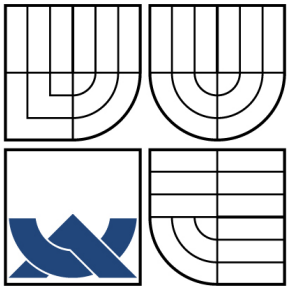
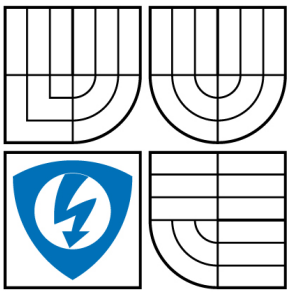


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

WORKFLOW S PODPOROU KONTROLY PROCESŮ

WORKFLOW WITH PROCESS CONTROL SUPPORT

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

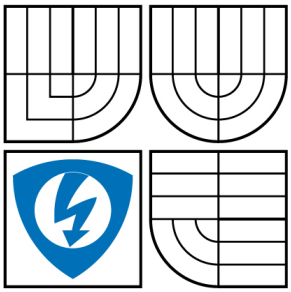
AUTOR PRÁCE — MICHAL BABIČKA

AUTHOR

VEDOUCÍ PRÁCE — Ing. RADOVAN HOLEK, CSc.

SUPERVISOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Babička Michal	***ID:***	74557
Ročník:	3	***Akademický rok:***	2007/2008

NÁZEV TÉMATU:

Workflow s podporou kontroly procesů

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte podporu pro kontrolu procesů v informačních systémech, založených na workflow. Předpokládá se, že informační systém je realizován nad databázovým strojem MySQL prostřednictvím PHP, nebo nad databázovým strojem ORACLE prostřednictvím PLSQL.
Cílem práce je navrhnou systemové řešení umožnující automatickou kontrolu jednotlivých procesů řízených životními cykly entit definovaných v IS.

DOPORUČENÁ LITERATURA:

Termín zadání:	1.2.2008	***Termín odevzdání:***	2.6.2008

Vedoucí práce: Ing. Radovan Holek, CSc.

prof. Ing. Pavel Jura, CSc.
předseda oborové rady

UPOZORNĚNÍ:

Vysoké učení technické
Fakulta elektrotechniky a komunikačních technologií
Ústav automatizace a měřící techniky

Workflow s podporou kontroly procesů

Bakalářská práce

Abstrakt:

Cílem této práce je nalézt systémové řešení pro kontrolu procesů v informačních systémech, založených na workflow. Řešení je hledáno pro informační systém založený na databázi MySQL a programovacím jazyku PHP. Nejprve si definujeme pojmy jako je pracovní postup, proces a wokflow. Následně se zaměříme na možnosti, která máme pro kontrolu procesů nad databází MySQL. Těmito možnostmi je kontrola uživatelského formuláře, dále uložené procedury a triggery a konečně možnost volání kontrolních funkcí.

Klíčová slova:

kontrola, proces, workflow, informační systém, mysql, php, životní, cyklus, automaticky

Brno University of Technology
Faculty of Electrical Engineering and Communication
Department of Control and Instrumentation

Workflow with process control support

Bachelor's Thesis

Abstract

The aim of this thesis is to find a system solution for process control within the information systems on the basis of workflow. The solution is needed for an information system based on MySQL and PHP programming language. At first we must define notions as labor progress, process and workflow. Then we locate for possibilities which we have for process control upon MySQL database. Possibilities are users form control, saved procedures, triggers and at last calling control functions.

Key words

control, process, workflow, informational system, mysql, php, life cycle, automatically

Bibliografická citace

BABIČKA, Michal. *Workflow s podporou kontroly procesů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. s.61, příloh 1. Vedoucí práce Ing. Radovan Holek, CSc.

P r o h l á š e n í

„Prohlašuji, že svou bakalářskou práci na téma "Workflow s podporou kontroly procesů" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne : Podpis:

P o d ě k o v á n í

Děkuji tímto Ing. Radovanu Holkovi, CSc. za cenné připomínky a rady při vypracování bakalářské práce.

V Brně dne : Podpis:

OBSAH:

SEZNAM OBRÁZKŮ

Obrázek 6.25 – Výpis tabulky _report

Obrázek 6.26 – Výpis tabulky _kontrola

Obrázek 6.27 – Výpis tabulky _report

1. ÚVOD

Téměř v každé lidské činnosti se setkáme s nějakým předem naplánovaným pracovním postupem a u každého postupu nás nezajímá jen výsledek práce, ale potřebujeme mít i přehled o jeho průběhu. Nejinak je tomu i v případě zpracovávání dat nějakou aplikací nebo informačním systémem. Je tedy nutné mít nějaký nástroj, který nám takovou kontrolu umožní. V našem případě máme informační systém založený nad databází MySQL a programovacím jazykem PHP, podívejme se tedy jaká možná řešení kontroly procesů máme k dispozici a jakými nástroji lze životní cyklus sledovat.

2. PRACOVNÍ POSTUP

Pracovní postup je základem každé činnosti, na jejímž konci má být nějaký výrobek, informace nebo data. Na správné volbě pracovního postupu závisí doba i kvalita práce. Popis pracovního postupu se vyznačuje přesným a výstižným vyjadřováním a přehledností. K přesnému vyjadřování přispívají především odborné názvy (termíny). Jednotlivé činnosti (fáze postupu) jsou zachyceny v přesné časové posloupnosti. Ke každému pracovnímu postupu lze přistupovat ze dvou hledisek:

2.1 FUNKČNÍ PŘÍSTUP

- jediný zodpovědný vedoucí
- zajištění podmínek
- přehled o využití lidí
- ověření a pochopení zadání
- chybí záznam o zadání úkolu
- obvykle nedůsledné sledování plnění úkolu
- lidé pracují pro své vedoucí
- rutinní činnosti se nezadávají
- procesní chyby se nahrazují úsilím

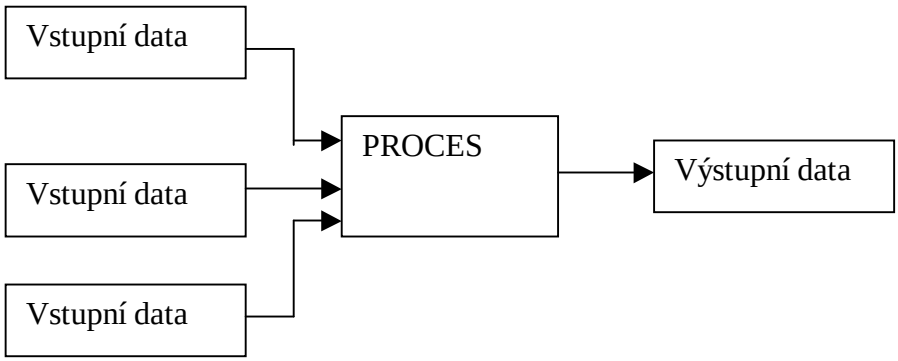
2.2 PROCESNÍ PŘÍSTUP

- spouštěcí událost
- nalezení nejvhodnějšího zdroje probíhá dle pravidel
- samostatná práce, předání na navazující činnost
- předání musí být evidováno
- koordinátor
- stálý přehled o řešení

Nás bude zajímat procesní přístup a proto je třeba si definovat co je to proces.

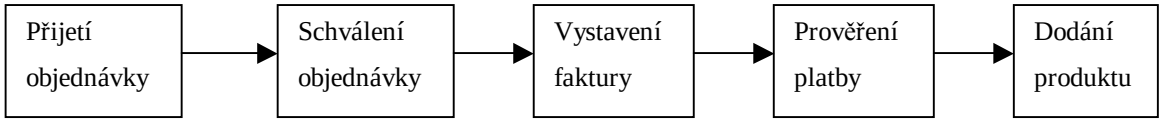
3. PROCES

Proces je série kroků určených k vytvoření produktu nebo poskytnutí služby. Požadavky na kvalitní proces je jeho automatizace a detailní možnosti sledování a také jeho efektivnost. Cílem je na základě vstupních dat získat v co nejkratším čase a s co nejmenšími náklady požadovaná výstupní data.



Obrázek 3.1 - Obecný model procesu

Jako konkrétní příklad můžeme uvést podnikový proces kterým může být zpracování objednávek a výdej zboží.



Obrázek 3.2 – Podnikový proces

Procesy jsou nastaveny podle konkrétní činnosti kterou mají vykonávat, ale měly by se řídit pravidly pro přehlednost a jednoduchou modifikaci. Tzn. měly by být rozděleny do více tabulek a odpovídat normalizovanému datovému modelu na základě normalizace celého procesu (primární klíče, každý neklíčový atribut musí být závislý na primárním klíči, netranzitivní závislosti, odstranění nejednoznačností, duplicity v datech apod.)

3.1 CO NÁS ZAJÍMÁ U PROCESŮ

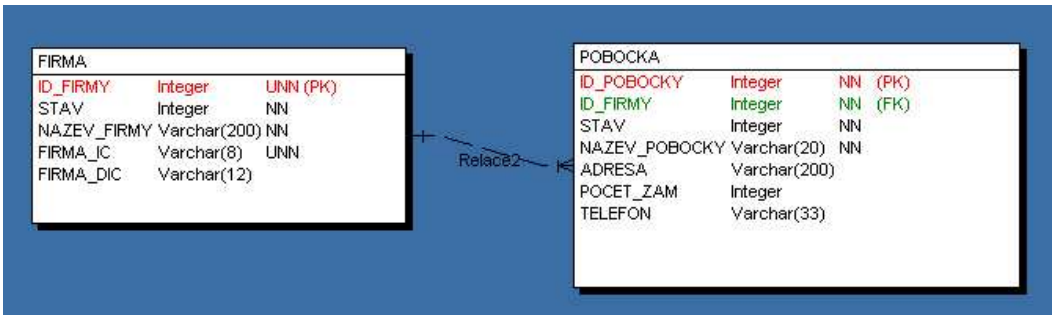
Abychom se mohli snažit nějaký proces řídit musíme se nejprve postarat o jeho zmapování a dokumentaci. Tato dokumentace by měla zahrnovat všechny důležité atributy a vazby na ostatní procesy. Dále je třeba provést analýzu a pokusit se odstranit to co není trvale využito, nebo by bylo příliš nákladné v porovnání s důležitostí.

U vazeb mezi jednotlivými procesy je důležité kromě statických vlastností jednotlivých procesů nezapomínat také na jejich vzájemné dynamické vazby. Procesy je nutné po analýze zpětně ověřit z ohledem na konkrétní podmínky ve firmě, aby nenastala situace, kdy je proces sice implementován správně, ale nedá se podle něj pracovat.

Také je třeba analyzovat výjimky a nastavit pravidla, kterými mají být řešeny v případě podpory procesu systémem a dále je třeba si uvědomit že každá změna v procesu musí být zaznamenána do systému z důvodu statistického pohledu kdy je třeba posoudit frekvenci výskytu změn a navrhnout takový systém který bude schopen na tyto změny včas reagovat.

3.2 DATOVÝ MODEL (ERD DIAGRAM)

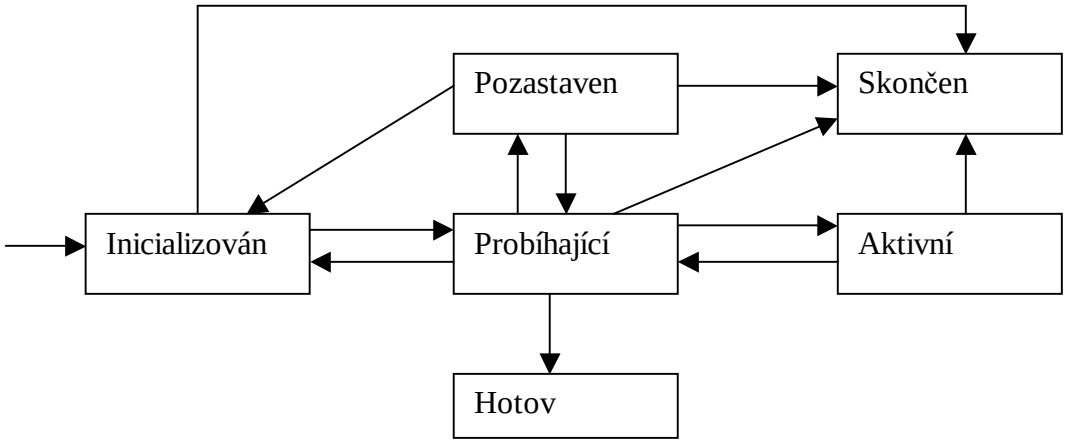
Hledáme normalizovaný datový model na základě dostupných informací o aplikaci. Musíme zjistit jaká data budeme zpracovávat, jak mají být uchovávána a jaké bude třeba připravit výstupy, dále je nutné brát na zřetel, že data musí být provázána s ostatními procesy. Hledáme primární klíče jednotlivé vazby mezi tabulkami a co nejjednodušší strukturu s ohledem na možnou další modifikaci.



Obrázek 3.3 – ERD diagram

3.3 ŽIVOTNÍ CYKLUS PROCESU

Každý proces se řídí procesním cyklem, který začíná vznikem a končí předáním nebo zánikem. Přesná analýza životního cyklu je nejdůležitější pro správnou funkci a chování celé aplikace (informačního systému). Je třeba přesně definovat a zajistit každou změnu stavu ke které dojde v průběhu procesu.



Obrázek 3.4 - Obecný model stavů výskytu procesu

Stavy procesu v životním cyklu mohou být např. tyto:

- Inicializován

  Proces je vytvořen buď příkazem uživatele nebo na žádost operačního systému o provedení služby či na žádost jiného procesu.
- Pozastaven

  Stav pozastaven vyjadřuje, že proces čeká na určitou událost, např. dokončení I/O operace.
- Probíhající

  Stav probíhající charakterizuje proces připravený k vykonání a čeká pouze na přidělení procesoru.
- Aktivní

  Ve stavu běžící je procesu přidělen procesor a právě se provádí příslušné programy.

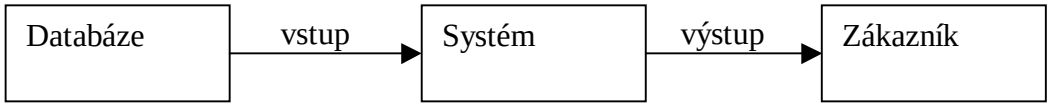
- Hotov

  U procesu byly vykonány všechny činnosti a jeho výskyt byl odstraněn
- Skončen

  Proces byl skončen dříve než došlo k jeho kompletnímu zpracování a byly odstraněny všechny vazby k němu plynoucí

3.4 DIAGRAM DATOVÝCH TOKŮ (DATA FLOW DIAGRAM)

Jedná se o grafickou reprezentaci “toku” dat v informačním systému. DFD může být používán k vizualizaci procesu. Znázorňuje odkud kam směřuje tok dat a kdo do nich může zasahovat.



Obrázek 3.5 – Data Flow Diagram

4. WORKFLOW

Workflow (doslovný překlad znamená pracovní tok) lze přiblížit jako tok informací v podnikovém procesu a jejich automatizované řízení. Efektivnějším řízením těchto procesů lze redukovat jejich náklady, zkrátit životní cyklus, zrychlit realizaci technologických změn, zlepšit zákaznický servis.

Pojem workflow se používá v mnoha významech, od vlastního procesu až po počítačové systémy, které zajišťují jeho automatizaci (např. Helios, http://www.helios.eu/, SAP http://www.sap.com/, Microsoft Dynamic http://www.microsoft.com/dynamics/, K2 http://www.k2atmitec.cz/, ABRA http://www.abra.eu/ a další)

4.1 SYSTÉM ŘÍZENÍ WORKFLOW

Systém řízení workflow definuje, vytváří a řídí průběh procesu. Je schopen interpretovat definici procesu, komunikovat s účastníky workflow a v případě potřeby spustit další aplikace.

Systém řízení workflow zajišťuje automatizaci podnikového procesu řízením posloupnosti pracovních činností a vyvoláváním odpovídajících lidských nebo technických zdrojů. Poskytuje administrativní a monitorovací funkce, jako je např. zrušení procesu, změna účastníka procesu, kontrola stavu procesu apod.

4.2 WORKFLOW SYSTÉM

Workflow systém propojuje principy, metodiky a technologie z různých oblastí informatiky a řízení. Workflow systémy obvykle pokrývají nejenom fázi realizační (řízení průběhu procesu) , ale i fázi přípravnou (definici) procesu a také fázi sledovací a vyhodnocovací.

Za skutečný workflow systém je považován ten, který poskytuje:

- grafický návrh workflow, vytvoření procesního modelu toku činností a úkolů od jejich vzniku po jejich ukončení
- role, tzn. možnost přiřadit jednotlivým činnostem role nebo pracovní funkce, aby workflow nebylo vázáno na konkrétního pracovníka, ale danou roli
- pravidla, vložit do definice workflow logiku procesu bez potřeby programování
- možnosti řešit vyjímečné situace
- monitoring, sledovat jednotlivé výskyty procesů
- měřitelnost, schopnost generovat statistické zprávy, jako podkladu pro zjištění časového průběhu procesu a jeho nákladů
- simulace, možnost testovat procesy workflow na jednom počítači před jeho spuštěním v síti
- aktivita, workflow musí uživatele informovat o nových úkolech, upozorňovat na termíny a případně směrovat úkoly na jiné uživatele
- databázové rozhraní, řada worklow procesů buď využívá informaci z databázi, aby uživatel mohl učinit patřičné rozhodnutí poskytovat kvalitní databázové rozhraní

4.3 TYPY WORKFLOW SYSTÉMŮ

4.3.1 Administrativní workflow

- jedná se o proces jednoduchý a dobře strukturovaný, obvykle spojený s formuláři nebo jinými dokumenty (žádost o služební cestu)

4.3.2 Ad hoc workflow

- je založeno na náhodnosti vzniku procesu, cílem jsou nulové náklady a žádná správa (např. domluvení schůzky za účelem upřesnění požadavků klienta a zainteresovaných stran)

4.3.3 Kolaborativní workflow

- podporuje především týmovou spolupráci, jejíž výsledkem je např. nějaký dokument, rozhodnutí apod. (příprava reklamy pro nový produkt)

4.3.4 Produkční workflow

- cílem je vysoká produktivita, co nejkratší doba mezi jednotlivými stavy (kroky) procesu, vyžadují integraci s dalšími aplikacemi (např. vyřízení příspěvku na dovolenou).

4.4 VÝHODY WORKFLOW A CO LZE OČEKÁVAT:

- zavedení standardních postupů zvyšuje efektivitu práce a snižuje náklady
- přispívá ke zjednodušení podnikových procesů, zlepšuje organizaci a kvalitu práce
- pracovní postupy jsou uchovány v systému, nikoloiv v hlavách odcházejících pracovníků
- noví pracovníci se mohou snáze zapracovat
- na základě vyhodnocení zdokumentovaných pracovních postupů je možné lépe navrhovat změny
- v každém okamžiku lze zjistit stav konkrétního případu
- vyřizování případů se značně urychluje
- veškeré změny v kolujících dokumentech či datech jsou autorizovány
- průběh každého případu je zachycen v historii, kterou nelze dodatečně měnit
- manažeři získávají věrohodnější podklady pro hodnocení pracovníků
- dokumenty a aplikace jsou integrovány
- je podporováno řízení kvality

5. PHP A MYSQL

Jedná se o velmi rozšířenou kombinaci nástrojů pro vytváření HTML stránek, především tam kde potřebujeme dynamicky měnit obsah (redakční systémy), jednoduchých aplikací i komplexních informačních systémů.

5.1 PROGRAMOVACÍ JAZYK PHP

PHP je v současnosti velmi rozšířená technologie umožňující snadné programování na straně serveru. Toho lze využít k tvorbě různých interaktivních webových stránek. Stručně lze říci, že skript napsaný v PHP je proveden na serveru podle zadaných kritérií a výsledek je odeslán volajícímu počítači stejným způsobem, jakým se odesílají běžné statické (XHTML) stránky. Jakmile je však stránka načtena klientem, pomocí PHP ji již není možné dále měnit.

PHP je programovací jazyk umožňující procedurální i objektově orientované programování. Znalost objektově orientovaného programování (OOP) tedy může být při práci v PHP výhodou, není však nutnou podmínkou. PHP také patří mezi jazyky, kde například není nutné předem definovat typ proměnných, navíc jakákoli proměnná může kdykoli změnit svůj typ. Jednoduše řečeno, co se týče psaní kódu, z PHP při psaní skriptů "sálá" určitá volnost a neomezenost. Na druhé straně záleží plně na programátorovi, jaký si bude v kódu udržovat pořádek.

5.2 DATABÁZE MYSQL

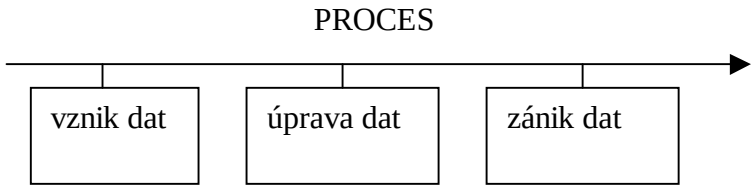
Do MySQL (My Structured Query Language = systém pro řízení databází) lze ukládat různá data (texty, obrázky atd.), s nimiž lze dále jednoduše pracovat (třídit, řadit, filtrovat apod.). Nejčastěji se MySQL používá ve spojení s jazykem PHP, které umožňuje přístup k uloženým datům.

Každá databáze v MySQL obsahuje tabulky, každá tabulka má sloupce a řádky – v každém řádku jsou záznamy předem určeného typu.

Databáze MySQL je jeden z prvních hojně rozšířených systémů. Práce s tímto systémem se dá využít v C, C++, Java, Perl, PHP, Python, Tcl, Visual Basic nebo .NET.

6. KONTROLA PROCESŮ VE WORKFLOW

Během každého procesu dochází ke změně stavu jednotlivých entit v závislosti na životním cyklu. Proto je nutné sledovat každou změnu stavu. Každá data vznikají na základě určitých přesně definovaných vstupních podmínek (tzn. musíme přesně stanovit, které informace jsou pro nás důležité, je nutné pamatovat na zákon o ochraně osobních údajů apod.). Obecně se data nacházejí ve třech úrovních – vznikají, jsou upravována, zanikají.



Obrázek 6.1 – Vznik a zánik dat v procesu

6.1 KONTROLA NA STRANĚ UŽIVATELE

6.1.1 Kontrola uživatelského formuláře

Kontrola na straně uživatele probíhá především pomocí JavaScriptu. Typickým příkladem může být například kontrola formulářů před odesláním, zda jsou vyplněny všechny důležité položky, zda je dodržen správný formát a zda nejsou vkládány nepovolené znaky.

Použití kontrolovaných formulářů je výhodné většinou u jednoduchých aplikací, tzv. “ pořizovaček dat“, např. různých dotazníků apod. Nehodí se však jako systémové řešení pro rozsáhlejší aplikace.

Výhody:

- kontrola na straně uživatele, přijímáme pouze ověřená data
- efektivnější u jednoduchých aplikací
- data se odesílají až po ověření a zbytečně aplikace nezatěžuje server

Nevýhody:

- nutná znalost JavaScriptu
- nárůst velikosti stránky
- komplikovaná možnost kontroly dat, data ještě nejsou uložena v databázi, ale jsou stále na straně uživatele
- zakázaný JavaScript v prohlížeči na straně uživatele způsobí, že data odeslaná na server nemusí být validní a je tedy potřeba provádět kontrolu na straně serveru

6.2 KONTROLA NA STRANĚ SERVERU

Řešení spočívá v kontrole procesu, při jeho změně stavu, např. při ukládání nebo editaci dat, při změně v životním cyklu apod.

6.2.1 Uložené procedury

Jednou z možností jak procesy kontrolovat jsou uložené procedury přímo v databázi MySQL. Uložené procedury jsou jednou z věcí, kterou se liší "malé" databáze od "velkých". Od verze MySQL 5.0 jsou uložené procedury normálně ve výbavě tohoto databázového systému. Uložená procedura je sada příkazů SQL, které jsou:

- uložené na serveru
- zkompilované pro rychlejší použití

Jedním důvodem pro použití procedur je fakt, že některé databázové operace se neustále opakují. Pokud je taková databázová operace provedena jako dávka na serveru, nemusejí se jednotlivé příkazy vypisovat pokaždé, když je chceme provést.

Dalším důvodem pro zavedení procedur je pokud chce několik databázových klientů vykonávat stejnou práci. Uložené procedury rovněž mohou podstatným způsobem ulehčovat správu databázových aplikací.

Jestliže je potřeba uloženou proceduru upravit , může se to udělat na jednom místě a tato změna je ihned použitelná pro všechny aplikace, které s databází komunikují.

Dalším významným rysem uložených procedur je to, že přispívají k zabezpečení serveru a to zejména ze dvou důvodů:

- Spouštění procedur může být omezeno jen na konkrétního uživatele
- Procedury mohou samy kontrolovat parametry (velikost, typ, počet), které jim jsou předávány

Nejdůležitějším argumentem pro nasazení uložených procedur na server je však to, že jsou obecně schopny běžet rychleji, než kdyby se příslušný kód vykonával příkaz po příkazu z klientské aplikace. Důvodem je fakt, že uložené procedury jsou na serveru zkompilovány.

Mezi nejčastější činnosti, které uložené procedury vykonávají, patří zejména:

- vybírání dat
- vkládání, aktualizace, odstraňování dat
- vytváření, používání a rušení dočasných tabulek
- matematické a statistické výpočty

Uložené procedury se vytvářejí příkazem **create procedure**.

Příklad použití procedury

Vytvoříme např. tabulku data:

```
CREATE TABLE data (id int, jmeno varchar(50));
```

s následující sadou dat:

```
INSERT INTO data (id, jmeno) values (1, 'Petr');
INSERT INTO data (id, jmeno) values (2, 'Pavel');
INSERT INTO data (id, jmeno) values (3, 'Jan');
```

Proceduru, která vrátí všechny záznamy z této tabulky můžeme vytvořit pomocí následujícího příkazu, který spustíme přímo SQL příkazového okna v databázi:

```
        create procedure sp_VratData()
begin
  select * from data;
end
```

A zavoláme ji pomocí příkazu:

call sp_VratData()

Tělo procedury je uloženo mezi klíčovými slovy BEGIN a END

Samozřejmě, že uložené procedury nemusejí obsahovat jen sadu příkazů, které se budou sekvenčně provádět od jednoho k druhému. Mohou obsahovat určité příkazy pro větvení kódu, jako jsou smyčky nebo podmínky. To dnes prozkoumáme; takové věci totiž dělají z uložených procedur docela užitečné pomocníky. Alespoň v některých situacích.

V procedurách lze také provádět větvení kódu jako v programovacím jazyku PHP.

Příklad použití procedury:

```
create procedure sp_showlog (jennejnovejsich5 boolean)
begin
 if jennejnovejsich5=1 then
  select * from log order by datum desc limit 5;
 else
  select * from log order by datum desc;
 end if;
end
```

Výhody použití:

- Zjednodušení kódu aplikace.

Nevýhody:

- Znalost syntaxe SQL procedur konkrétní databáze.
- Kompatibilita mezi verzemi databáze.

6.2.2 Triggery

Další možností jak kontrolovat data při přechodu procesu v životním cyklu přímo v databázi MySQL jsou triggery. Opět ale lze stejně jako procedury využívat triggery až od verze 5 databázového stroje MySQL.

Trigger je uložená procedura, která se spouští v souvislosti s provedením nějakého akčního dotazu na tabulce. A jeho vlastnosti jsou zejména:

- Trigger je "uložená procedura", to znamená, že uvnitř triggeru lze provádět většinu věcí, které umí uložené procedury. V triggeru lze mít například smyčku, podmínku, lokální proměnnou, matematický výpočet a podobně.
- Spouští se "v souvislosti" s akčním dotazem. To v praxi znamená, že se trigger může spustit buď předtím, než je úprava dat provedena, nebo poté,

co jsou změny v datech zapsány do databáze. Některé databáze umožňují pouze triggery před uložením dat, ale většina (včetně MySQL) má k dispozici oba typy.

- "Akční dotaz" znamená, že trigger lze spustit při vkládání dat, při jejich aktualizaci nebo při odstraňování dat z databáze. Rovněž z toho vyplývá, že trigger nelze v žádném případě spustit příkazem SELECT.
- "na tabulce" - každý trigger patří právě jedné tabulce. Ačkoli samozřejmě může existovat tabulka bez triggeru, není možné, aby existoval trigger nezávisle na tabulce. V některých DBMS (Database Management System) může být více triggerů stejného typu na jedné tabulce a lze dokonce určit pořadí, v němž budou spuštěny. V MySQL však některé z těchto věcí nejsou možné.

Triggery mají sice stejnou syntaxi jako uložené procedury, ale liší se především v tom, že triggerům není možné předávat žádné vstupní parametry. Triggery navíc narozdíl od uložených procedur nemohou vracet sadu záznamů a databázové systémy mívají některá omezení, které se v triggerech nesmějí objevit. Pro MySQL platí, že se v triggeru nesmí objevit přinejmenším tyto příkazy:

- Příkazy ALTER (ALTER TABLE, ALTER DATABASE a tak dále)
- Příkazy pro řízení transakcí (START TRANSACTION, ROLLBACK, COMMIT)
- Volání procedur (CALL)
- Příkazy pro nastavování práv (GRANT, REVOKE) a některé další příkazy

Na druhou stranu mohou triggery oproti uloženým procedurám přistupovat k datům, která se právě mění. To například znamená, že trigger, který se spouští před aktualizací nějaké tabulky má přístup k hodnotám těch řádků, které se snažíme změnit. Také to znamená, že v triggeru můžeme provést nějaké rozhodnutí v závislosti na datech, která má tento trigger měnit.

Volání triggeru v databázi:

- Databáze zachytí požadavek na změnu dat.
- DMBS zjistí, zda je pro tuto tabulku definován trigger, který se má provést před aktualizací záznamů. Pokud ano, provede jej.
- Jestliže trigger aktualizaci nezrušil, zapíší se data do tabulky.
- DMBS zjistí, zda je pro tuto tabulku definován trigger, který se má provést po aktualizací záznamů. Pokud ano, provede jej.
- Dokončeno. Systémové prostředky použité pro zápis dat jsou vráceny systému a čeká se na další požadavek.

Příklad použití triggeru:

Vytvoříme tabulku:

```
CREATE TABLE data (id int, jmeno varchar(50), plat int(11) not null);
```

S testovacími daty:

```
insert into data (jmeno, plat) values ('Petr', 15000);
insert into data (jmeno, plat) values ('Pavel', 11000);
insert into data (jmeno, plat) values ('Jan', 13000);
```

Pokud budeme chtít např. kontrolovat velikost platu a omezit jeho hodnotu na max. 20 000, můžeme zavést trigger:

```
create trigger Plat
before insert
on data
for each row
begin
  if new.plat>20000 then new.plat=20000;
  end if;
end;
```

Základní podmínky pro tvorbu triggerů:

- Trigger se tvoří pomocí příkazu **create trigger**
- Jméno triggeru musí být pro danou databázi jedinečné. To znamená, že nelze mít dva shodně pojmenované triggery ani na dvou různých tabulkách.
- U každého triggeru musí být uvedeno, před nebo po jaké akci je spuštěn. Možnosti jsou BEFORE INSERT, BEFORE UPDATE, BEFORE DELETE, AFTER INSERT, AFTER UPDATE, AFTER DELETE. V MySQL navíc platí omezení, že pro každou z vyjmenovaných akcí smí existovat nejvýše jeden trigger.
- Každý trigger musí souviset právě s jednou tabulkou.
- Triggery mají přístup k měněným datům, která jsou reprezentován virtuálními tabulkami old a new. Jejich význam je ten, že tabulka old obsahuje původní data a tabulka new obsahuje konečná data
- V triggeru lze využít i vestavěných funkcí.
- Trigger se neprovede při použití příkazu truncate table. Je to vlastnost DMBS, takže pokud se spoléháte na nějakou akci, která se spustí při odstraňování dat z tabulky, buďte opatrní. Při použití DELETE FROM TABLE se však trigger spustí a zpracuje všechny řádky, které ještě v nebohé tabulce zbývají.

Výhody:

- Při zálohování dat není třeba programovat speciální kód na zálohování změn v tabulkách.

Nevýhody:

- Pokud budeme mít triggerů více a/nebo pokud budou provádět časově dlouhé operace, můžete si snížit dobu odezvy databáze na akční dotazy.
- Triggery se obtížně ladí. Klasický akční dotaz změní data a skončí; trigger může data modifikovat, uložit jinam nebo operaci zakázat. Díky tomu bývá občas velmi obtížné zjistit, co se vlastně v databázi děje
- Ne všechny SQL příkazy lze v triggeru použít.

6.2.3 Použití PHP funkcí uložených v MySQL databázi

Další z možností kontroly dat je použití PHP funkcí, které jsou uloženy v databázi jako text. Při jejich implementaci do zdrojového kódu využijeme PHP funkce eval. Na rozdíl od uložených procedur a triggerů nám pro funkčnost stačí nižší verze MySQL databáze např. 4.0

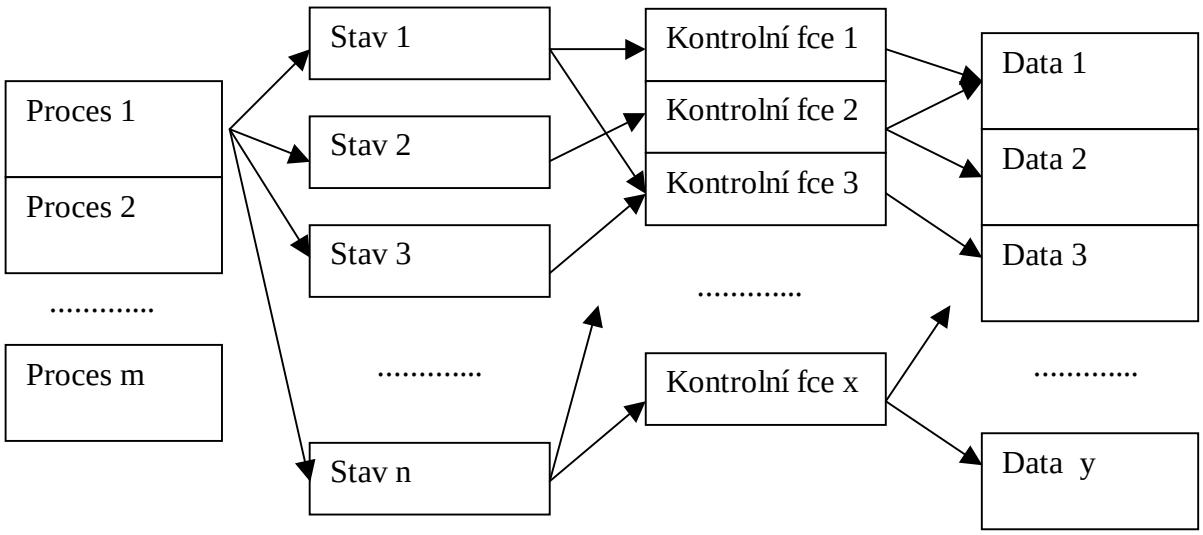
Funkce EVAL

Funkce **eval()** vyhodnotí předaný řetězec jako PHP kód. Kromě jiného se dá využít na ukládání kódu v textovém sloupci databáze pro pozdější vykonání. Hodnoty přiřazené proměnným v **eval()** těmto proměnným zůstanou i v hlavním skriptu.

Příklad použití funkce eval:

```
<?php
$string = 'cup';
$name = 'coffee';
$str = 'This is a $string with my $name in it.<br>';
echo $str;
eval ("\$str = \"$str\";");
echo $str;
?>
```

Celý princip by měl spočívat v tom, že v databázi budeme mít v tabulce uloženy všechny funkce kontrol, které využíváme pro kontrolu všech procesů v informačním systému. Následně pak ze zdrojového programu budeme vždy volat jen konkrétní kontroly pro daný proces. Princip ukazuje následující obrázek.

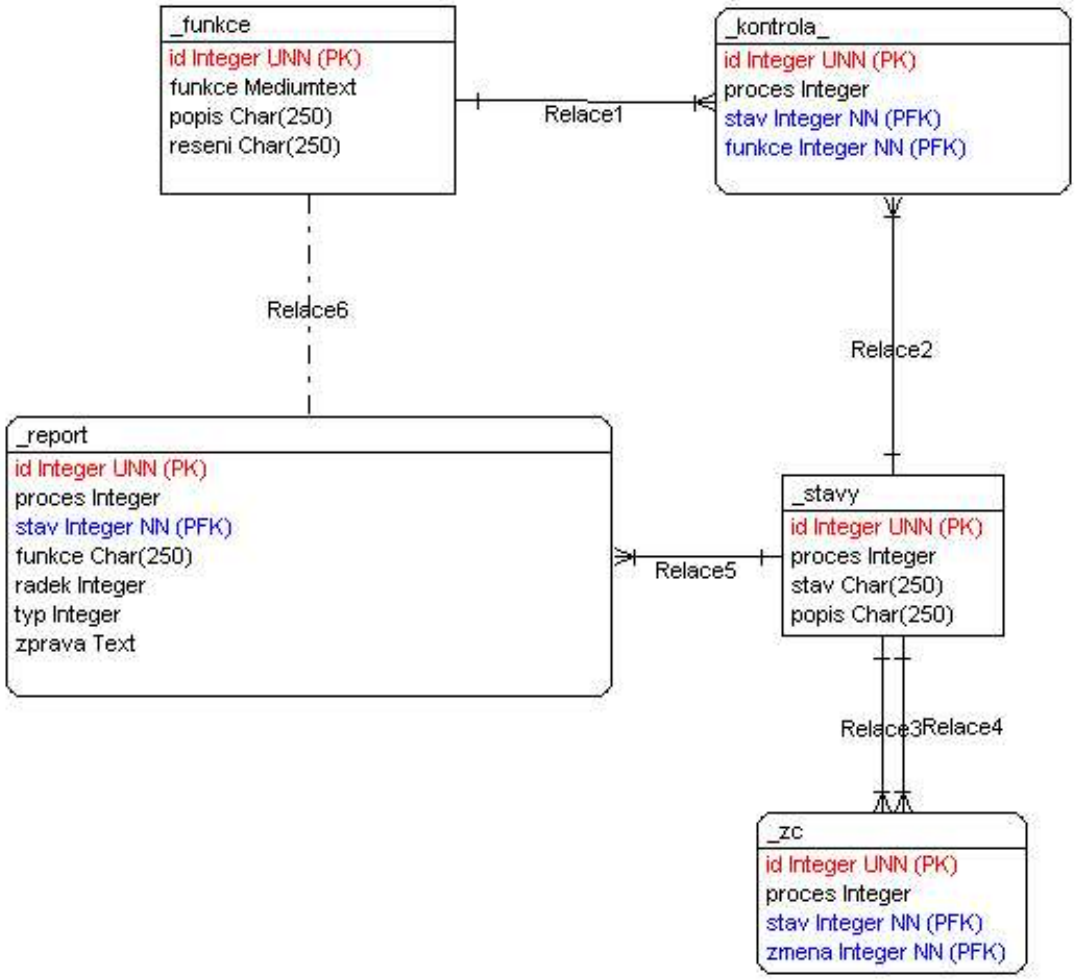


Obrázek 6.2 – Model kontroly dat

Jak je patrné z obrázku, každý proces na základě stavů životního cyklu může obsahovat 1 až x kontrolních funkcí nad y daty.

6.2.3.1 Definice databázové struktury

Databázová struktura v MySQL se skládá z pěti tabulek, které postačí pro bezproblémový chod celého systému kontroly. Jedná se o tabulku „_funkce“, „_kontrola“, „_report“, „_stavy“ a tabulku „_zc“.

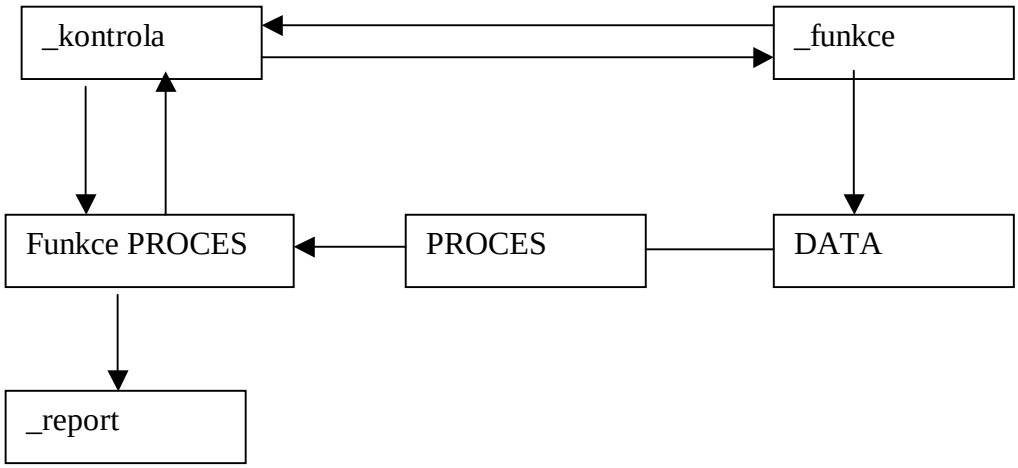


Obrázek 6.3 – ERD diagram

- Relace1 představuje vazbu mezi tabulkou _funkce a _kontrola, v tabulce kontrola je obsažen foreign key tabulky _funkce. Tato relace říká, která funkce má být volána.
- Relace2 představuje vazbu na stavy životního cyklu z tabulky _stavy
- Relace3 a Relace4 představují propojení tabulky _zc, která obsahuje životní cyklus do tabulky _stavy, první vazba je číslo stavu a druhá následující stav.

- Relace5 slouží jako spojení do tabulky reportu chyb a upozornění _report, kde je přiřazena ke každé chybě (upozornění) i informace v jakém stavu k události došlo.
- Relace6 je pouze jako informativní vazba, představující že v reportu je i údaj o funkci která událost (chybu nebo upozornění) vyvolala.

Detailněji budou popsány jednotlivé tabulky dále.



Obrázek 6.4 – Princip činnosti kontroly

PROCES (ve stavu životního cyklu např. 1) nad daty DATA zavolá funkci PROCES, která na základě stavu životního cyklu zavolá z tabulky _kontrola příslušné kontrolní funkce. Výsledek kontroly je pak uložen do tabulky _report.

_funkce (funkce kontrol uložené v databázi)

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(11)			Ne		auto_increment
funkce	mediumtext	utf8_czech_ci		Ano	*NULL*	
popis	varchar(250)	utf8_czech_ci		Ano	*NULL*	
reseni	char(250)	utf8_general_ci		Ano	*NULL*	

_kontrola (přiřazení funkcí kontrol danému stavu v životním cyklu)

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(10)			Ne		auto_increment
proces	int(5)			Ne		
stav	int(5)			Ne	0	
funkce	int(5)			Ne	0	

_report (výsledky kontroly)

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(11)			Ne		auto_increment
proces	int(5)			Ne		
stav	int(5)			Ne		
funkce	char(50)	utf8_general_ci		Ano	*NULL*	
radek	int(10)			Ne	0	
typ	int(1)			Ne		
zprava	text	utf8_general_ci		Ano	*NULL*	

_stavy (popis stavů životního cyklu)

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(11)			Ne		auto_increment
proces	int(5)			Ne		
stav	char(250)	utf8_czech_ci		Ano	*NULL*	
popis	char(250)	utf8_general_ci		Ano	*NULL*	

_zc (životní cyklus)

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(11)			Ne		auto_increment
proces	int(5)			Ne		
stav	int(5)			Ne		
zmena	int(5)			Ne		

Obrázek 6.5 – Datový model

6.2.3.1.1 Tabulka – _funkce

Tabulka obsahuje kromě primárního klíče sloupec funkce typu mediumtext, kde jsou uloženy všechny funkce sloužící pro kontrolu v datech. Sloupec popis slouží pouze k upřesnění (popisu) a lepší orientaci v přehledu funkcí.

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(11)			Ne		auto_increment
funkce	mediumtext	utf8_czech_ci		Ano	*NULL*	
popis	varchar(250)	utf8_czech_ci		Ano	*NULL*	
reseni	char(250)	utf8_general_ci		Ano	*NULL*	

Obrázek 6.6 – Tabulka _funkce, definice sloupců

```
CREATE TABLE `_funkce` (
   `id` int(11) NOT NULL auto_increment,
   `funkce` mediumtext character set utf8 collate utf8_czech_ci,
   `popis` varchar(250) character set utf8 collate utf8_czech_ci
default NULL,
   `reseni` char(250) default NULL,
   PRIMARY KEY  (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
```

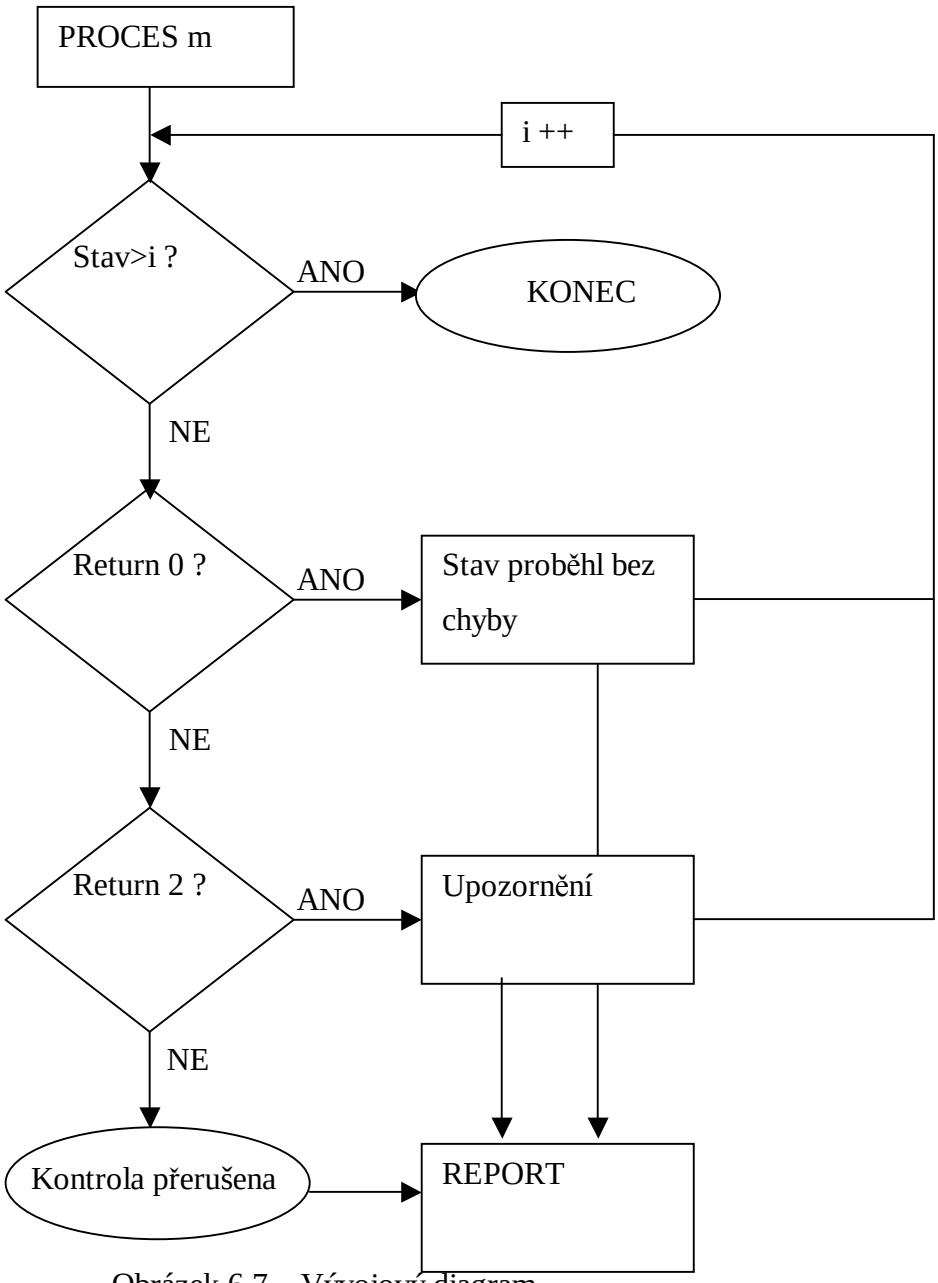
Definice funkce – předpis pro syntaxi funkce

Název funkce je složen ze slova „kontrola“ a čísla primárního klíče v tabulce „Funkce“ (tzn. první funkce musí mít název kontrola1), má pouze jeden parametr, kterým je číslo řádku v datech kde se provádí kontrola. Funkce musí vždy vracet hodnotu 0-2, přičemž 0 znamená že testovací podmínka je splněna, 1 – došlo k ke stavu, pro který není možné pokračovat dále v procesu, 2 – upozornění, že nastal stav, který se blíží nějaké mezní hodnotě, ale nemá přímý vliv pro pokračování procesu.

Ilustrativní příklad funkce s názvem kontrola1:

```
function kontrola1($id){
if (x<1) return 0; //podmínka splněna
if (x==1) return 2; //mezní stav
if (x>1) return 1; //podmínka nesplněna
}
```

Vývojový diagram možných událostí



Obrázek 6.7 – Vývojový diagram

V obrázku 6.2 bylo uvedeno, že počet stavů je „n“. Ve vývojovém diagramu je však uvedena proměnná „i“, je tak z toho důvodu, že proměnná i nepředstavuje počet stavů v daném procesu, ale pouze počet procesů ke kterým dojde v konkrétním životním cyklu (tzn. např. stavy 1 – 2 – 5, i = 3).

6.2.3.1.2 Tabulka – _kontrola

Tabulka slouží pro třídění kontrol a jejich přiřazování k jednotlivým procesům.

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(10)			Ne		auto_increment
proces	int(5)			Ne		
stav	int(5)			Ne	0	
funkce	int(5)			Ne	0	

Obrázek 6.8 – Tabulka _kontrola, definice sloupců

```
CREATE TABLE `_kontrola` (
   `id` int(10) NOT NULL auto_increment,
   `proces` int(5) NOT NULL,
   `kontrola` int(5) NOT NULL default '0',
   `funkce` int(5) NOT NULL default '0',
    PRIMARY KEY  (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
```

Definice hlavní funkce

Hlavní funkce se nemění a zůstává stále stejná, umístěna v hlavním programu. Slouží pro přiřazování zvolených kontrol daného procesu k jednotlivým funkcím kontrol podle tabulky „Kontrola“. Funkce vrací buď hodnotu 0 v případě že během procesu nedošlo k žádné chybě, hodnotu 1 pokud došlo k události po které nemůže proces dále pokračovat (nelze přejít do následujícího stavu) a hodnotu 2, pokud došlo k události která vyžaduje upozornění, ale nemá zásadní vliv na přechod procesu z jednoho stavu do druhého. Funkce má tři parametry, první udává o který proces se jedná, další je číslo stavu životního cyklu v daném procesu (např. pokud během procesu zavedeme 4 kontroly, první kontrolu voláme s parametrem 1,1 ; druhou 1,2 atd.), třetí parametr je číslem řádku v tabulce dat, která chceme kontrolovat.

Hlavní funkce PROCES

```
/*****************************************************/
/*        FUNKCE PROCES                              */
/*****************************************************/
function proces($proces,$stav,$id){
global $stavold, $prubeh, $tabulkazc, $tabulkastavy, $tabulkareport, $tabulkaproces, $tabulkafunkce, $tabulkakontrola;
        $prubeh=0;
        if ($stavold[$proces]){
        @$vysledekK = mysql_query("select * from $tabulkazc where proces = $proces and zmena=$stav and stav=$stavold[$proces]");
        if (mysql_num_rows($vysledekK)==0){
        @$vysledekC = mysql_query("insert into $tabulkareport ( proces , stav , funkce, radek, typ, zprava ) values ( '$proces' , '$stav', '$kon' ,  '$id','$prubeh', 'Došlo k závažné chybě, Chybný průběh životního cyklu procesu' )");
        die ("Došlo k závažné chybě, Chybný průběh životního cyklu procesu");
        }
        }
        $stavold[$proces] = $stav;

        @$vysledekF = mysql_query("select * from $tabulkakontrola left join $tabulkafunkce on $tabulkakontrola.funkce = $tabulkafunkce.id where $tabulkakontrola.proces=$proces and $tabulkakontrola.stav=$stav");
          while($zaznamF=mysql_fetch_array($vysledekF)){
          $kon = "kontrola".$zaznamF["id"];
          if (!function_exists($kon)) eval ("".$zaznamF["funkce"].");
            if ($kon($id) > 0){
            if ($kon($id) == 1) $text = "Chyba";
            if ($kon($id) == 2) $text = "Upozornění";
            $prubeh=$kon($id);
            @$vysledekC = mysql_query("insert into $tabulkareport ( proces , stav ,
```

```
funkce, radek, typ, zprava ) values ( '$proces' , '$stav', '$kon' ,  '$id','$prubeh', '$text: v procesu $proces v kontrole stavu $stav ve funkci $kon v datech na radku c. $id Řešení: $zaznamF[reseni]' )");
                if ($kon($id)==1) die ("Došlo k závažné chybě, Chyba v procesu $proces v kontrole stavu $stav ve funkci $kon v datech na radku c. $id Řešení: $zaznamF[reseni] ");
            }else{
            @$vysledekC = mysql_query("insert into $tabulkareport ( proces , stav , funkce, radek, typ, zprava ) values ( '$proces' , '$stav', '$kon' ,  '$id', '$prubeh', 'Během procesu nedošlo k žádným chybám.' )");
            }
          }
        return $prubeh; //pokud vse ok $prubeh=0
}
```

6.2.3.1.3 Tabulka _report

Tabulka zaznamenává stavy při průběhu kontroly. Pro každou funkci, která je volána pro danou kontrolu se provede vyhodnocení, zda proběhla v pořádku či nikoliv a údaj se zaznamená.

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(11)			Ne		auto_increment
proces	int(5)			Ne		
stav	int(5)			Ne		
funkce	char(50)	utf8_general_ci		Ano	*NULL*	
radek	int(10)			Ne	0	
typ	int(1)			Ne		
zprava	text	utf8_general_ci		Ano	*NULL*	

Obrázek 6.9 – Tabulka _report, definice sloupců

```
CREATE TABLE `_report` (
  `id` int(11) NOT NULL auto_increment,
  `proces` int(5) NOT NULL,
  `kontrola` int(5) NOT NULL,
  `funkce` char(50) default NULL,
  `radek` int(10) NOT NULL default '0',
  `typ` int(1) NOT NULL,
  `zprava` text,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
```

Detailní popis sloupců:

- proces – číslo procesu který se kontroluje
- stav – číslo stavu pro daný proces, v kterém se provádí kontrola
- funkce – název funkce, která byla provedena
- radek – číslo řádku v datech, která byla kontrolována
- typ – typ stavu ke kterému došlo (0 – kontrola v pořádku, 1 – chyba, která přerušila proces, 2 – upozornění)
- zprava – výsledek kontroly, informace zda proběhla správně nebo zda se vyskytl problém s návrhem řešení (který je uložen v tabulce „Funkce“ ve sloupci řešení)

6.2.3.1.4 Tabulka _stavy

Tabulka stavů procesu definuje všechny stavy, ke kterým dochází během životního cyklu.

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(11)			Ne		auto_increment
proces	int(5)			Ne		
stav	char(250)	utf8_czech_ci		Ano	*NULL*	
popis	char(250)	utf8_general_ci		Ano	*NULL*	

Obrázek 6.10 – Tabulka _stavy, definice sloupců

- proces – číslo procesu (m), kterého se stav týká

- stav – číslo stavu (n) v životním cyklu
- popis – název stavu slovně (např. čekající)

6.2.3.1.5 Tabulka _zc

Tabulka obsahuje hierarchii životního cyklu, tzn. všechny možnosti kdy proces přechází z jednoho do následujícího stavu (životní cyklus)

Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra
id	int(11)			Ne		auto_increment
proces	int(5)			Ne		
stav	int(5)			Ne		
zmena	int(5)			Ne		

Obrázek 6.11 – Tabulka _zc, definice sloupců

- proces – číslo procesu (m), kterého se stav týká
- stav – číslo stavu (n) v životním cyklu
- zmena – číslo následujícího stavu (může jich být víc, např. proces ve stavu 1 může přecházet do stavu 2 nebo 3)

Výhody:

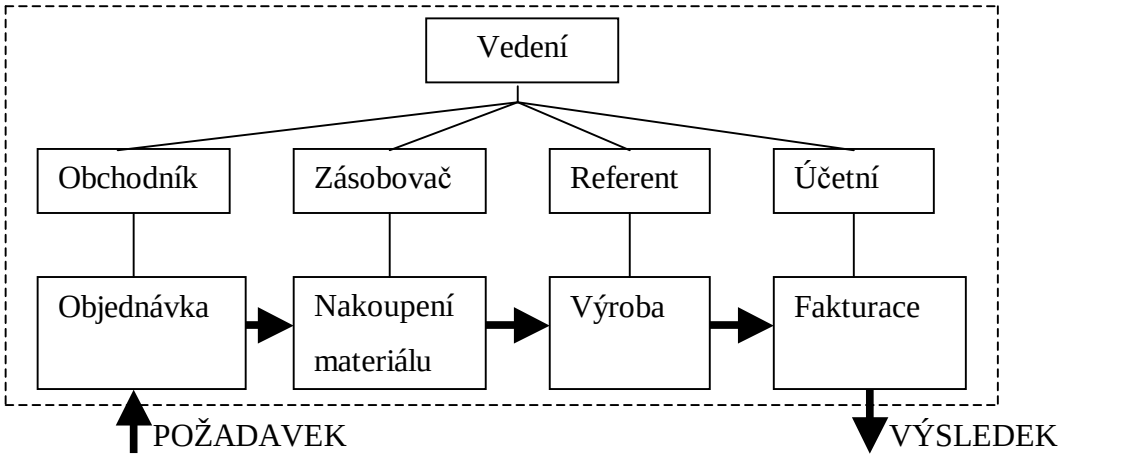
- snadná úprava jednotlivých kontrol přímo v databázi, bez nutnosti zasahovat do zdrojového kódu
- jednoduché volání kontrolních funkcí přes funkci proces
- řešení lze použít i pro nižší verze MySQL
- možnost kombinace se všemi předchozími řešeními (uživatelský formulář, procedura, trigger)

Nevýhody:

- volané kontrolní funkce tvoří kód

6.2.3.2 Praktická ukázka kontroly procesů

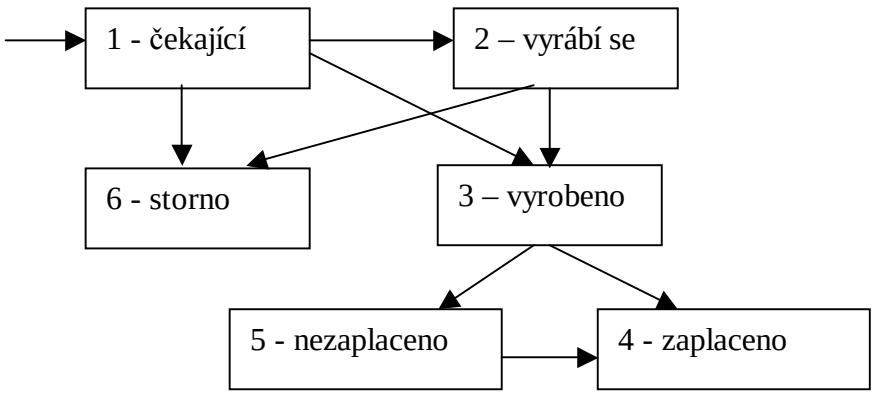
Pomocí předchozího postupu lze kontrolu aplikovat i na složitější procesy. Příkladem mohou být procesy v podniku, při jeho řízení apod. Mějme následující strukturu typu „Vyřízení objednávky“:



Obrázek 6.12 – Podnikový proces

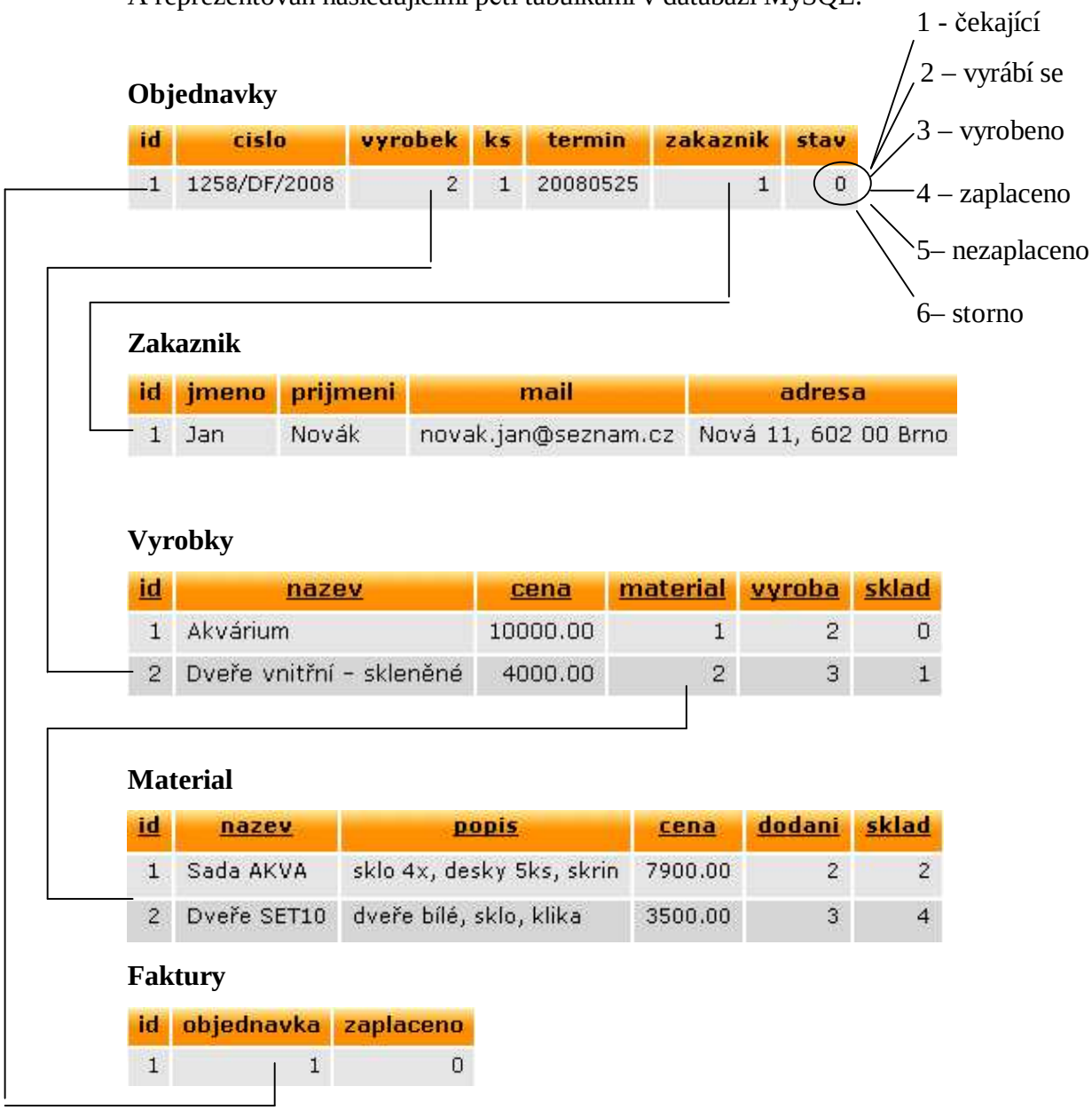
Z uvedeného obrázku vidíme, že se jedná o produkční workflow a kontrola by měla probíhat především vždy při přechodu z jednoho stavu procesu do druhého, tzn. nejméně 3 kontroly uvnitř podniku. Také je patrné, že se jedná o proces zhotovení výrobku, požadavkem je tedy dodání výrobku za určitý čas. Jedním ze stěžejních kritérií pro kontrolu bude tedy dodržení termínů.

Nejdříve je třeba zajistit aby došlo k podchycení všech důležitých stavů životního cyklu. Model stavů procesu dodání výrobku by mohl vypadat např. takto:



Obrázek 6.13 – Životní cyklus procesu

A reprezentován následujícími pěti tabulkami v databázi MySQL:



Objednavky

id	cislo	vyrobek	ks	termin	zakaznik	stav
1	1258/DF/2008	2	1	20080525	1	0

Zakaznik

id	jmeno	prijmeni	mail	adresa
1	Jan	Novák	novak.jan@seznam.cz	Nová 11, 602 00 Brno

Vyrobky

id	nazev	cena	material	vyroba	sklad
1	Akvárium	10000.00	1	2	0
2	Dveře vnitřní - skleněné	4000.00	2	3	1

Material

id	nazev	popis	cena	dodani	sklad
1	Sada AKVA	sklo 4x, desky 5ks, skrin	7900.00	2	2
2	Dveře SET10	dveře bílé, sklo, klika	3500.00	3	4

Faktury

id	objednavka	zaplaceno
1	1	0

Obrázek 6.14 – Datový model podnikového procesu

Nejprve přidáme do databáze 5 tabulek potřebných pro kontrolu, tak že databáze bude obsahovat těchto deset tabulek:

faktury	_funkce
material	_kontrola
objednavky	_report

vyrobky _stavy

zakaznik _zc

Nyní si podle modelu stavů procesu nadefinujeme tabulku _stavy (zvolíme si číslo procesu např. 1):

id	proces	stav	popis
1	1	1	Čekající
2	1	2	Vyrábí se
3	1	3	Vyrobeno
4	1	4	Zaplaceno
5	1	5	Nezaplaceno
6	1	6	Storno

Obrázek 6.15 – Definování tabulky _stavy

V tabulce _zc pak provedeme definici životního cyklu. Tabulka _zc by měla na základě životního cyklu z modelu stavů vypadat takto:

id	proces	stav	zmena
1	1	1	2
2	1	1	6
3	1	1	3
4	1	2	3
5	1	2	6
6	1	3	4
7	1	3	5
8	1	5	4

Obrázek 6.16 – Definování tabulky _zc

Nyní můžeme začít vytvářet jednotlivé funkce kontrol pro stavy procesu. Jako změnu stavu budeme brát změnu údaje v tabulce „objednavka“ ve sloupci stav. Počáteční stav má číslo 1, tzn. je ve stavu čekající, první funkcí můžeme např. testovat tento stav. Zda je roven jedné.

Funkce by pak mohla vypadat takto:

```
function kontrola1($id){
$dotaz="select * from objednavky where id=$id";
@$vysledek = mysql_query($dotaz);
$zaznam=mysql_fetch_array($vysledek);
  if ($zaznam["stav"]==1){
  return 0;
  }else{
  return 1;
  }
}
```

popis: Kontrola stavu 1

reseni: Stav nemůže být různý od 1

V databázi v tabulce _funkce budeme mít tedy tento záznam:

id	funkce	popis	reseni

1	function kontrola1($id){
	$dotaz="select * from objednavky
	where id=$id";
	@$vysledek = mysql_query($dotaz);
	$zaznam=mysql_fetch_array
	($vysledek);
	if ($zaznam["stav"]==1){
	return 0;
	}else{
	return 1;
	}
	}
		kontrola stavu 1	Stav nemůže být různý od 1

Obrázek 6.17 – Definování tabulky _funkce

V tabulce _kontrola navážeme funkci na stav 1:

id	proces	stav	funkce
1	1	1	1

Obrázek 6.18 – Definování tabulky _kontrola

Nyní si vytvoříme skript, který nám bude plnit funkci informačního systému, tzn. bude nám simulovat změny stavu v procesu. Nejprve si vytvoříme připojení k databázi:

```
$server = 'localhost';
$dbuser = 'root';
$dbpasswd = '';
$db = 'projekt2';
$tabulkazc='_zc';
$tabulkastavy='_stavy';
$tabulkafunkce='_funkce';
$tabulkakontrola='_kontrola';
$tabulkareport='_report';

$connection = mysql_pconnect($server,$dbuser,$dbpasswd) ;
mysql_select_db($db,$connection);
if (!$connection){
  echo "Nepodařilo se připojit k databázi";
}else{
mysql_query("SET CHARACTER SET utf8",$connection);
```

Dále vkopírujeme funkci proces viz. předchozí kapitola – Definice hlavní funkce a pro přehlednost si můžeme ještě vytvořit funkci, která nám bude zobrazovat data v tabulkách dat, funkci „zobrazdata“:

```
        function zobrazdata($tabulka){
          echo "<br />".$tabulka."<br />";
          @$vysledek = mysql_query("select * from $tabulka");
          while($zaznam=mysql_fetch_array($vysledek, MYSQL_ASSOC)){
print_r($zaznam);echo "<br />";}
}
```

Nyní můžeme zobrazit všechna data v tabulkách voláním funkce „zobrazdata“:

```
zobrazdata("objednavky");
zobrazdata("zakaznik");
zobrazdata("vyrobky");
zobrazdata("material");
zobrazdata("faktury");
```

Vytvoříme a nadefinujeme pole, která nám bude představovat životní cyklus:

```
$zivotnicyklus = array(1,3,4);
```

Čísla 1,3,4 znamenají, že ze stavu 1 přecházíme do stavu 3 a následně do stavu 4. Nyní pomocí příkazu *while* vytvoříme cyklickou podmínku na základě pole definující životní cyklus s voláním kontroly pro daný stav pomocí funkce proces:

```
$i = 0; $radek = 1;
while (count($zivotnicyklus)>$i){

proces(1,$zivotnicyklus[$i],$radek);

switch ($zivotnicyklus[$i]):
case "1":
   @$vysledek = mysql_query("update objednavky set stav=1 where id=$radek");
break;
case "2":
   @$vysledek = mysql_query("update objednavky set stav=2 where id=$radek");
break;
case "3":
   @$vysledek = mysql_query("update objednavky set stav=3 where id=$radek");
break;
```

```
case "4":
  @$vysledek = mysql_query("update objednavky set stav=4 where id=$radek");
break;
case "5":
  @$vysledek = mysql_query("update objednavky set stav=5 where id=$radek");
break;
case "6":
  @$vysledek = mysql_query("update objednavky set stav=6 where id=$radek");
break;
        endswitch;
  $i++;
}
```

Proměnná $radek znamená který řádek v tabulce dat budeme kontrolovat, nastavíme např. na první řádek pro všechny stavy. Vycházíme z toho, že všechny stavy jsou vazané k prvnímu řádku v tabulce „objednavka“. Po volání funkce proces následuje přepínač pomocí příkazu *switch*, který nám zajistí přepnutí do následujícího stavu po proběhnutí kontroly.

Skript ukončíme uzavřením spojením s databází:

```
        mysql_close($connection);
}
```

a můžeme testovat jaké záznamy se nám objeví v tabulce _report. Po proběhnutí skriptu by měla tabulka _report vypadat takto:

id	proces	stav	funkce	radek	typ	zprava
1	1	1	kontrola1	1	0	Během procesu nedošlo k žádným chybám.

Obrázek 6.19 – Výpis z tabulky _report

Vidíme, že v procesu 1 (náš proces kontroly výroby výrobku) ve stavu 1 (čekající objednávka) po provedení kontrolní funkce „kontrola1“ (test stavu zda je roven jedné) nedošlo k žádné chybě (typ je roven nule).

Pokud bychom pustili skript znova vypíše nám report toto:

id	proces	stav	funkce	radek	typ	zprava
1	1	1	kontrola1	1	1	Chyba v procesu 1 v kontrole stavu 1 ve funkci kontrola1 v datech na radku c. 1 Řešení: Stav nemůže být různý od 1

Obrázek 6.20 – Výpis z tabulky _report

Protože proces podle nastavení cyklu

$zivotnicyklus = array(1,3,4);

skončil ve stavu 3 a podmínka pro kontrolu1 už neplatí.

Abychom stále nemuseli měnit zpět nastavení dat v databázi (hodnotu stavu v objednávce na 1 apod.) můžeme si napsat jednoduchý skript, který nám bude tabulky s daty obsluhovat:

Skript *reset.php*

```
<?
$connection = mysql_pconnect($server,$dbuser,$dbpasswd) ;
mysql_select_db($db,$connection);
        if (!$connection){
          echo "Nepodařilo se připojit k databázi";
        }else{
        mysql_query("SET CHARACTER SET utf8",$connection);

        @$vysledek = mysql_query("update objednavky set stav=1 where id=1");
        @$vysledek = mysql_query("update vyrobky set sklad=5 where id=2");
        @$vysledek = mysql_query("update faktury set zaplaceno=0 where id=1");
        @$vysledek = mysql_query("truncate $tabulkareport");


          mysql_close($connection);
}
?>
```

Můžeme jej rovnou spustit a zkusit změnit nastavení životního cyklu na nějakou nesmyslnou posloupnost pro ověření zda se proces zastaví (kontrola stavů vychází z definice v tabulce _zc a je implementována do funkce process), např. takto:

$zivotnicyklus = array(1,3,6);

Ve skriptu dojde k vypsání varovného hlášení „Došlo k závažné chybě, Chybný průběh životního cyklu procesu“ a proces se zastaví. Tabulka reportu pak vypadá takto:

id	proces	stav	funkce	radek	typ	zprava
1	1	1	kontrola1	1	0	Během procesu nedošlo k žádným chybám.
2	1	3	kontrola4	1	0	Během procesu nedošlo k žádným chybám.
3	1	6		1	0	Došlo k závažné chybě, Chybný průběh životního cyklu procesu

Obrázek 6.21 – Výpis z tabulky _report

Kontrola ve stavu 1 a 3 proběhla v pořádku, protože byla v souladu s životním cyklem, k zastavení došlo až u stavu 6.

Můžeme přidat další kontroly, např. pro kontrolu termínu objednávky před výrobou (stav 2), kontrolu ceny materiálu (stav 2), která může být typu 2 (což znamená že se jedná o upozornění, která nám nebude zastavovat proces), pro stav 3, tzn. vyrobeno, můžeme např. testovat zda máme nějaký výrobek na skladě a pro stav 4 např. zda byla uhrazena faktura. Zmíněné čtyři funkce kontrola2 až kontrola5 jsou uvedeny zde:

Funkce kontrola2 pro kontrolu termínu:

```
function kontrola2($id){
    $dotaz="select * from objednavky where id=$id";
    @$vysledek = mysql_query($dotaz);
    $zaznam=mysql_fetch_array($vysledek);
    $datum = $zaznam["termin"];
```

```
        $vyrobek = $zaznam["vyrobek"];

        $dotaz="select * from vyrobky where id=$vyrobek";
        @$vysledek = mysql_query($dotaz);
        $zaznam=mysql_fetch_array($vysledek);
        $vyr = $zaznam["vyroba"];
        $material = $zaznam["material"];

        $dotaz="select * from material where id=$material";
        @$vysledek = mysql_query($dotaz);
        $zaznam=mysql_fetch_array($vysledek);
        $dod = $zaznam["dodani"];
        $d = date("Ymd")+$vyr+$dod;
        if ($d <= $datum){
        return 0;
        }else{
        return 1;
        }
}
```

popis: Kontrola termínu výroby

reseni: Termín pro vyrobení je příliš krátký

Funkce kontrola3 pro kontrolu ceny materiálu:

```
        function kontrola3($id){
        $dotaz="select * from objednavky where id=$id";
        @$vysledek = mysql_query($dotaz);
        $zaznam=mysql_fetch_array($vysledek);
        $vyrobek = $zaznam["vyrobek"];
```

```
        $dotaz="select * from vyrobky where id=$vyrobek";
        @$vysledek = mysql_query($dotaz);
        $zaznam=mysql_fetch_array($vysledek);
        $cena1 = $zaznam["cena"];
        $material = $zaznam["material"];

        $dotaz="select * from material where id=$material";
        @$vysledek = mysql_query($dotaz);
        $zaznam=mysql_fetch_array($vysledek);
        $cena2 = $zaznam["cena"];

        if ($cena1 > $cena2){
        return 0;
        }else{
        return 2;
        }
}
```

popis: Kontrola stavu 0

reseni: Stav nemůže být různý od 0

Kontrola4 – máme výrobek na skladě?:

```
        function kontrola4($id){
        $dotaz="select * from objednavky where id=$id";
        @$vysledek = mysql_query($dotaz);
        $zaznam=mysql_fetch_array($vysledek);
        $vyrobek = $zaznam["vyrobek"];
        $ks1 = $zaznam["ks"];

        $dotaz="select * from vyrobky where id=$vyrobek";
```

```
        @$vysledek = mysql_query($dotaz);
        $zaznam=mysql_fetch_array($vysledek);
        $ks2 = $zaznam["sklad"];

        if ($ks1 <= $ks2){
        return 0;
        }else{
        return 1;
        }
}
```

popis: Výrobek na skladě?

reseni: Výrobek se nevyrobil

Kontrola5, zaplacení faktury

```
function kontrola5($id){
$dotaz="select * from faktury where objednavka=$id";

@$vysledek = mysql_query($dotaz);
$zaznam=mysql_fetch_array($vysledek);
$zaplaceno = $zaznam["zaplaceno"];

if ($zaplaceno == 1){
return 0;
}else{
return 1;
}
}
```

popis: Faktura k objednávce je zaplacena?

reseni: Nedošlo k zaplacení faktury

Nyní provedeme přiřazení jednotlivých funkcí k daným kontrolám, pak by tabulka „_kontrola“ měla vypadat následovně (stavu 4 přiřadíme znovu kontrolu termínu, tzn. funkci kontrola2):

id	proces	stav	funkce
1	1	1	1
2	1	2	2
3	1	2	3
4	1	3	4
5	1	4	2
6	1	4	5

Obrázek 6.22 – Výpis tabulky _kontrola

Nastavíme cyklus na objednání – vyrábění – vyrobeno – zaplaceno:

$zivotnicyklus = array(1,2,3,4);

a do přepínače switch musíme ještě přidat informaci o zapalcení faktury, kterou nám testuje funkce kontrola5. Přepínač 3 pak bude vypadat takto:

```
case "3":
@$vysledek = mysql_query("update faktury set zaplaceno=1 where id=1");
@$vysledek = mysql_query("update objednavky set stav=3 where id=$radek");
break;
```

Po resetu dat (reset.php) a spuštění skriptu bychom měli dostat v tabulce reportu tyto výsledky:

id	proces	stav	funkce	radek	typ	zprava
1	1	1	kontrola1	1	0	Během procesu nedošlo k žádným chybám.
2	1	2	kontrola2	1	0	Během procesu nedošlo k žádným chybám.
3	1	2	kontrola3	1	0	Během procesu nedošlo k žádným chybám.
4	1	3	kontrola4	1	0	Během procesu nedošlo k žádným chybám.
5	1	4	kontrola2	1	0	Během procesu nedošlo k žádným chybám.
6	1	4	kontrola5	1	0	Během procesu nedošlo k žádným chybám.

Obrázek 6.23 – Výpis tabulky _report

Z tabulky je patrné, že došlo k provedení všech kontrol (1 až 5) pro stavy 1 až 4 a nebyl zjištěn žádný problém. A dále, že objednávka přešla ze stavu 1 do stavu 4.

6.2.3.2.1 Chyba během procesu (typ 1)

Tuto chybu jsme si už vyzkoušeli u funkce kontrola1. Nyní si ověříme zda při chybě typu 2 (upozornění) opravdu nedojde k zastavení procesu.

6.2.3.2.2 Upozornění během procesu (typ 2)

Typem 2 je definovaná funkce kontrola3, která nám hlídá zda cena materiálu pro výrobu není vyšší než cena výrobku. V databázi tedy otevřeme tabulku „material“ a změníme cenu materiálu tak aby byla vyšší než cena výrobku (jedná se o matriál na řádku 2 – Dveře SET10):

id	nazev	popis	cena	dodani	sklad
1	Sada AKVA	sklo 4x, desky 5ks, skrin	7900.00	2	2
2	Dveře SET10	dveře bílé, sklo, klika	4500.00	3	4

Obrázek 6.24 – Tabulka material

V tabulce _report pak dostaneme následující stav:

id	proces	stav	funkce	radek	typ	zprava
1	1	1	kontrola1	1	0	Během procesu nedošlo k žádným chybám.
2	1	2	kontrola2	1	0	Během procesu nedošlo k žádným chybám.
3	1	2	kontrola3	1	2	Upozornění: v procesu 1 v kontrole stavu 2 ve funkci kontrola3 v datech na radku c. 1 Řešení: Cena materiálu je vyšší než výrobku
4	1	3	kontrola4	1	0	Během procesu nedošlo k žádným chybám.
5	1	4	kontrola2	1	0	Během procesu nedošlo k žádným chybám.
6	1	4	kontrola5	1	0	Během procesu nedošlo k žádným chybám.

Obrázek 6.25 – Výpis tabulky _report

Vidíme, že celý proces proběhl, pouze u kontroly3 došlo k upozornění, že cena materiálu je vyšší.

6.2.3.2.3 Zanoření dalšího procesu

Během procesů v konkrétních případech určitě dochází k tomu, že se jednotlivé procesy mezi sebou překrývají, ukažme si nyní jak by takové prolnutí s jiným procesem vypadalo.

Podle předchozí syntaxe budeme tedy další proces volat takto:

proces(2,1,$radek);

Stav samozřejmě nemusí být 1, jedná se samozřejmě také o proces na základě životního cyklu s n stavy. Ale pro ukázku nám bude stačit pouze jeho první stav.

Vytvoříme si novou funkci kontrola6, která nám bude sledovat uskladnění materiálu pro výrobu. Funkce by mohla vypadat například takto:

```
function kontrola6($id){
$dotaz="select * from material where id=$id";

@$vysledek = mysql_query($dotaz);
```

```
$zaznam=mysql_fetch_array($vysledek);
$nazev = $zaznam["nazev"];
$sklad = $zaznam["sklad"];

  if ($sklad < 5){
  return 2;
  }else{
  return 0;
  }
}
```

popis: Kontrola materiálu ve skladech

reseni: Je třeba objednat materiál

Funkce kontroluje zda se počet materiálu pro výrobek nesnížil na stav menší než 5. V tom případě se provede upozornění (typ 2).

Volání procesu 2 umístíme mezi stav 2- vyrábí se a 3 – vyrobeno, kde dochází k odečtu materiálu ze skladu pro potřeby výroby.

Volání s podmínkou pak vypadá takto:

if ($zivotnicyklus[$i]==3) proces(2,1,2);

Voláme všechny kontroly pro proces 2 ve stavu 1 pro řádek v datech číslo2.

Nezapomeneme, že musíme nastavit kontrolu k funkci v tabulce „_kontrola“:

id	proces	stav	funkce
1	1	1	1
2	1	2	2
3	1	2	3
4	1	3	4
5	1	4	2
6	1	4	5
7	2	1	6

Obrázek 6.26 – Výpis tabulky _kontrola

Nastavení v tabulce _zc provádět nemusíme, protože máme pouze jeden stav.

Průběh kontroly pak vypadá takto:

id	proces	stav	funkce	radek	typ	zprava
1	1	1	kontrola1	1	0	Během procesu nedošlo k žádným chybám.
2	1	2	kontrola2	1	0	Během procesu nedošlo k žádným chybám.
3	1	2	kontrola3	1	2	Upozornění: v procesu 1 v kontrole stavu 2 ve funkci kontrola3 v datech na radku c. 1 Řešení: Cena materiálu je vyšší než výrobku
4	1	3	kontrola4	1	0	Během procesu nedošlo k žádným chybám.
5	2	1	kontrola6	2	2	Upozornění: v procesu 2 v kontrole stavu 1 ve funkci kontrola6 v datech na radku c. 2 Řešení: Je třeba objednat materiál
6	1	4	kontrola2	1	0	Během procesu nedošlo k žádným chybám.
7	1	4	kontrola5	1	0	Během procesu nedošlo k žádným chybám.

Obrázek 6.27 – Výpis tabulky _report

Ve výsledném hlášení vidíme oba procesy a jejich stavy tak jak probíhaly za sebou. Přidání dalších procesů by probíhalo obdobně.

7. ZABEZPEČENÍ

Z důvodů, že kontrola probíhá nad informačním systémem založeným nad PHP a MySQL tzn. webovskou aplikací je nutné se ještě krátce zmínit o zabezpečení na které je nutné pamatovat.

7.1 OVĚŘENÍ UŽIVATELE

7.1.1 HTTP cookie

První z možností jak ověřit uživatele v aplikaci je ze strany klienta pomocí tzv. cookie, kterým se v protokolu HTTP označuje malé množství dat, která WWW server pošle prohlížeči, který je uloží na počítači uživatele.

Cookies neznamenají žádné nebezpečí pro počítač jako takový. Přesto cookies mohou být nebezpečné pro ochranu soukromí. Navštívený web si totiž může ukládat do cookies jakékoliv informace, které o návštěvníkovi zjistí a může tak postupně zjišťovat zájmy konkrétního návštěvníka. Které stránky navštěvuje, jaké informace vyhledává, jak často daný web navštěvuje apod.

7.1.2 SESSION

Výhodnější než cookie je použití Session. Session neboli relace umožňuje přesnou identifikaci a pohyb uživatele na serveru. Používá se také pro možnost vlastní úpravy vzhledu na www stránkách, kdy po vstupu na stránku server ověří identitu a zobrazí informace, novinky nebo zprávy přesně podle dřívějšího nastavení. Session jsou implementovány do PHP až od verze 4.0.

7.2 ZABEZPEČENÍ PŘIPOJENÍ

7.2.1 HTTPS

HTTPS je nadstavba počítačového protokolu HTTP, která poskytuje zvýšenou bezpečnost před odposloucháváním či podvržením dat. HTTPS není přímo zvláštní protokol, data jsou přenášena pomocí HTTP, ale data nejsou přenášena v běžném textu, ale jsou šifrována pomocí SSL nebo TLS. HTTPS implicitně komunikuje prostřednictvím TCP portu 443 (u nechráněného HTTP je to port 80)

5.2.2 SSL

Secure Sockets Layer, **SSL** je protokol, resp. vrstva vložená mezi vrstvu transportní (např. TCP/IP) a aplikační (např. HTTP), která poskytuje zabezpečení komunikace šifrováním a autentizaci komunikujících stran.

Protokol SSL se nejčastěji využívá pro bezpečnou komunikaci s internetovými servery pomocí HTTPS, což je zabezpečená verze protokolu HTTP. Po vytvoření SSL spojení (*session*) je komunikace mezi serverem a klientem šifrovaná a tedy zabezpečená.

5.2.3 TLS

Protokol **Transport Layer Security** (**TLS**) a jeho předchůdce, **Secure Sockets Layer** (**SSL**), jsou kryptografické protokoly, poskytující možnost zabezpečené komunikace na Internetu pro služby jako WWW, elektronická pošta, internetový fax a další datové přenosy. Mezi protokoly SSL 3.0 a TLS 1.0 jsou drobné rozdíly, ale v zásadě jsou stejné.

8. ZÁVĚR

Pro kontrolu procesů lze s využitím MySQL databáze a programovacího jazyku PHP využít čtyř uvedených způsobů. Kontrola uživatelského formuláře má však řadu nevýhod, z nichž zásadní je vypnutí JavaScriptu v prohlížeči uživatele a tím vypnutí všech kontrol.

Jak procedury tak triggery jsou omezeny jednak verzí databáze (minimálně verze 5.0), dále je nutné hlídat unikátní názvy procedur a triggerů (může být vždy jen jeden název pro konkrétní databázi). Naproti tomu zavedení uložených funkcí v databázi nevylučuje kombinaci s triggery nebo procedurami a máme v tabulce přehledný seznam funkcí, které lze snadno editovat. Také nejsme omezeni verzí MySQL.

9. SEZNAM POUŽITÝCH ZDROJŮ

[1] Antonín Carda, Renáta Kunstová: *Workflow, Nástroj manažera pro řízení podnikových procesů*, GRADA, Praha 2003

[2] ARTIC STUDIO, Co je to databáze MySQL?
Dostupné z:
<http://www.artic-studio.net/slovnicek-pojmu/databaze-mysql/>

[3] Lhoták Jaroslav. *Pracujeme se session v PHP*
Dostupné z: <http://interval.cz/clanky/pracujeme-se-session-v-php/>

[4] Ing. Petr Opletal. *Workflow - znázornění role a uplatnění*
Dostupné z: <http://www.contros.cz/procesy/workflow.htm>

[5] Petr Zajíc. *MySQL (47) - Triggery a praxe*
Dostupné z: <http://www.linuxsoft.cz/article.php?id_article=1026>

[6] Rozsypal, Petr. *Co je to PHP?*
Dostupné z: <http://php.interval.cz/clanky/co-je-to-php/>

[7] Růžička, Pavel. *Začínáme používat sessions v PHP*
Dostupné z:
<http://interval.cz/clanky/zaciname-pouzivat-sessions-v-php/>

[8] Wikipedie – Otevřená encyklopedie
Dostupné z: <http://cs.wikipedia.org/>

10. SEZNAM PŘÍLOH

CD se zdrojovými daty.