.

### 3.10.2 Konstrukce switch

Konstrukce představuje náhradu několika příkazů *ifelse*. Konstrukce `switch` slouží k porovnávání stejného výrazu s mnoha hodnotami. Přesně řečeno, je to něco, co bysme zvládli napsat pomocí konstrukce `if`, ale `switch` může být daleko šikovnější. Konstrukce `switch` pracuje tak, že vyhodnotí výraz za slovem `switch` a pak hledá příslušný řádek *case* s odpovídající hodnotou.

```
○ Switch ($zprava){
 Case 0: echo "Nemáte žádné zprávy\n";
 break;

 Case 1: echo "Máte jednu novou zprávu\n";
 break;

 Case 2 : echo "Máte dvě nové zprávy\n";
 break; }
```

### 3.11 PHP cykly

Cykly patří mezi jedny ze základních kamenů programovacích jazyků. Slouží k opakovanému provádění určitého bloku příkazů, obvykle dokud není splněna určitá podmínka. To nám může často ušetřit spoustu práce. Dobrým příkladem může být třeba tvorba tabulek, kde se opakují tagy `<TR>` a `<TD>` pro každý řádek, respektive buňku.

#### 3.11.1 Cyklus typu *while*

Cyklus *while* je takzvaný s podmínkou na začátku. Vyhodnocuje podmínku, pokud je vyhodnocena jako pravda, je proveden kód ve složených závorkách, pokud je vyhodnocena jako nepravda, je přeskočen. Jakmile je proveden kód uvnitř složených závorek, je znovu vyhodnocena podmínka a pokud je znovu pravda, je proveden znovu. Toto se pořád opakuje až do chvíle, kdy podmínka není pravdivá.

#### 3.11.2 Cyklus typu *do*

Druhým typem je cyklus s podmínkou na konci. Tento cyklus je podobný jako *while*, rozdíl je v tom, že se podmínka vyhodnocuje až na konci cyklu. To znamená, že máme jistotu, že se blok příkazů provede vždy alespoň jednou. Na rozdíl od předchozího, kde se může stát, že se neprovede ani jednou.

#### 3.11.3 Cykly typu *for*

Cyklus typu *for* je velmi flexibilní. Vypadá asi takto : `for( parametr1 ;parametr2 ; parametr3) {blok příkazů}`. *Parametr1* se vyhodnotí pouze jednou, a to na začátku, *parametr2* s vyhodnocuje vždy na začátku každého průchodu cyklem. A konečně *parametr3* se vyhodnotí na konci každého cyklu. Ukážeme si jednoduchý příkaz, který vypíše čísla od 0 do 99.

- `For ( $x=0 ; $x<100 ; $x++ )  
    { echo $x; }`

A jak to celé funguje? Před započítím cyklu se vyhodnotí *parametr1* v závorce za příkazem `for` (v našem příkladu je to výraz `$x=0`). Poté se po pořadě provádějí následující tři úkony:

- 1) Před započítím konkrétního cyklu se vyhodnotí *parametr2*, to je  $\$x < 100$ . Když neplatí, cyklus se ukončí. Jestliže platí, pokračujeme dále.
- 2) Proveďte se tělo smyčky. V našem případě se vypíše číslo, ale mohou se použít složené závorky a vykonat i více příkazů.
- 3) Po ukončení těla smyčky se provede *parametr3* (my v příkladu zvýšíme  $\$x$  o jedničku, na což existuje šikovný pomocník  $\$x++$ ) a vrátíme se k bodu 1.

Všechny parametry cyklu `for` jsou nepovinné, pokud je tedy nevypíšete, jsou implicitně chápány jako pravda. Při vynechání 2. parametru se tak z `for` stane nekonečný cyklus.

### 3.11.4 Cyklus *foreach*

Poslední cyklus *foreach* se od verze PHP4 neustále více používá, protože je jeho použití velmi jednoduché a zároveň je kód dobře čitelný. *Foreach* postupně zpracovává jednotlivé členy řady a poskytuje prvek v podobě proměnné *\$key* a jeho hodnotu ukládá do proměnné *\$value*. Příkaz *as* je nepovinný a v případě potřeby ho můžeme vynechat. Cyklus *foreach* umožňuje procházet prvky pole a zpracovávat je. Při první iteraci je automaticky nastaven vnitřní ukazatel na první element pole. Ukažme si to na jednoduchém příkladu.

```

○ $Cisla_Popisne = array (
 "Novak" => 563,
 "Dusek" => 198,
 "Kalab" => 226
);

Foreach($Cisla_Popisne as $Jmeno => $Cislo){
 Echo "$Jmeno má číslo popisné : $Cislo\n";
}

```



### 3.11.5 Break a continue

Příkaz *break* slouží k předčasnému ukončení cyklu. V následujícím příkladu budeme hledat náhodné číslo od 0 do 99 a zároveň toto číslo vypisovat. Jakmile hledané číslo najdeme tak cyklus ukončíme. Funkce *rand()* vrací náhodné číslo.

```
○ $Nahodne = rand(0,99);
 for($x=0 ; $x<100 ; $x++){
 echo "Hledám číslo $Nahodne : $x\n";
 if ($x==$Nahodne)
 break;
 }
```

## 4 Popis vytvořené aplikace

V předposlední kapitole teoretické části si popíšeme vytvořenou aplikaci, která si vzala za úkol vytvořit Kalendář akcí a rezervace techniky HZS ČR. Je to webová aplikace, která je určena pouze pro zaměstnance HZS ČR. Testována byla na územním odboru HZS v Ústí nad Orlicí. Požadavky na správný chod aplikace jsou dvě.

- Webový server s podporou PHP.
- Databázový systém MySQL

Obě tyto služby jsou dostupné na více operačních systémech a jsou k dispozici zdarma. Aplikace byla vyvíjena v prostředí Windows XP optimalizovaná na webový prohlížeč Opera s nainstalovaným balíkem XAMPP verze 1.7.2, který obsahuje webový server Apache, MySQL i PHP. V praktické části k tvorbě programu byly využity následující technologie:

- HTML
- CSS
- PHP
- MySQL

### 4.1 Návrh databáze

Databáze, o které se v této kapitole budeme bavit, se nazývá *bakalarka*. Návrh databáze probíhal podle zadaných vstupů, které se museli brát v úvahu. Základem této databáze je tabulka, pojmenována *prislusnik*. Jedná se o tabulku, ve které jsou ukládány data o příslušníku HZS ČR. Každý příslušník má své jedinečné osobní evidenční číslo “OEC“, dále se v tabulce ukládají data jako jméno, příjmení, věk a hodnost. Příslušník může mít maximálně jeden účet, pod kterým se přihlašuje do aplikace. Informace o tomto účtu jsou udržovány v tabulce *ucet* a kromě “OEC“ v tabulce najdeme ještě heslo a roli. Role můžou být tři. První z nich je obyčejný uživatel, který nemá skoro žádná práva. Druhým je strojník, jenž se stará o veškerou techniku na útvaru. Poslední je správce aplikace a jako jediný má neomezený pohyb a práva. Podrobnosti k právům si popíšeme později u popisování PHP aplikace. Vztah mezi tabulkami *prislusnik* a *ucet* je 1:1.

Každý kdo má přístup do aplikace, může zanechat vzkaz, který se ukládá do tabulky *komenrate*. Počet vzkazů není nijak omezen. Vztah mezi tabulkami *prislusnik* a *komentare* je 1:N.

Další důležitá tabulka se jmenuje *akce*. V této tabulce jsou všechny potřebné informace, které potřebujeme znát pro pořádání nějaké akce. Hlavní je název, musí být jedinečný a zároveň musí být vždy zadán. Dále potřebujeme znát místo, kde se akce bude konat, kdo jí bude pořádat a také co se tam bude dít. Jestli, jde o školení, sportovní akce, nebo nějaké porady. V neposlední řadě potřebujeme uchovávat v tabulce také datum, kdy se tato akce uskuteční. Vztah mezi tabulky je stejný jako v předchozím případě. Počet akcí také není omezen.

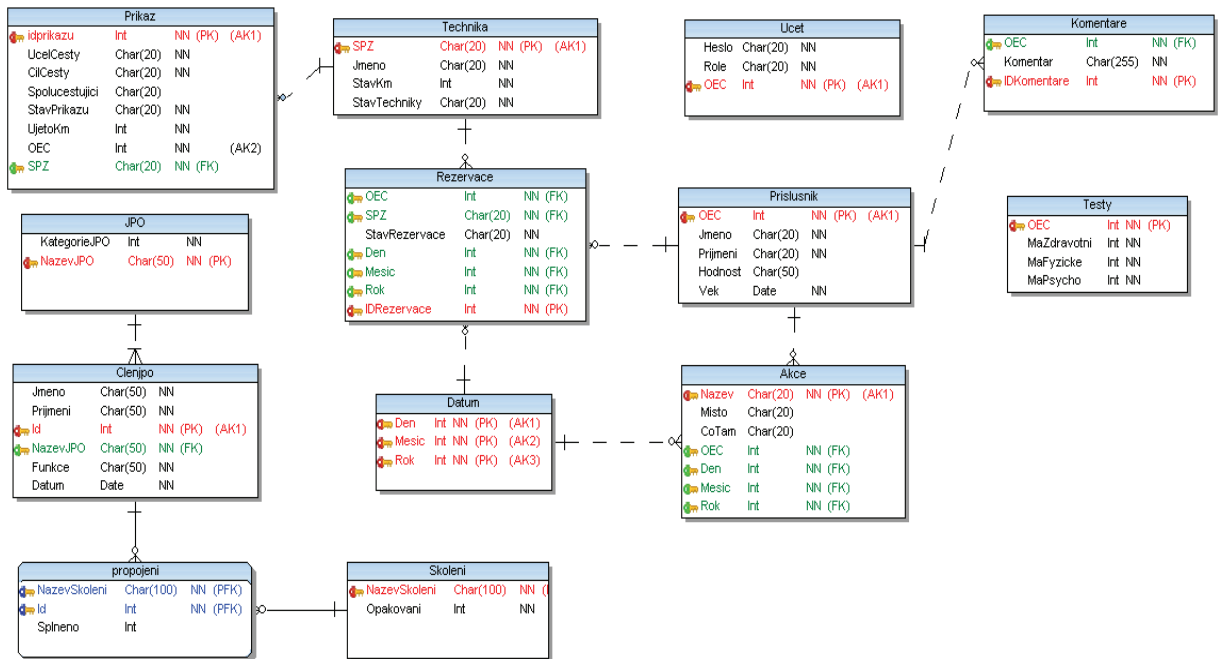
V tabulce *rezervace* potřebujeme především uchovávat informaci o jejím stavu. Máme dvě možnosti, buď je rezervace schválená, nebo neschválená. Další podstatná věc je *IDRezervace*, jedinečné číslo generováno automaticky při vložení pomocí omezení sloupce v MySQL AUTO\_INCREMENT. Jsou zde cizí klíče z ostatních tabulek, *OEC* z tabulky *prislusnik*, *SPZ* z tabulky *technika*. A také zde nesmí chybět datum, na kdy chceme techniku rezervovat. Příslušník může mít více rezervací, z toho vyplývá, že je vztah opět 1:N.

Tabulka, bez které bychom se nemohli obejít, se nazývá *technika*. Jsou zde uložena všechna potřebná data pro techniku HZS ČR. Hlavní je státní poznávací značka *SPZ*, je to zároveň i primární klíč. Dále zde můžeme najít jméno techniky. Jako je například 1CAS 24 Tatra815 4x4. Což znamená 1 – první výjezdové auto. CAS – cisternová automobilní stříkačka. 24 znamená výkon čerpadla (2400 litru za minutu). Tatra 815 pohon na všechny čtyři kola. V tabulce se také udržuje záznam o stavu tachometru a o stavu techniky, je-li schopná provozu, nebo je vyřazena z výjezdu. Technika, může být na více rezervacích, ale naopak na jedné rezervaci nemůže být více vozidel.

Na každou jízdu s technikou, ať už jde o ostrý výjezd k mimořádné události, nebo kondiční jízdy musí být vystaven příkaz k jízdě. Pro tento účel slouží tabulka *prikaz*. Je zde opět číslo příkazu, které je řešené, stejně jako v tabulce *rezervace*, pomocí omezení sloupce AUTO\_INCREMENT. Cizí klíč *SPZ* z tabulky *technika* určuje vozidlo, pro které je příkaz vystaven. Na příkazu dále musí být vypsáno, kam se pojedje, za jakým účelem a jestli je někdo jako spolucestující. Ve sloupci *UjetoKm* se ukládá hodnota

ujetých kilometrů po uzavření příkazu. Jestli je příkaz uzavřen poznáme podle sloupce *StavPrikazu*. Na jednom příkaze může být maximálně uvedena jen jedna technika, ale pro jednu techniku může být zadáno více příkazů najednou. To se stává například při živelných pohromách, kdy jedna cisterna přejíždí od zásahu k zásahu.

Výše popsané tabulky tvoří základní kostru celé aplikace. Jsou zde ale i další tabulky, které mají umožnit budoucí rozšíření aplikace.



obr 17 E-R diagram vytvořené databáze

#### 4.1.1 Aktualizace dat

Abychom mohli udržet aktuální data v databázi *bakalarka*, byly k tomuto účelu vytvořeny dva triggery, tedy programových jednotek, které se automaticky provedou v případě předem definované operace s daty. První trigger jsem pojmenoval *vymazprislusnika*. Má za úkol hlídat případ, když správce aplikace smaže příslušníka, aby po něm nezbyl v databázi nějaký nepořádek, jako třeba účet, rezervace, nebo příkazy k jízdě.

- CREATE TRIGGER *vymazprislusnika* AFTER DELETE ON *prislusnik*  
FOR EACH ROW  
BEGIN  
delete from *ucet* where *ucet.oec*=OLD.*oec*;  
delete from *rezervace* where *rezervace.oec*=OLD.*oec*;  
delete from *akce* where *akce.oec*=OLD.*oec*;  
delete from *komentare* where *komentare.oec*=OLD.*oec*;  
delete from *testy* where *testy.oec*=OLD.*oec*;  
delete from *prikaz* where *prikaz.oec*=OLD.*oec*;  
END;

Výše uvedený příklad reaguje na akci *DELETE*. Po té co se tato akce provede nad tabulkou *prislusnik*. Spustí se blok příkazu, mezi *BEGIN* a *END*, který vymaže vše co se vázalo k *OEC* odstraněného příslušníka.

Druhý trigger jsem nazval *vymaztechniku* a stará se o to, aby po odstranění techniky nezbyl žádný příkaz k jízdě ani žádná rezervace. Tento trigger se opět chová jako v předchozím případě, jediný rozdíl je že reaguje na tabulku *technika*.

- CREATE TRIGGER *vymaztechniku* AFTER DELETE ON *technika*  
FOR EACH ROW  
BEGIN  
delete from *prikaz* where *prikaz.spzt*=OLD.*spz*;  
delete from *rezervace* where *rezervace.spz*=OLD.*spz*;  
END;

Pro zobrazení všech triggerů v databázi se použije příkaz:

- Show triggers;

#### 4.1.2 Zajištění bezpečnosti

Pro zajištění bezpečnosti databáze a aplikace bylo využito funkce MySQL *mysql\_real\_escape\_string()*. Tato funkce opatří speciální znaky v neupraveném řetězci zpětným lomítkem pro bezpečné použití ve funkci *mysql\_query()* v závislosti na

aktuální znakové sadě spojení. To zabrání útokům, které jsou známé jako SQL Injection.<sup>3</sup>

## 4.2 Aplikace z pohledu uživatelů

V této kapitole si popíšeme možnosti vytvořené aplikace. Jelikož je aplikace určena jen pro zaměstnance HZS ČR, proto chybí na přihlašovací stránce možnost jakékoli registrace. Přidávat nové uživatele můžeme, jen pokud máme oprávnění správce, kde si vybíráme ze tří základních rolí.

První role je obyčejný uživatel, tedy *user*. Příslušník, který dostal přidělené toto oprávnění, se po přihlášení se dostane na hlavní stránku programu, která je rozdělená do čtyř částí jak je vidět na obrázku. V levé horní části je umístěn kalendář, kde máme možnost zadat datum v rozmezí let 2000 až 2050. Zadání data je ošetřeno proti nepovoleným znakům, jako je například písmeno, nebo datum 3.13.2010.

```
if (is_numeric($_POST['mesic']) and is_numeric($_POST['den']) and is_numeric($_POST['rok']))
{
 if((2000 < $_POST['rok']) and ($_POST['rok'] < 2050))
 {
 if (checkdate($_POST['mesic'], $_POST['den'], $_POST['rok']))
```

### obr 18 Ošetření data

Výpis akcí je zobrazován v levém dolním rohu. Rezervace techniky najdeme na obrazovce v pravém dolním rohu. Pravý horní roh je určen pro menu, kde můžeme najít pět sekcí a tlačítko pro odhlášení ze systému.

---

<sup>3</sup> `mysql_real_escape_string()` nepřidává zpětná lomítka před `%` and `_`.

The screenshot shows the main interface of an application. It is divided into four main sections:

- Calendar:** A calendar for May 2010 (květen 2010) with days of the week (Po, Út, St, Čt, Pa, So, Ne) and dates (1-31).
- Date Input:** A section titled "Zadané datum je v pořádku 14.5.2010" with input fields for "Den:", "Mesic:", and "Rok:", and a "zpracuj" button.
- Menu:** A section titled "Menu" with links: "Evidence JPO", "Evidence Uživatelů", "Zjednodušená evidence jízd", "Evidence akcí", and "Forum". There is an "Odhlásit" button.
- Akce:** A table with columns "Název", "Místo", and "Účel". It contains one entry: "Orlická věž HZS Ústí nad Orlicí Požární sport".
- Technika:** A table with columns "Rezervoval", "Jméno techniky", "SPZ", "Stav", "Den", "Mesic", and "Rok". It contains three entries:
 

| Rezervoval | Jméno techniky                  | SPZ      | Stav        | Den | Mesic | Rok  |
|------------|---------------------------------|----------|-------------|-----|-------|------|
| 793599     | OA - Š Octavia                  | 3E5 4876 | Neschválená | 20  | 5     | 2010 |
| 793599     | 3CAS 32/8200/800 T-815 UO 74-48 |          | Neschválená | 10  | 5     | 2010 |
| 793444     | OA - Š Fabia 1,4                | 1E1 1908 | Neschválená | 12  | 5     | 2010 |
| 793444     | AZ 30 - Camiva                  | 2E5 5262 | Neschválená | 14  | 5     | 2010 |

obr 19 Hlavní stránka aplikace

První odkaz v menu se jmenuje Evidence JPO, slouží pro správu jednotek požární ochrany. V této sekci může přihlášený uživatel vyhledávat jednotlivé členy, nebo jednotky.

Pro obyčejného uživatele je druhý odkaz jen informativní, najdeme zde výpis všech uživatelů, kteří mají přístup do aplikace.

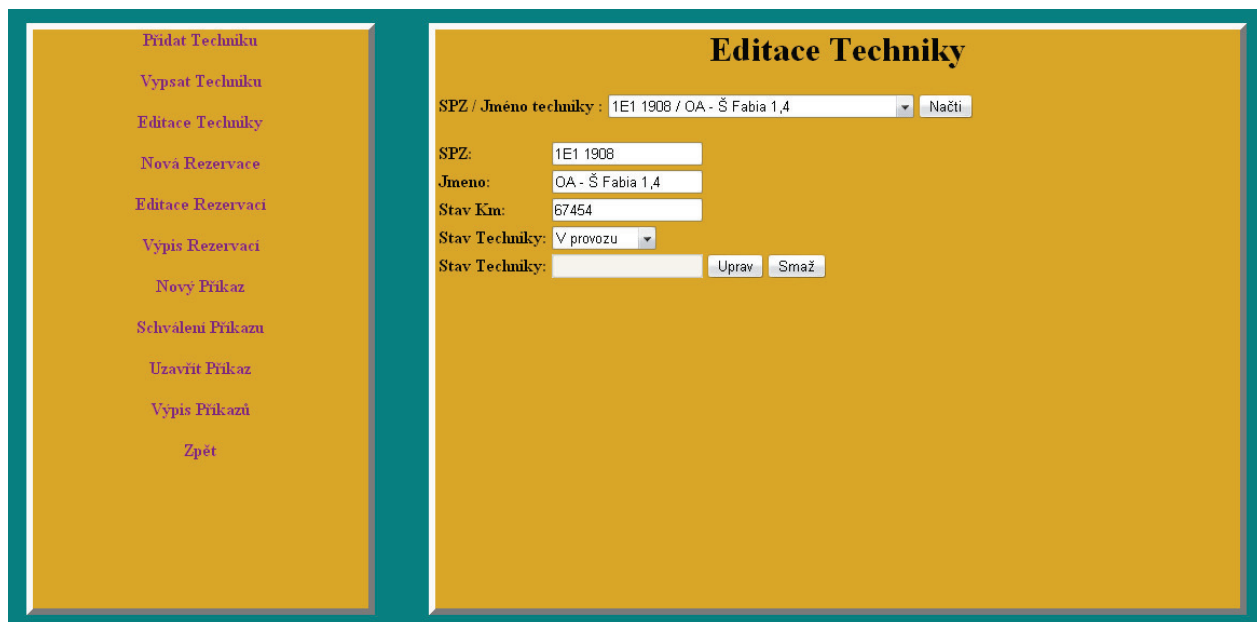
Zjednodušená evidence jízd je jedná z hlavních sekcí. Jako první tu nalezneme výpis veškeré techniky, se kterou můžeme dále pracovat. Příslušník smí požádat o rezervaci techniky na určité datum. Po odeslání rezervace musí počkat, jestli bude schválená a to může provést buď správce aplikace, nebo strojník. Své rezervace máme možnost sledovat v sekci výpis rezervací, zde můžeme vidět, zda byla schválena. K tomu abychom si mohli vypůjčit techniku a vyrazit na místo určení, je zapotřebí mít vystaven příkaz k jízdě. Jaký je rozdíl mezi rezervací a příkazem? Rezervace je pouze informace pro vedoucí, kde je uvedeno datum a technika o kterou příslušník žádá. Naopak příkaz je vystaven na konkrétní vůz, se kterým se má jet. Po ukončení jízdy se musí každý příkaz uzavřít. To znamená zadat počet ujetých kilometrů, který je připočten k aktuálnímu stavu tachometrů. Uzavírat příkaz má právo pouze strojník a správce aplikace. Příkazy můžeme sledovat v sekci výpis příkazů.

Jediné co uživatel může zadat je nová akce. To uděláme, tak že si v menu najedeme do sekce evidence akcí a zvolíme nová akce. Nemůžeme jí ovšem editovat, na to nemáme

oprávnění. Nalezneme zde navíc výpis všech pořádaných událostí. V sekci fórum můžeme zanechat vzkazy pro ostatní.

Vedoucího strojní služby má za úkol udržovat v chodu veškerou techniku na útvary HZS ČR a to ať už jde o technický stav, nebo o administrativní část, kde například musí sedět stavy tachometrů na vozidlech. Role pro vedoucího strojní služby se jmenuje *strojník*, má stejné práva jako obyčejný uživatel. Navíc má právo do databáze přidávat novou techniku. Část nazvaná editace techniky, slouží pro případ, kdy je zapotřebí sundat techniku z výjezdu, nastavit její stav na mimo provoz. V této části dále můžeme měnit stav ujetých kilometrů, jméno, nebo spz techniky. Krom zadávání nových příkazů, strojník schvaluje a uzavírá již vytvořené příkazy. Podobné je to u rezervací, zde je pouze schvaluje. Další rozdíl najdeme v tom, že může nejen založit novou akci, ale zároveň již zadané akce editovat.

Správce aplikace navazuje právy na vedoucího strojní služby s tím rozdílem, že v sekci evidence uživatel můžeme přidávat nové a editovat již zadané uživatele. Dále můžeme vymazat příspěvky ve fóru.



obr 20 Editace techniky

#### 4.2.1 CSS

CSS vzniklo někdy kolem roku 1997. Je to kolekce metod pro grafickou úpravu webových stránek. Zkratka znamená Cascading Style Sheets, česky "kaskádové styly". Kaskádové, protože se na sebe mohou vrstvit definice stylu, které od sebe dědí.



Kaskádové styly jsem použil při tvorbě aplikace, díky tomu není problém změnit vzhled nebo velikost jednotlivých částí.

```
.hlavni{ background-color:teal;
 border-style:outset;
 width:20%;
 height:auto;
 position:absolute;
 left:0%;
 top:120px;
}
```

obr 21 Ukázka CSS

## 5 Závěr

Cílem této bakalářské práce, bylo vytvořit funkční webovou aplikaci kalendář akcí a rezervace techniky HZS ČR a nasazení do reálného provozu. Tato aplikace byla testována na územním odboru HZS v Ústí nad Orlicí, po dobu čtrnácti dnů od 19.4.2010 do 2.5.2010.

Z testování aplikace vyplynula tato opatření. Ve vytvořené aplikaci bude změněno zadávání datumů do kalendáře s využitím rolovacích seznamů. Výpis rezervací na hlavní stránce bude změněn jen na určitý den, jako je to u výpisů akcí. Co se týče funkčnosti aplikace, nenastal žádný problém.

Jelikož, hlavním úkolem této práce byla funkčnost, byl zvolen jednoduchý vzhled pro jednodušší orientaci a zadávání nových dat.

Po testování vytvořené aplikace byla navržena tato rozšíření. Bude vytvořena nová sekce evidence kurzů, která bude mít za úkol hlídat splnění jednotlivých kurzů u členů jednotek požární ochrany. Dále bude upozorňovat měsíc dopředu na obnovení procházejících osvědčení. Jako další rozšíření byl navržen docházkový systém a program na plnění fyzických testů.

## 5.1 Použité zdroje

- [1] GUTMANS, Andi; BAKKEN, Stig Sather; RETHANS, Derick. Mistrovství v PHP 5. 2. vydání. Praha : Computer Press, 2007. 656 s. ISBN 9788025115190.
- [2] WELLING, Luke; THOMSON, Laura. MySQL Průvodce základy databázového systému. Vyd. 1. Brno : CP Books, 2005. 255 s. ISBN 80-251-0671-3.
- [3] Kolektiv autorů. PHP6, MySQL, Apache : Kolektiv autorů. Praha : Computer Press, 2006. 816 s. ISBN 978-80-251-2767-4.
- [4] *MySQL* : *Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2010-04-13]. Dostupný z WWW: <<http://cs.wikipedia.org/wiki/MySQL>>.
- [5] CHURÝ, Lukáš. *Programujte* [online]. 207 [cit. 2010-05-05]. Teoretický úvod do relačních databází. Dostupné z WWW: <<http://programujte.com/>>.
- [6] GRIMMICH, Šimon. *Tvorba-webu.cz* [online]. 2003 [cit. 2010-05-05]. PHP - Základy a text. Dostupné z WWW: <<http://www.tvorba-webu.cz/php/text.php>>.