Vysoká škola báňská – Technická univerzita Ostrava

DATABÁZOVÉ A INFORMAČNÍ SYSTÉMY

Učební text

Jana Šarmanová

Ostrava 2007

Recenzoval: Prof. RNDr. Alena Lukasová, CSc.

Název: Databázové a informační systémy
Autor: Jana Šarmanová
Vydání: první, 2007
Počet stran: 122

Studijní materiály pro studijní obor Informační a komunikační technologie
Jazyková korektura: nebyla provedena.

Určeno pro projekt:
Operační program Rozvoj lidských zdrojů
Název: E-learningové prvky pro podporu výuky odborných a technických předmětů
Číslo: CZ.O4.01.3/3.2.15.2/0326
Realizace: VŠB – Technická univerzita Ostrava
Projekt je spolufinancován z prostředků ESF a státního rozpočtu ČR

ISBN 978-80-248-1499-5

POKYNY KE STUDIU

Databázové a informační systémy

Pro předmět Databázové a informační systémy studijního programu „Informační a komunikační technologie“ jste obdrželi studijní balík obsahující

- integrované skriptum pro distanční studium
- sérii animací na CD
- studenti kombinované formy harmonogram průběhu semestru a rozvrh prezenční části s kontaktem na tutora

Prerekvizity

Pro studium tohoto předmětu se předpokládá absolvování předmětu Úvod do softwarového inženýrství, Teorie zpracování dat a kterýkoliv programovací jazyk.

Cílem modulu

je seznámení se základními pojmy používanými při vývoji informačního systému. Důraz je kladen na způsoby identifikace požadavků, analýzu a návrh softwarového díla a specifikaci jeho implementace. Současně jsou teoreticky popsány problémy spojené s konzistencí databáze a víceuživatelským provozem, jejichž znalost je nutná v etapě návrhu implementace. Po prostudování předmětu by měl student být schopen všech činností spjatých s vývojem informačního systému.

Pro koho je modul určen

Modul je sice zařazen do bakalářského studia všech oborů studijního programu „Informační technologie“, ale může jej studovat i zájemce z kteréhokoliv jiného oboru, pokud splňuje požadované prerekvizity.

Při studiu každé kapitoly doporučujeme následující postup:

Skriptum se dělí na části, kapitoly, které odpovídají logickému dělení studované látky, ale nejsou stejně obsáhlé. Předpokládaná doba ke studiu kapitoly se může výrazně lišit, proto jsou velké kapitoly děleny dále na číslované podkapitoly a těm odpovídá níže popsaná struktura.

Čas ke studiu: 1 hodina

Na úvod kapitoly je uveden **čas** potřebný k prostudování látky. Čas je orientační a může vám sloužit jako hrubé vodítko pro rozvržení studia celého předmětu či kapitoly. Někomu se čas může zdát příliš dlouhý, někomu naopak. Jsou studenti, kteří se s problematikou databází ještě nikdy nesetkali a naopak takoví, kteří již v tomto oboru mají bohaté zkušenosti.

Cíl Po prostudování tohoto odstavce budete umět

- popsat …
- uvést příklady z praxe, kdy …

Ihned potom jsou uvedeny cíle, kterých máte dosáhnout po prostudování této kapitoly – konkrétní dovednosti, znalosti.

Výklad

Následuje vlastní výklad studované látky, zavedení nových pojmů, jejich vysvětlení, vše doprovázeno řešenými příklady z praxe. Při výkladu je základní text doplňován tučně vyznačenými důležitými a novými pojmy. Kurzivou jsou psány příklady z praxe, buď v rámci textu nebo jako ucelené řešené příklady samostatně označené.

Shrnutí pojmů

Na závěr kapitoly jsou zopakovány hlavní pojmy, které si v ní máte osvojit. Pokud některému z nich ještě nerozumíte, vraťte se k nim ještě jednou.

Otázky

Pro ověření, že jste dobře a úplně látku kapitoly zvládli, máte k dispozici několik teoretických otázek.

Úlohy k řešení

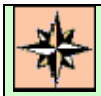
Protože většina teoretických pojmů tohoto předmětu má bezprostřední význam a využití v praxi softwarového inženýra, jsou Vám nakonec předkládány i praktické úlohy k řešení. V nich je hlavní význam kurzu a schopnost aplikovat čerstvě nabyté znalosti při řešení reálných situací hlavním cílem kurzu.

Klíč k řešení

Výsledky zadaných příkladů – stejně jako teoretických otázek výše jsou uvedeny v závěru učebnice v klíči k řešení. Používejte je až po vlastním vyřešení úloh, jen tak si sebekontrolou ověříte, že jste obsah kapitoly skutečně úplně zvládli.

Příprava na tutoriál

Souhrn znalostí nebo vypracovaných úloh, se kterými máte přijít na tutoriál. Mohou to být náměty k diskusím, otázky k promýšlení. Studující se tak mohou připravit na společná setkání a výsledkem je omezení okamžitých improvizací.



Průvodce studiem

V tomto rámečku budou občas napsány pokyny o tom, co je důležité umět, co stačí jen přečíst informativně apod.

Úspěšné a příjemné studium s touto učebnicí Vám přeje autorka kurzu

Jana Šarmanová

OBSAH

1. ŽIVOTNÍ CYKLUS VÝVOJE INFORMAČNÍHO SYSTÉMU

Čas ke studiu kapitoly: 2 x 2 hodiny studium + 2 hodiny řešení úloh

Cíl Po prostudování této kapitoly budete

- vědět, co je systém obecně a v této souvislosti co je informační systém,
- vědět, co je životní cyklus vývoje informačního systému,
- vědět, co všechno patří k zadání informačního systému,
- umět spolupracovat na formulaci zadání informačního systému a zpracovat jeho vnější modely.

Výklad

1.2. Co je to systém

❑ Obecný systém

Vzhledem k tomu, že informační systémy modelují a automatizují jistou část reality (část reality = objekt našeho zájmu ~ systém), zmíníme se nejprve o pojmu systém obecně.

Slovo systém se používá v různých souvislostech. Původně znamenal jen seskupení, sjednocení, celek. Dnes chápeme systém jako seskupení prvků doplněné o vazby mezi nimi, o jejich uspořádanost, jejich strukturu a hierarchii. Je podobný pojmům **organizace či struktura**. Používá se téměř ve všech oborech lidské činnosti.

Příklad

Mnoho příkladů systémů můžeme pozorovat v přírodě: hvězdné systémy, geologické systémy jako hory, pohoří, povodí apod., jednotlivé živočichy, stáda živočichů, národy atd..

♦

S rozvojem poznání se jako systém začaly označovat i jiné zkoumané části reálného světa, které jsou předmětem našeho zájmu. Část reality, kterou chceme zkoumat, může být součástí většího systému a naopak zkoumaný objekt se může skládat z částí, které můžeme chápat opět jako systém. Jednou z důležitých věcí při definování systému jako objektu našeho zájmu je tedy určit **hranice systému**: co je jeho součástí a co je již vně systému; je-li zkoumaný objekt chápán jako podsystém nadřízeného systému, nebo jej zkoumáme jako samostatný celek. Tato hranice zřejmě závisí na pozorovateli.

❑ Dělení systémů podle původu

Z hlediska existence systémů v závislosti na člověku můžeme rozdělit systémy na

- systémy přirozené (*přírodní objekty od buňky po vesmír, systémy fyzikální, živočišné apod.*)
- systémy umělé vytvořené člověkem (*telefonní síť, systém škol, systém zákonů, dětská stavebnice atd.*); informační systémy jsou jedním z typů umělých systémů.

□ Systémový přístup k řešení problémů

Při práci nazveme systémem takový objekt našeho zájmu, který chceme **poznat, popsat nebo vytvořit**.

Existuje obor systémové inženýrství, které se zabývá studiem společných vlastností systémů, jejich analýzou (popisem od celku k detailům) a syntézou (sestavením, vybudováním z dílčích částí). Pro poznání a popis systému se postupně vyvinula řada metod a metodologií. Od náhodných a intuitivních postupů při poznávání reality až po systémové metody výzkumu, od slovních nestrukturovaných popisů až po normovaná pravidla způsobů dokumentace. Souhrn metodologických prostředků používaných při výzkumu a popisu existujících či plánovaných systémů pak nazýváme **systémovou analýzou**. Již dávno je známo, že způsoby zkoumání, popisu a návrhu systémů jsou ve svých základech stejné, ať jde o systémy z naprosto odlišných částí skutečného světa.

Systémový přístup je pak způsob chápání reálného světa, cesta k hledání společných vlastností mezi nejrůznějšími typy systémů jak přirozených, tak umělých.

□ Tvorba umělých systémů

Podobně, jak se člověk postupně naučil vyrábět různé věci, tak se v posledních desetiletích učí zpracovávat informace.

Příklad:

Při výrobě nového stroje si dost dobře nedovedeme představit, že dělník (byť zkušený) vezme kus železa a začne řezat, vrtat, pilovat, frézovat atd., aby vyrobil motor, bez předem zpracovaného profesionálního návrhu, mnoha výpočtů, prototypů, testování.

U programového systému je však bohužel stále ještě běžné, že někdo požádá známého programátora o zpracování programu (nejprve účetnictví, faktur, mezd, skladu, později styk s bankou, pojišťovnou, celnicí atd.), programátor si nechá vysvětlit základy problému, sedne si k počítači a píše program. Dopadne stejně, jak by dopadl dělník s železem a pilníkem. Proplýtvá mnoho času (na rozdíl od dělníka nezničí materiál, což je důvod, proč si není dostatečně vědom všech škod) a výsledek je amatérské nedochůdče. Nic nepomůže, když programátor je skvělý programátor a zná spoustu programovacích jazyků.

♦

Než začneme s výkladem, co je vývoj informačního systému, připomeňme si dávno samozřejmý příklad vývoje systému z jiného technického oboru.

Příklad:

Jeden z historicky nejstarších příkladů promyšleného vývoje systému pochází ze stavebnictví. Úkolem je postavit dům.

Aby byl výsledek úspěšný, opět není možné koupit cihly a začít stavět. Tisíciletími prověřené řešení rozděluje celý projekt do několika etap.

1. *Rozmyslíme a zadáme, jaký dům se má postavit: obytný (nebo jiný), rodinný dům (dvojdomek, činžák, hotel, továrna ...), pro kolik lidí, kde bude stát, kolik máme na něj peněz atd.*
2. *Architekt vypracuje architektonický návrh: celkový vzhled domu, jeho zasazení do okolí, rozdělení domu na patra, místnosti, jejich účel a vzhled atd.*
3. *Stavební inženýři různých profesí vypracují technický projekt: propočtou nosné části, navrhnou vhodné materiály, vypracují detaily řešení stavby, rozvodů vody, elektřiny apod., určí vazby domu na okolí - přívody energie, odvody odpadů atd.*

4. *Řemeslníci provedou dle technické dokumentace realizaci domu. V průběhu stavby provádí stavební dozor kontrolu, kontroluje pořadí a kvalitu provedení, může konzultovat se stavebníkem další detaily atd.*
5. *Do hotového domu se nastěhují lidé, provedou připojení vazeb na okolí (zjistí dopravní spojení, přihlásí adresu, elektřinu, plyn, ...), naučí se zacházet s novým zařízením (topení, bezpečnostní zařízení,..) atd.*
6. *Dům se obývá, udržuje, v případě poruchy opravuje atd..*

♦

Z uvedeného příkladu můžeme zobecnit postup, vhodný pro řešení jakéhokoliv většího úkolu. Zatím jej zformulujeme jen přibližně, dále si tento postup pro vývoj informačního systému pojmenujeme a zformulujeme do přesně definovaných etap.

Při řešení každého většího díla je vhodné dodržovat tyto etapy:

1. formulovat zadání, popsat požadavky na výsledné dílo,
2. analyzovat věcné požadavky detailně, vytvořit modely výsledku,
3. na základě vytvořených modelů popsat způsob technického provedení díla a jeho částí,
4. realizovat dílo podle technického popisu,
5. otestovat správné fungování všech požadovaných funkcí díla,
6. předat dílo zadavateli,
7. používat dílo, případně jej dále udržovat.

Protože informační systém je bezpochyby „větší dílo“, i pro něj bude vysoce vhodné dodržet obdobný postup. Metodologiemi pro tvorbu SW systémů, tedy programových děl většího rozsahu, se zabývá obor **SW inženýrství.**

1.2. Informační systémy

Zopakujme si, co je informační systém. Z předchozího víme, že jde o umělý systém, vytvořený člověkem. Z názvu můžeme usoudit, že jde o systém organizující informace. V minulém semestru v předmětu Teorie zpracování dat jsme si zavedli následující pojmy:

Daty nazýváme údaje získané měřením, pozorováním nebo zaznamenáním z reálné skutečnosti.

Informace jsou smysluplné interpretace dat a vztahů mezi nimi.

Zpracováním dat nebo také hromadným zpracováním dat nazýváme evidenci a zpracování velkého množství údajů (obvykle desítky až stovky) o velkém množství objektů (obvykle od desítek po miliony i víc).

Vést evidenci o objektech znamená

1. zaznamenat vhodně organizované údaje na nějaké médium
2. provádět změny údajů při změně evidované reality
3. provádět výběry informací podle různých kritérií
4. odvozovat a počítat z uložených údajů další
5. třídit údaje dle různých kritérií
6. zaznamenávat vztahy mezi údaji o objektech různých druhů
7. o všech údajích zaznamenaných i odvozených vydávat informace ve vhodné grafické úpravě

Informačním systémem obecně nazýváme organizaci údajů vhodnou pro systémové zpracování dat: pro jejich sběr, uložení a uchování, zpracování, vyhledávání a vydávání informací o nich, to vše pro účely rozhodování.

Množinu datových souborů, uchovávajících data o nějakém uceleném úseku reality, nazýváme **databází.**

Programový systém umožňující definování datových struktur a datových souborů, řešící fyzické uložení dat ve vnější paměti počítače, umožňující manipulaci s daty a formátování vstupních i výstupních informací, nazýváme **systémem řízení báze dat.**

Množina aplikačních úloh nad společnou databází může tvořit ucelený systém, nazývaný **automatizovaným informačním systémem** nad použitým SŘBD. V tomto pojetí tedy databázovým systémem rozumíme celek, řešící rozsáhlejší oblast aplikační, naprogramovaný obvykle v jednom SŘBD s vhodně navrženými datovými strukturami tak, aby všechny aplikační úlohy k nim měly optimální přístup. Řeší uložení, uchování, zpracování a vyhledávání informací a umožňuje jejich formátování do uživatelsky přívětivého tvaru.

V dalším budeme pod pojmem informační systém rozumět vždy automatizovaný informační systém.

Příklad

Neautomatizované informační systémy vidíme v praxi všude kolem sebe: kniha telefonního seznamu, informační tabule na úřadech, papírová kartotéka pacientů u lékaře apod. Každý z nich slouží k organizaci dat tak, aby se v nich dobře data ukládala, vyhledávala, upravovala a doplňovala, to vše pro uživatelovu lepší informovanost a možnost dalšího rozhodování (podle informační tabule najdeme příslušnou kancelář, podle karty pacienta a jejího obsahu se lékař rozhoduje o léčbě apod.)

Automatizované informační systémy také známe: jízdní řád IDOS, mnohé obchody s evidencí svého zboží, firemní informační systémy s řadou podsystémů jako zákazníci a zakázky, sklad zboží, účetnictví apod. Jejich společnou vlastností je realizace na počítačích, lokálních počítačových sítích nebo na internetu, tedy automatizované provádění všech automatizovatelných funkcí systému – kontrola správnosti ukládaných dat, jejich vyhledávání, modifikace, výpočty, třídění, formátování do uživatelsky přívětivého tvaru .

♦

Úkolem tohoto předmětu je naučit se vytvářet automatizované informační systémy – od zadání až po předání uživateli. Protože se budeme zabývat profesionálními systémy obvykle většího rozsahu, bude zřejmě nutné rozdělit opět jeho vývoj do etap, obdobně jako při vývoji jakéhokoliv velkého díla.

1.3. Životní cyklus vývoje informačního systému

Procesem vývoje programových systémů se zabývá SW inženýrství. Celý proces od rozhodnutí o budování systému až po ukončení jeho vývoje, jeho využívání a údržbu nazýváme **životním cyklem vývoje SW systému**.

Informační systémy jsou jedním z druhů počítačových programových systémů a proto pro ně platí vše, co ze SW inženýrství známe. Protože však jde o speciální typ systémů, bude nutné některé etapy životního cyklu probrat mnohem detailněji, případně zavést řadu nových pojmů a technik.

❑ Životní cyklus vývoje IS

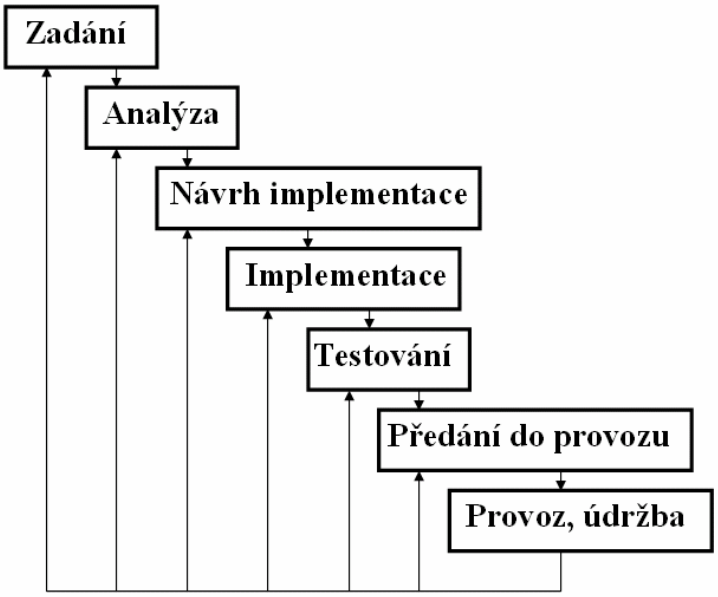
Existuje řada způsobů, jak rozložit vývoj informačního systému do etap, liší se obvykle jen různou mírou podrobnosti či přeléváním některých činností z jedné etapy do druhé. Připomeňme si jeden často uváděný životní cyklus informačního systému, nazývaný obvykle vodopádovým:

1. Zadání - **SPECIFIKACE POŽADAVKŮ**

Etapu zahájí nápad někoho využít pro řešení nějaké evidenční úlohy počítač.

Zadání je vhodné zpracovat písemně do formy "odborného článku", tj. první verze zadání. To provádí osoba znalá oboru, která nemusí nic vědět o počítači. Zadání formuluje slovně. Proto však zadání bývá často nejednoznačné, neúplné, nepřesné. Protože cena za předělání programu po realizaci takového zadání by byla vysoká, je vhodné upřesňovat zadání již v této etapě.

Ideální by bylo použití nějakého formálního jazyka, to však zadavatel obvykle neumí. Proto se na této úrovni dělají kompromisy ve formě zpracování zadání. Používají se jednoduché formální metody, zadavateli přirozeně srozumitelné.



2. Analýza - **SPECIFIKACE PROBLÉMU**

Etapa znamená převod zadání do několika modelů budoucího systému, které mohou sloužit jako podklad pro zahájení návrhu řešení a pro implementaci. Etapu nazýváme obvykle analýzou problému nebo specifikací problému. Závěrem se provádí návrh ovládání programu a jeho uživatelského vzhledu. Pro analýzu je navržena řada metod, ty budou hlavní náplní následujících kapitol. Vhodné je použít i prototypového řešení systému, zvláště pokud máme některý prostředek automatizovaného vygenerování aplikace.

Výsledkem analýzy může být i nedoporučení realizace.

3. Návrh - **NÁVRH IMPLEMENTACE**

Po ukončení specifikace (prototyp je doladěn nebo zadavatel souhlasí s formulací specifikace) se zahájí návrh implementace. Doplňují se detaily funkcí, systémové funkce, bere se v úvahu budoucí HW a SW apod. Výsledkem jsou zpřesněné datové struktury, zpřesněné a doplněné algoritmy funkcí systému o systémové části. Vhodný je i návrh dekompozice řešení na malé moduly.

4. Programování - **IMPLEMENTACE A DOKUMENTACE**, programování v malém

Po předání návrhu modulů programátorům se implementuje systém po dílčích částech, ladí se jednotlivé funkce, moduly, subsystémy a celý systém. Zpracovává se dokumentace programu: příručky uživatele, příručky programové.

5. Testování - **TESTOVÁNÍ SYSTÉMU**

   Testování, zda je systém bezchybný a zda se shoduje se zadáním, se specifikací a s dokumentací.

6. Předání - **ZAVEDENÍ SYSTÉMU DO PROVOZU**

   Systém otestovaný na zkušebních datech ve zkušebním provozu se po odstranění chyb předá uživateli. To znamená přechod uživatele na nový způsob práce, zaškolení uživatele, napojení na okolí systému, na vstupy (na informace vstupující z okolí do systému) a výstupy (kam směřují informace vystupující ze systému).

7. Provoz - **PROVOZ, ÚDRŽBA A ROZVOJ**

   Po zaběhnutí provozu se dále sleduje provoz, provádí se opravy chyb u programu i dokumentace, provádí se údržba, registrují připomínky, návrhy na zlepšení, doplnění ap.

V dalších kapitolách projdeme jednotlivé etapy podrobněji.

1.5. Zadání informačního systému

Na začátku existence informačního systému je rozhodnutí nějaké osoby využít počítače či počítačové sítě pro řešení nějakého reálného evidenčního problému. Této osobě budeme říkat zadavatel a poněkud zjednodušeně řekneme, že zadavatel formuluje požadavky na budoucí dílo.

Zadání je vhodné požadovat od zadavatele písemně, formou tzv. odborného článku, první verze písemného zadání. Zadavatel sice zná věcnou stránku zadání, ale často neví nic o programování, analýze a formálních specifikacích. Proto zadání obvykle formuluje slovně a proto zadání bývá často nejednoznačné, neúplné, nepřesné. Cena za předělání programu po realizaci takového zadání by byla vysoká (*jako cena přestavby domu při změně požadavků na jeho funkce*), proto je nutné upřesňovat zadání již v této etapě. Při upřesnění zadání musí spolupracovat manažer projektu, zkušený analytik.

Ideálem je **přesná, úplná a bezesporná specifikace**. Používají se různé metody pro zadavatele jednoduché a srozumitelné. Pomocí nich se zpracovávají modely budoucího IS, upřesňující zadání.

Požadavky uživatelů z věcného hlediska dělíme obvykle na funkční a nefunkční a je vhodné s tímto dělením předem seznámit zadavatele. Pokud nejsou dostatečně v zadání jednotlivé části formulovány, je nutné se na ně doptat.

1.4.1 Funkční požadavky

Funkčními požadavky rozumíme požadavky na **věcný, problémový obsah** systému. Zkušený analytik umí již při studiu zadání rozpoznat jeho neúplnosti nebo rozpory. Pak formou diskuze se zadavatelem musí tyto problémové části dořešit. Je vhodné, když si vypracuje jakousi **šablonu otázek**, podle níž se systematicky ptá, případně požádá zadavatele o dodržení úplnosti těchto informací. Má tak větší jistotu, že mu zadavatel uvede všechny informace o zadávaném IS.

Společně vytvořené modely jsou vhodné jako prostředek komunikace analytika se zadavatelem pro upřesňování a zpodrobňování zadání. Analytik tak bude mít dobrý základ pro následnou analýzu.

Základní struktura takové šablony je: **proč, k čemu, kdo, vstupy, výstupy, funkce, okolí**.

Proberme si jednotlivá hesla podrobněji.

❑ PROČ vůbec je zapotřebí nový informační systém

Zdánlivě divná otázka obvykle uvede zadavatele do rozpaků. Úkolem je popsat okolnosti rozhodnutí o budování IS: popsat současný stav evidence, stručně vysvětlit, proč tento stav nevyhovuje, charakterizovat nové potřeby a představy o tom, jak má nový systém fungovat.

Příklady:

1. *Zatím je vedena evidence ručně, jde o mnoho údajů, je nutná automatizace;*

2. *Dosavadní SW pro evidence již nestačí, nemá všechny potřebné funkce, neeviduje všechny informace;*

3. *V současnosti existuje několik vzájemně nespolupracujících programů, je potřeba jejich funkce propojit apod.*

♦

Může se i stát, že si zadavatel uvědomí, že nový systém není zapotřebí.

❑ K ČEMU má systém sloužit

Opět zdánlivě divná otázka má dát odpověď na to, co je hlavní prioritou pro funkci systému: vnitřní evidence, vnější reprezentace informací apod.

Příklady:

1. Firemní IS má sloužit administrativě firmy, vést všechny potřebné evidence: zákazníky a zakázky, sklad zboží, majetek, účetnictví, zaměstnance, mzdy apod. Hlavním účelem tedy je mít pořádek ve vlastní vnitřní evidenci.

2. *Jiný typ firemního IS má sloužit široké veřejnosti jako internetový obchod: nabízet zboží ze skladu a umožnit objednávání do nákupního košíku, vést objednávky, jejich stav a vyřizování prodeje, vést evidenci o platbách zákazníků, o množství zboží na skladě apod. Hlavním cílem je co nejlépe prezentovat firmu a její nabídku navenek.*

3. *Ještě jiný firemní systém má sloužit managementu firmy pro analýzy prodeje a rozhodování o strategii dalšího rozvoje firmy: které zboží, ve kterém čase, ve které oblasti se prodává hodně nebo málo, jací zákazníci nakupují které zboží apod. To vše může sloužit k lepšímu zacílení reklamy, cílených nabídek některým skupinám zákazníků atd. Hlavním cílem jsou analýzy umožňující fundovaná a ekonomická rozhodnutí pro vedení firmy.*

♦

Samozřejmě jeden systém může mít více priorit. Odpověď zadavatele pomáhá i jemu ujasnit si jejich existenci a pořadí. Tím odpovídá na otázku, které funkce bude nutné optimalizovat a které případně budou poněkud potlačeny - z hlediska rychlosti odezvy systému, z hlediska snadného ovládání systému, z hlediska bezpečnosti dat, bezpečnosti přístupu k funkcím apod.

❑ KDO s ním bude pracovat - běžně, příležitostně, zřídka

Otázka souvisí bezprostředně s předcházející odpovědí: primární funkce systému budou sloužit hlavním uživatelům systému s nejčastějším přístupem. Případné další funkce mohou sloužit občasným uživatelům nebo i náhodným dotazům na informace z databáze.

Příklady:

1. IS pro administrativu firmy znamená zaškolení menšího počtu uživatelů, kteří dobře znají svá data, své úkoly a IS jim pomáhá v jejich práci. Uživatelské prostředí zvládnou během zaškolení,

mohou provádět i složité funkce. Jsou lépe schopni formulovat své připomínky, nové požadavky, rozeznávat chyby.

2. *Internetový obchod používají nejrůznější cizí uživatelé, od nichž se lze nadít čehokoliv. Systém musí mít velmi jednoduché, stručné a intuitivní ovládání, jinak mu zákazníci odejdou jinam.*

3. *Analytický systém slouží managementu, který nejspíš nezná strukturu databáze, neví, kde jsou které informace uloženy. Analýzy ale potřebuje rychle a pohodlně zadávat vstupní požadavky, výstupy formou přehledných tabulek nebo grafů.*

♦

❑ Jaké budou VSTUPY do systému

Vstupy znamenají informace, které mají být v IS evidované. Je to seznam entit a jejich atributů. Jsou základem pro datovou analýzu, proto mají obsahovat skutečně podrobný výčet. Protože analytik nemusí být odborníkem v oblasti, pro niž je IS budován, jsou vhodné i komentáře vysvětlující odborné pojmy nebo zdůvodnění speciálních potřeb.

Příklad:

IS veřejné knihovny potřebuje evidovat: knihy (název knihy, autory, ISBN, přírůstkové číslo, vydavatele, rok vydání), čtenáře (jméno, adresa, telefon, číslo průkazu), výpůjčky knih (datum výpůjčky, datum vrácení, číslo čtenáře, přírůstkové číslo knihy).

Přírůstkové číslo je jednoznačné v rámci celé knihovny pro každý exemplář knihy, ISBN je číslo titulu pro jedno vydání knihy.

♦

❑ Jaké budou VÝSTUPY ze systému

Výstupy znamenají všechny výstupní sestavy, které budou uživatelé potřebovat. Stačí název sestav a seznam informací na nich, lepší jsou načrtnuté ukázky nebo existující sestavy.

Příklad:

IS veřejné knihovny bude potřebovat: inventurní seznam knih (název knihy, autoři, ISBN, přírůstkové číslo), výpůjční lístek (jméno čtenáře, datum výpůjčky, název, autoři, ISBN, přírůstkové číslo), upomínku nevrácené výpůjčky (jméno a e-mailovou adresu čtenáře, název knihy, přírůstkové číslo) atd.

♦

Tyto informace jsou jednak určeny k analýze výstupních funkcí, jednak jako kontrola úplnosti zadaných vstupů. Může se totiž stát, že zadavatel požaduje na výstupu informace, které na vstupu nezadal, ani se nedají ze vstupů odvodit. Ty je pak nutné buď doplnit do vstupů, nebo zrušit požadované výstupy.

❑ Jaké FUNKCE bude systém plnit

Tyto informace jsou jednak určeny k zadání všech potřebných funkcí IS, jednak jako kontrola úplnosti zadaných vstupů. Může se totiž stát, že zadavatel požaduje na výstupu informace, které na vstupu nezadal, ani se nedají ze vstupů odvodit.

I tyto informace jsou podkladem pro funkční analýzu. I zde slouží jako kontrola úplnosti zadaných vstupů. Opět se může stát, že zadavatel nezadal některé vstupní informace, které jsou pro správné fungování funkcí nezbytné. Ty pak analytik doplní do vstupů. Někdy se ukáže potřeba doplnit pomocné další atributy, které budou sloužit správné funkčnosti.

Příklad:

IS veřejné knihovny musí mít funkce: záznam nové knihy, inventurní seznam knih, záznam čtenáře a modifikace informací o něm, záznam výpůjčky a vrácení knihy, záznam o ztrátě nebo zničení knihy, kontrola včas nevrácených výpůjček, možnost prodloužení výpůjčky atd.

U knih nebyl zadán atribut umožňující zaznamenat ztrátu nebo zničení knihy. Analytik tedy přidá atribut „stav" ke knize, v něm může být jedna z hodnot nevypůjčeno, vypůjčeno, zničeno, ztraceno atd. Podobně zatím není možno zaznamenat prodloužení výpůjčky (původně se předpokládala výpůjčka např. na 28 dnů). Přidá se tedy atribut termín vrácení, v něm bude předepsaný den vrácení. Datum vrácení pak bude buď NULL (= dosud nevráceno) nebo skutečné datum vrácení. Při prodloužení výpůjčky se termín vrácení změní na nové datum.

♦

Jako pomocný nástroj pro zadání všech funkcí je vhodné zadavateli doporučit tzv. **seznam událostí a reakcí systému.** Model má za úkol získat od zadavatele seznam všech funkcí, které bude od IS požadovat. Protože IS odráží realitu, je i každá funkce IS reakcí na nějakou událost.

Sestaví se tabulka se sloupci událost a reakce, do ní se formou krátkých textů zapisují všechny možné vnější události, podněty působící na systém a jim odpovídající reakce systému.

Události se někdy dále dělí na

F – událost (flow oriented): vstup dat do systému, ne odvozené vstupy (nová objednávka, ...); v kontextovém diagramu jsou zobrazeny jako datový tok,

T – událost (temporar): časová událost řízená vnitřními hodinami systému, nenese data (denně ve 24.00, konec měsíce, ...); nezávisí na vstupu dat nebo na nějakém povelu,

C – událost (control): zvláštní případy, výjimečné události na pokyn zvenčí, událost nenese data ani čas (alarm, zablokuj dveře, ...).

Příklad

Události a reakce IS Knihovna

UDÁLOST	REAKCE SYSTÉMU	Aktér
nový čtenář	zapiš do seznamu čtenářů	
výpůjčka knihy	zapiš výpůjčku	
vrácení knihy	zapiš vrácení	
nová kniha	. . .	
ztracená kniha	. . .	
konec roku	. . .	
. . .	. . .	

♦

❑ Jaké je OKOLÍ systému

Okolí systému znamená definovat všechny objekty (budeme je nazývat aktéry), kteří jsou zdrojem informací plynoucích do systému nebo cílem informací ze systému. Aktéry mohou být lidé, instituce nebo jiné SW systémy. Aktér není úplně totéž co uživatel.

Příklad:

IS veřejné knihovny má jako zdroj informací například nakladatele nebo knihkupce, od nichž kupuje knihy nebo alespoň bere nabídky na knihy (přitom mohou být vzdáleni a nikdy nepracovat

s IS knihovny). Dalším aktérem jsou čtenáři, kteří dodají informace o sobě při přihlášení se za čtenáře, dávají informace, které knihy si půjčují nebo vracejí. Jiným aktérem je knihovník, kterého zajímají nabídky knih, počty výpůjček jednotlivých titulů, inventurní seznamy apod.

Má-li systém možnost přímého přístupu čtenáře k nabízeným knihám, jsou uživateli systému jednak knihovník, jednak čtenář. Ovšem nakladatel není uživatelem, ale aktérem je.

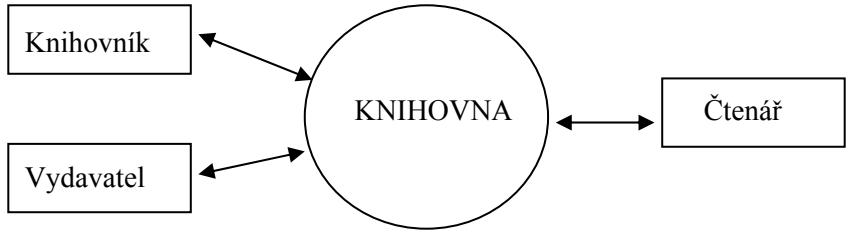
♦

Standardním nástrojem pro grafické zobrazení systému a jeho okolí je **kontextový diagram.** Systém se chápe jako černá skřínka, neví se nic o jeho vnitřní struktuře.

Základní prvky kontextového diagramu jsou: obdélníkový uzel zobrazuje aktéra = zdroj nebo cíl informací pro IS, kruhový uzel (bublina) zobrazuje celý systém jako proces. Orientované hrany znázorňují datové toky, dodávání a odebírání informací ze systému. Graf celkově strukturuje systém a jeho okolí.

Příklad

Kontextový diagram systému Knihovna



Aktér Vydavatel dává informace o svých nabídkách knih a dodaných knihách, odebírá informace o objednaných knihách. Čtenář dává systému informace o sobě a potom o tom, které knihy si půjčuje nebo vrací. Jsou mu určeny informace – upomínky včas nevrácených knih, případně nabídky knih pro vypůjčení. Knihovníka zajímají například informace souhrnné o počtech výpůjček nebo inventurní seznamy apod., dodává systému například informace o zařazení knih do žánrů, o umístění knih v regálech apod.

♦

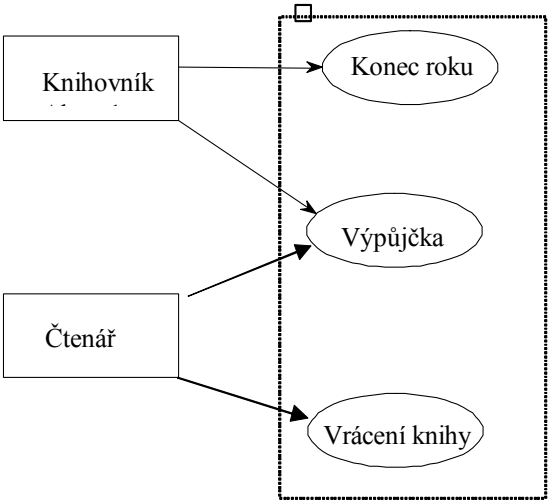
Model jednání

Kombinací seznamu funkcí a modelu okolí je tzv. **model jednání**. Tento model vlastně propojuje informace kontextového diagramu se seznamem událostí a reakcí, přiděluje události jednotlivým aktérům.

Slouží zase k oddělení systému od okolí a k jemnější strukturalizaci okolí. (a také se v rámci modelu jednání o tuto vnitřní strukturu nezajímáme). Grafické zobrazení viz obrázek.

Příklad

Část modelu jednání systému Knihovna



Všimněme si, že zde není nakladatel, protože ten nepracuje přímo se systémem. Přesto je aktérem (viz kontextový diagram), protože odebírá a dodává data do systému.

♦

Aktér je část okolí systému, která komunikuje s vytvářeným systémem. Aktér není uživatel, aktér je role uživatele. Jeden člověk může působit (postupně) jako několik aktérů, aktérem může být člověk nebo jiný systém. Aktéři se mohou například lišit právy (administrátor systému je jiná role než prostý uživatel), nebo způsobem zacházení s daty (uživatel vkládající data je jiný aktér, než uživatel, který si data jen prohlíží). Graficky aktéra značíme obdélníkem.

Typ jednání je kompaktní popis komunikace mezi aktérem a vytvářeným systémem. Je to způsob použití systému, který využívá část jeho funkcí. Je znázorněn oválným uzlem.
Typ jednání určuje funkce modelu, tj. pomocí jejich popisu se definují funkční požadavky na vytvářený systém. Jeden aktér může vést několik typů jednání, jeden typ jednání může být použitelný několika aktéry.

Typ jednání znázorňuje posloupnost stavů viděnou zvenčí. Při jeho tvorbě se vžíváme do role aktéra a měli bychom popsat

- jaký je hlavní úkol aktéra
- zda aktér čte / zapisuje / mění data
- zda aktér informuje systém o změnách vně systému *(ukládá, modifikuje, ruší data)*
- zda má být aktér informován o změnách uvnitř systému *(prošel termín vrácení knihy, ...)*.

Model jednání je souhrn všech typů jednání, zakreslených grafem.

Podstatou modelu jednání nejsou nakreslené obdélníky a ovály, jak je ukazují obrázky, ty poskytují celkový přehled, **podstatou modelu je rozčlenění uživatelů** do skupin, rozlišení typů komunikace mezi uživateli a systémem.

Jiným způsobem můžeme například přiřadit role aktérům doplněním tabulky událostí – reakce o sloupec aktér, případně ji podle aktérů setřídit.

1.4.2 Nefunkční požadavky

Mimo věcnou náplň systému je nutné v zadání uvést řadu dalších informací o okolnostech řešení. Tyto další požadavky, zvané nefunkční, jsou omezení kladená na systémové služby. Jsou několika typů:

1. Požadavky na priority **výsledného programu**: efektivita (rychlost řešení, výkon = rychlost odezvy IS, paměťová náročnost), spolehlivost, přenositelnost do jiných SW prostředí, variabilita řešení pro různé uživatele.
2. Požadavky na **způsob řešení**: mohou předepisovat metody a použití standardů z oblasti metodiky vývoje, dokumentace, programovací jazyk, programovací a uživatelské prostředí. Celkem podmínky dodání, implementační požadavky a použití standardů.
3. **Vnější požadavky**: ostatní nefunkční požadavky, jako cenová omezení, doba řešení a harmonogram, podmínky dodání, dodržení firemních standardů, omezení daná legislativou (platné zákony, vyhlášky), požadavky na spolupráci nového systému s již existujícími systémy, daná organizační struktura firmy, bezpečnost, atesty apod.

Tyto požadavky se nevyužijí při analýze systému (mimo případně předepsanou metodiku pro analýzu), tam se řeší jen věcná náplň. Nefunkční požadavky se zohledňují až v etapě návrhu implementace nebo při uzavírání smlouvy.

Příklad

Knihovní IS pro rozsáhlou veřejnou knihovnu s beletrií i odbornou literaturou požaduje:

1. *Není nutné co nejrychlejší ukončení IS, nejdůležitější je **rychlost odezvy** systému při vyhledávání publikací podle různých kritérií, export a import dat při spolupráci s jinými knihovními systémy, dále také bezpečnost evidovaných dat (zálohování).*
2. *Nejsou požadavky na metodiku řešení ani implementační prostředí, pokud budou dodrženy **standardní způsoby ovládání** systému.*
3. ***Cena IS** max. xxx Kč. **Harmonogram řešení** – požadujeme nejprve realizovat a předat všechny vstupní funkce (knihovníci mohou ukládat data již současně s dalším vývojem systému). **Spolupráce** IS s knihovními systémy ABC a XYZ. Dodržení knihovního zákona.*

♦

Řešený příklad - Projekt – Zadání

Jako vzorový příklad projektu nás bude celým semestrem provázet již v dílčích příkladech použitý informační systém veřejné knihovny. Pravděpodobně každý se setkal s nějakou skutečnou knihovnou a má základní představu o jejím provozu. Další možná netušené podrobnosti získáme ze zadání nebo z odpovědí na průběžné doplňující dotazy od zadavatele. Na závěr každé kapitoly nebo odstavce budeme realizovat příslušnou etapu vývoje projektu nebo její část.

Zadavatelem obecně budeme nazývat vedoucího knihovny nebo kteréhokoliv dalšího pracovníka knihovny. Řešitelem nazýváme buď manažera projektu nebo kteréhokoli dalšího spolupracovníka (analytika, programátora apod.).

Zadání a celé řešení je vymyšleno, případná podobnost s konkrétní knihovnou je zcela náhodná. Ovšem situace při spolupráci s uživateli nejsou zcela vymyšleny.

Veřejná knihovna obce Velká Lhota

Zadání od zadavatele (vyžádáno v písemné formě)

V knihovně máme na počítači evidenci všech knih a časopisů. Potřebujeme lepší vyhledávací systém, který by uměl psát upomínky čtenářům, kteří nevrátili knihy včas. Také je třeba provádět 2 x ročně inventury. Když někdo ztratí nebo zničí knihu, musí ji zaplatit a my ji musíme zrušit z evidence.

Časopisy se půjčují jen na čtení v čítárně, domů ne. Ale musí se zaznamenat, kdo ho má půjčený, protože někteří lidé časopis nevrátí a odnesou.

Doplňující otázky řešitele (neboť zadání je zřejmě neúplné) **a odpovědi zadavatele** (zapsal v zestručněné a upřesněné podobě řešitel):

PROČ = současný stav, upřesnění potřeb:

1. Jak je vedena současná evidence?	V MS Wordu v tabulkách, tam se přepisuje, **kdo** si naposledy knihu nebo časopis půjčil a **kdy**. Pro lidi máme i zásuvkové katalogy podle názvů a autorů.
2. Co všechno se tam eviduje o knize?	**Název, autor, vydavatel, rok vydání, cena, ISBN, přírůstkové číslo.**
3. Kolik autorů evidujete?	Jeden, ale bylo by dobré evidovat **všechny**.
4. Co je přírůstkové číslo?	Každá kniha má své jednoznačné číslo, každý exemplář jiné, přidělované číselnou řadou.
5. Je potřeba evidovat ještě něco o knize?	Kdyby to moc nevadilo, tak i kdo si půjčil knihy dříve kvůli měsíčním statistikám.
6. Co jsou měsíční statistiky?	Podle počtu vypůjčených knih za měsíc dostáváme odměny k platu. Musíme spočítat zvlášť knihy a zvlášť časopisy. Knihy, které si dlouho nikdo nepůjčuje musíme nabízet nebo vyřadit.
7. Co je dlouho a jak je nabízíte?	Asi půl roku a potom je postavíme na zvláštní polici s nápisem Naše nabídka.
8. Ještě něco potřebujete o knihách?	Někde mají i **žánr** knihy a **anotaci,** kdyby to také šlo.
9. Čtenáře neevidujete?	Ano, oni mají průkazky a my máme knihu s jejich **adresami.**
10. Co všechno potřebujete o čtenáři vědět?	Dobré by bylo znát **telefon (mobil) a e-mail**.
11. Jak evidujete výpůjčky?	Na předtištěné papíry vypíšeme název knihy a přírůstkové číslo, čtenář to podepíše. Papírky skládáme podle abecedy do katalogu, každý měsíc zvlášť, aby se rychleji hledaly nevrácené knihy.
12. Na jak dlouho půjčujete?	Na 14 dnů, ale když zavolají, mohou si to prodloužit o dalších 14 dnů.
13. Dáte nám vzor toho výpůjčního papíru. Jak vypadá váš inventurní seznam?	Zvlášť knihy vypůjčené, zvlášť nevypůjčené, dáme vám také ukázku obou seznamů.
14. Ještě nám ukažte měsíční statistiku výpůjček. Jak řešíte ztracené knihy?	Podle stáří knihy a její ceny se určí pokuta čtenáři a potom se kniha vyřadí z evidence. Pokuta se vybírá do pokladny, z těch peněz se platí opravy knih.
15. Jak nakupujete knihy?	Dostáváme nabídky od mnoha vydavatelů, teď už skoro výhradně e-mailem upozornění na novou nabídku na webu – asi jednou týdně. Tam si vybereme a posíláme přes internet objednávky. Když knihy přijdou zásilkovou službou, zapisujeme je do seznamu knih. Také je vystavíme do police Nové knihy. Některé nabídky chodí v katalogu, ty objednáváme poštou.
...	
x. A o časopisech?	
...	

K ČEMU a KDO

1. K čemu budete systém používat? Nerozumím otázce, přece k seznamu knih a výpůjček.
Budete nabízet služby systému čtenářům? Nevím jak.
Třeba bude jedno PC sloužit pro výběr knih. Ale nepokazí nám ti čtenáři něco v evidenci?
Ne, budou smět jen prohlížet knihy a vybírat. Tak dobře, aspoň nebudou chodit mezi knihami a nemusíme je hlídat.
2. Máte mezi sebou – knihovníky rozděleny práce a kompetence?
Ano, vedoucí píše objednávky, kontroluje správný záznam nových knih a z měsíčních statistik navrhuje odměny. 3 knihovníci vypůjčují knihy a píší upomínky. Jedna specializovaná knihovnice zařazuje nové knihy do oddílů.
3. Co je oddíl? Knihy se dělí podle žánrů, jazyka (máme i cizojazyčné knihy), doby vzniku a tak a také se podle toho rozmísťují do polic.
4. Máte ty oddíly, žánry atd. někde zapsány? Ne, to známe zpaměti.
Takže je budeme také evidovat, uveďte, co všechno má vliv na to rozdělení.
Žánr, jazyk, země vydavatele, rok vydání. Základní rozdělení je podle jazyků a zemí, potom podle žánrů. Různá vydání ve stejném jazyce řadíme za sebe. Do knihy se zapíše **signatura** = označení umístění.
5. Mohou si čtenáři knihu objednat předem? Ne, na to nemáme zatím čas. Ale zájem je.
6. Předpokládáte časem i provoz na webu? To by bylo moc složité a stejně si čtenáři musí pro knihy přijít osobně, po internetu jim je nepošleme.

VSTUPY = co všechno bude uživatel do databáze ukládat
Shrňme si tedy:

Čtenáře - jméno, adresa, telefon, e-mail, číslo průkazu
Knihy - název, autoři, vydavatel, rok vydání, cena, ISBN, prir.cislo, žánr, jazyk, země, anotace, signatura
Vypůjčené knihy – která kniha, kterým čtenářem, odkdy, dokdy ji má půjčenu
Rezervované knihy – čtenář, kniha, datum rezervace, splněno
Objednávky knih – název, vydavatel, id_objednací_číslo, počet

Je to vše?
Asi ano – ještě ty upomínky.
Ano, k výpůjčkám přidáme atribut datum_vrácení a upomínka.
Posíláte více upomínek? Ano, tři – vždy po týdnu, potom posíláme pokutu.
Takže upomínka bude znamenat počet poslaných upomínek.
Každý soubor bude mít vlastní vstupní formulář.

VÝSTUPY = výpisy na papír nebo na obrazovku
Shrňme si dále, jaké jsou výstupy:

Výpůjční lístek pro podpis čtenáře,
měsíční statistika,
inventura a její 2 seznamy knih,
informace čtenáři o seznamu vybraných knih podle zvolených požadavků, požadavky podle všech atributů.

Ještě něco? Asi to je vše. Dodány jsou předlohy - tiskopisy.

FUNKCE = co všechno má systém umět
Dále si zopakujeme seznam funkcí, jak jsme o nich už mluvili (pro stručnost zde píšeme na jednom řádku několik souvisejících funkcí):

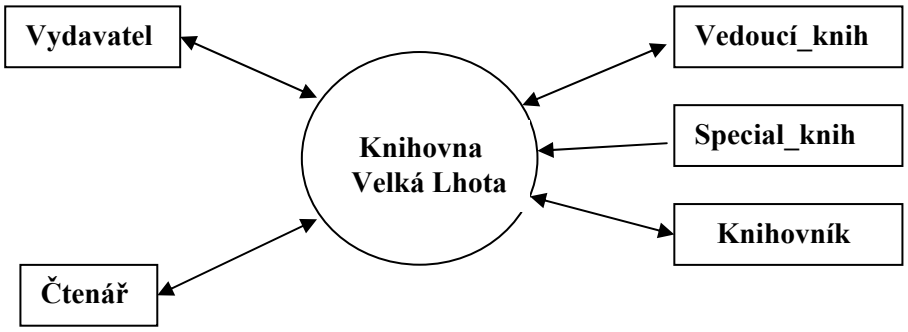
Nový čtenář, změny čtenářů. Místo zrušení bude archivace kvůli pozdějším statistikám.

Nová kniha, doplnění jejího zařazení do oddílu,
Zrušení knihy z evidence do archivu kvůli ztrátě, zničení nebo vyřazení.
Záznam výpůjčky a vrácení knihy, tisk výpůjčního lístku, tisk upomínek, pokuty.
Záznam nové objednávky pro vydavatele nebo knihkupce.
Výpočet měsíční statistiky výpůjček.
Provedení a tisk inventurních seznamů.
Výběr knih nevypůjčených alespoň 6 měsíců.
Možnost výběru knih pro čtenáře podle názvu, autora, žánru, vydavatele, jazyka, roku.

Je to vše?

OKOLÍ

Závěrem si namalujeme okolí systému a pojmenujeme jeho aktéry – role.



Nefunkční požadavky

Ještě nejsme hotovi, musíme projednat další podmínky řešení.

Zjistíme současné vybavení HW a SW, internetem *(mají 3 PC s internetem, ale „databáze" ve Wordu je na jediném PC)*. Zjistíme hustotu provozu – počet čtenářů denně *(kolem 200),* kolik je obsluhuje knihovníků současně *(1-3 podle potřeby),* kolik dalších pracovníků pracuje se systémem *(1-2, vedoucí a specialista)*.

Zjistíme ochotu zakoupit server pro databázi společně přístupnou ze všech stanic, případně dokoupit 2 PC pro každého pracovníka (ano).

Zjistíme, jestli má zadavatel nějaké požadavky na použitý SW *(ne)*, plynoucí z legislativy *(ne, jen pravidelné výpisy svého knižního fondu spřáteleným knihovnám)* ... je nutno doplnit do funkcí.

Dohodneme harmonogram *(týden na analýzu a její odsouhlasení, další měsíc na implementaci, potom předání ke zkušebnímu provozu, po dalším měsíci spuštění provozu nového IS),* cenu.

Závěr

Máme doplněné a upřesněné zadání, které již bude lepším základem pro všechny typy analýz.

♦

Shrnutí pojmů 1.

Systém, systémová analýza, systémový přístup k řešení projektů.
Informační systém, data, databáze, evidence dat objektech a její základní úlohy.
Životní cyklus vývoje IS a jeho etapy.
Zadání informačního systému, funkční a nefunkční požadavky, modely vnějšího chování IS.
Kontextový diagram, seznam událostí a reakcí systému, model jednání.

Otázky 1.

1. Co je obecný systém?
2. Jak se systémy dělí podle původu a vztahu k člověku?
3. Jaký je nejvhodnější postup při budování jakéhokoliv většího systému?
4. Co je informační systém?
5. Jak se dělí požadavky pro zadání IS?
6. Co jsou funkční požadavky na IS a které informace obsahují?
7. Jaké formální modely se při formulaci zadání používají?
8. Co jsou nefunkční požadavky na IS a které informace obsahují?

Úlohy k řešení 1.

Z následujících úloh si vyberte jednu a vytvořte pro ně seznam funkčních a nefunkčních požadavků a modely vnějšího chování - kontextový diagram, seznam událostí a reakcí, model jednání. Pokud to je zapotřebí, sestavte doplňující otázky pro zadavatele.

1. Firma potřebuje evidenci svých **zaměstnanců**, kde někteří pracují doma a výsledky práce firmě odevzdávají. Určete, které údaje se musí uchovávat, aby se mohly realizovat následující činnosti:
   - podle zastávané funkce jim vyplácet pevnou mzdu a posílat ji na jejich účet v bance,
   - měsíčně počítat počet zaměstnanců, vyplacenou sumu na mzdy, minimální, maximální a průměrnou mzdu,
   - posílat odvody z mezd sociální a zdravotní pojišťovně, zdravotní pojišťovnu může mít každý zaměstnanec jinou,
   - podle jazykových znalostí jim zadávat práci pro zahraniční zákazníky,
   - podle vzdělání je posílat na specializovaná odborná jednání se zákazníky.
2. Je potřeba evidovat rezervace a využití počítačů na **počítačových učebnách** studenty. Ve škole je k dispozici studentům několik počítačových učeben s různým počtem počítačů 24 hodin denně. Studenti si mohou rezervovat místo u konkrétního stroje vždy na 1h denně, počínaje celou hodinou. Každý počítač je jednoznačně identifikovatelný. V každé chvíli musí být jednoznačně určeno, zda lze provést rezervaci na daný stroj v danou dobu, zjistit, na které učebně a u kterého stroje byl, je nebo bude konkrétní student. Dále je třeba provádět statistiky využití učeben a počítačů studenty z jednotlivých ročníků a oborů včetně hodin nevyužitých.
3. Dětský lékař potřebuje evidenci o **očkování** svých pacientů. O dětech jméno, RČ, datum narození, adresu, datum, typ (proti čemu bylo dítě očkováno) a popis očkování, o typech očkování navíc, ve kterém měsíci života dítěte se očkování provádí. Pokud se totéž očkování opakuje v různém věku, je zapsáno znovu. Je potřeba zabezpečit objednávání pacientů v měsíci odpovídajícím očkování na volný termín u lékaře, záznam o realizovaném očkování, o zrušení objednávky, o objednání na náhradní volný termín. Jedno očkování trvá 10 minut. Dále je zapotřebí poslat výkazy o provedených očkováních příslušné zdravotní pojišťovně (datum, typ očkování, RČ pacienta) a sledovat, zda jsou tyto výkony lékaře od pojišťovny již zaplaceny. Konečně je potřeba vést statistiky o očkováních podle typu očkování a podle věku dětí.
4. Navrhněte informační systém žáků pro výběrovou **základní školu** s dělenou výukou. Evidují se jednotlivé třídy žáků, vyučované předměty, žáci a individuální rozvrh (kdy, který žák navštěvuje jaký předmět, přičemž každý žák má svůj rozvrh, který nemusí být identický s ostatními spolužáky). U žáků se eviduje rodné číslo, jméno, příjmení, studijní průměr, třída do které patří,

přičemž každý žák patří do nějaké třídy. Třídy žáků jsou určeny svým názvem (číslo ročníku + písmeno skupiny, například 5A), specializací (která nemusí být uvedena, například matematická), třídním učitelem. Ne všechny třídy musí být obsazeny žáky. Dále máme seznam předmětů obsahující číslo předmětu (jednoznačná zkratka názvu předmětu max. 5 znaků + ročník ve kterém se předmět vyučuje), název předmětu a odpovědný učitel; předmět nemusí být vyučován. V rozvrhu pro konkrétního žáka sledujeme číslo navštěvovaného předmětu, den a hodinu v rozvrhu a číslo místnosti, kde se vyučuje. Jeden předmět se může vyučovat pro stejného žáka více hodin týdně, v různém dni a čase v různých místnostech.

5. Navrhněte informační systém **autoservisu**. V autoservisu existuje seznam automobilů, které byly opravovány s informacemi SPZ auta, typ auta (jako text popisující model karoserie a motoru), číslo karoserie, číslo motoru, rok výroby. Každé auto v seznamu už bylo servisováno. Dále informační systém obsahuje katalog součástek, které se dají objednat pro opravu. Seznam obsahuje číslo součástky (jednoznačné číslo z katalogu), název součástky, typ součástky (základní rozdělení zda se jedná o součástku pro karosáře, elektrikáře, lakýrníky, motoráře), cenu součástky; ne každá součástka musí být použita při servisním zákroku. Dále se eviduje seznam oprav; oprava má jednoznačné číslování (rok + pořadí opravy), obsahuje informaci o autě, které bylo servisováno, datu a času přijetí vozu, datu předání zpět, ceně za opravu (pokud je vůz právě v servisu, datum předání a cena nejsou uvedeny), kontaktní telefon majitele. K opravě uvedeme seznam použitých součástek (ne vždy musí být pro opravu použity nějaké součástky, ale v rámci jedné opravy může být použito více součástek stejného typu).

Najděte sami v praxi alespoň dalších 5 případů, kdy je zapotřebí dlouhodobě evidovat údaje o nějakém typu objektů. Vyberte si jeden z nich a vypracujte pro ně seznam funkčních a nefunkčních požadavků a modely vnějšího chování.

Příprava na tutoriál - Zadání semestrálního projektu

Z vlastních úloh zadání informačního systému si vyberte jednu, kterou budete řešit jako semestrální projekt.

Zadání si nechte odsouhlasit od svého cvičícího nebo tutora.

Na konci každé kapitoly pak dostanete zadání další etapy řešení.

2. ANALÝZA INFORMAČNÍHO SYSTÉMU

Čas ke studiu kapitoly: celkem 12 hodin, 3 x 2 hodiny studium + 3 x 2 úlohy

Cíl V této kapitole se dozvíte

- co je analýza a jaké typy analýz se pro SW systémy provádějí,
- jakými nástroji se provádí funkční analýza,
- jakými nástroji se provádí dynamická analýza,
- jaká pravidla má dodržovat komunikace programu s uživatelem

a budete schopni

- provést úplnou analýzu menšího informačního systému,
- napsat ke všem typům analýzy dokumentaci,
- zpracovat návrh komunikace s uživatelem.

Výklad

2.1. Analýza a její druhy

❑ Co je analýza

Analýza znamená **studium problému (jeho poznání, popis, modelování)** dříve, než se začne provádět vlastní řešení problému. Za výsledek analýzy se považuje i odůvodněné negativní stanovisko, odmítnutí realizace.

V informatice je předmětem analýzy aplikační oblast reálného světa (*řízení podniku, obchod, školní evidence apod.*). Na základě analýzy se většinou provádí implementace nového systému.

Předmětem analýzy je nejčastěji

- existující systém - automatizovaný či neautomatizovaný, jehož struktura a funkce jsou zřejmé zákazníkovi z praxe a který se bude automatizovat *(evidence dosud ruční, buduje se IS),*
- existující systém, jehož struktura a funkce nejsou zřejmé ani zákazníkovi, ani řešiteli a je nutné je rozpoznat a popsat *(ztratila se dokumentace nebo je zastaralá)*,
- neexistující systém, o němž má zákazník nepříliš přesnou představu a neumí požadované funkce řešiteli vysvětlit *(evidence dosud neexistuje, bude se zavádět)*.

Výsledkem analýzy je specifikace systému. Specifikace systému stanoví:

- cíl řešení
- podrobně zdokumentovaný cílový stav v takové podobě, aby bylo možné na závěr posoudit, zda implementace cílového stavu dosáhla,
- důležité průvodní parametry spojené s řešením a provozem, jako cena, přínos, plánování, dosažený výkon ap.

Specifikační dokument = výsledek analýzy bývá součástí smlouvy a proto mívá právní platnost.

❑ Analytické modely

Analytik nemůže být odborníkem ve všech oblastech reality, které ve své profesi analyzuje. Proto je nutná spolupráce se zadavatelem a budoucími uživateli. Výsledky analýzy tedy musí být těmto odborníkům na analyzovanou realitu (ale většinou ne odborníkům na informační technologie) dostatečně srozumitelné. Natolik, aby uměli rozpoznat jejich věcnou správnost či chyby a nedostatky. K tomu slouží různé typy analytických modelů.

Protože na druhé straně je výsledek analýz zadáním pro implementaci, je nutné popsat a namodelovat systém přesně, jednoznačně, bezesporně. **Modelem rozumíme abstraktní obraz reality**.

Užitečné bude opět porovnání použití modelování programového systému s modelováním jiného oboru - architektonickým návrhem.

Příklad:

> *Architekt také nejprve modeluje (dvourozměrné nákresy budovy z různých pohledů, v perspektivě, denní a noční apod., nebo trojrozměrné modely ze dřeva, polystyrenu apod. i s modelem okolí - terénu, zeleně, sousedních budov atd.). Různé modely mohou být redundandní – zobrazovat stejné části systému. Pokud si neodporují a jsou srozumitelné, dávají dobrou představu uživateli a lze na jejich základě zahájit tvorbu detailní výkresové dokumentace.*

♦

Obdobně jsou užívány různé modely při analýze programu.

Z hlediska zkoumání informačního systému a z hlediska typů metod používaných k jeho vývoji rozeznáváme obvykle **tři základní dimenze:**

- datová analýza – modelující statickou strukturu databáze,
- funkční analýza – modelující algoritmy transformující (počítající) vstupní data na výstupní,
- časová nebo dynamická analýza – modelující možné časové návaznosti popsaných funkcí.

2.2. Datová analýza

Datová analýza je základem pro takové SW systémy, kde databáze, ukládání dat a vyhledávání informací z ní jsou hlavním účelem, tedy pro informační systémy. Datovou analýzu jsme probrali důkladně v předcházejícím semestru – v Teorii zpracování dat. V kapitole o struktuře relační databáze jsme se naučili správně navrhovat databázi. Z kapitoly o konceptuálním modelu umíme výsledek analýzy správně popsat a zobrazit.

Zopakujme si tedy, jak dostaneme výsledné konceptuální schéma:

Ze zadání se vyberou potřebné evidence objektů a jejich atributů, určí se funkční závislosti mezi atributy, pomocí již známých metod se navrhne struktura databáze v alespoň 3NF (= vznikne seznam typů entit a jejich atributů, typy entity se pojmenují).

Výsledný konceptuální model obsahuje

- lineární zápis seznamu typů entit a jejich atributů
- úplný grafický tvar ERD (2 úrovně)
    1. konceptuální schéma modelující realitu
    2. transformovaný ERD pro databázové schéma (bez multiatributů, vazeb M:N atd.)
- úplné tabulky atributů – datový slovník
- seznam dalších IO týkajících se domén, entit a vztahů.

Řešený příklad - Projekt – Datová analýza

Ze zadání IS Knihovna Kocourkov nejprve zaznamenáme atributy a jejich funkční závislosti (FZ). Ukáže se, že některá fakta k určení FZ ještě nejsou jednoznačná a musíme se doptat zadavatele.

Otázky a odpovědi zadavatele:

1. Je cena titulu při jenom vydání stejná? Ano, kupujeme od stálých vydavatelů za stálé ceny.

2. Máte dost knih stejných autorů? Ano, v beletrii je řada oblíbených autorů.

3. Kolik je vydavatelů a jak je evidujete? Máme adresář poštovní a e-mailový s jedinou adresou.

Ze 2. odpovědi vyplývá vhodnost doplnit id autorům a přiřadit knihy a autory k sobě.

Ze 3. odpovědi vyplývá vhodnost evidovat navíc i vydavatele a jejich adresy.

Také musíme zvážit doplnění některých umělých klíčů u entit s víceatributovým nebo nedefinovaným klíčem (tučně).

RU (**id_ctenar**, jmeno_ctenar, adr_ctenar, telef_ctenar, mail_ctenar; prir_cis, nazev, ISBN, rok, zanr, jazyk, zeme, anotace, cena; **id_autor**, jmeno_autor, poradi; dat_od, dat_do, dat_vraceno, pocet_upom; dat_rezer, dat_vypuj; **id_vydav**, naz_vydav, ulice_vydav, obec_vydav, psc_vydav, mail_vydav; **id_objed**, dat_objed, pocet)

F = {id_ctenar → jmeno_ctenar, adr_ctenar, telef_ctenar, mail_ctenar;

prir_cis → nazev, ISBN, id_vydav, rok, zanr, jazyk, zeme, anotace, cena; ... pozor! ne autor!

ISBN → nazev, id_vydav, rok, zanr, jazyk, zeme, anotace, cena;

id_autor → jmeno_autor;

ISBN, id_ctenar → poradi;

id_ctenar, prir_cis, dat_od → dat_do, dat_vraceno, pocet_upom;

id_ctenar, prir_cis, dat_rezer → dat_vypuj;

id_vydav → naz_vydav, ulice_vydav, obec_vydav, psc_vydav, mail_vydav;

id_objed → id_vydav, ISBN, dat_objed, pocet, dat_dodavky;}

Postupem známým z TZD provedeme návrh struktury databáze do 3NF. Primární klíče jsou podtrženy, cizí klíče označeny kurzivou. Výsledkem bude:

Ctenar (id_ctenar, jmeno_ctenar, adr_ctenar, telef_ctenar, mail_ctenar)

Titul (ISBN, nazev, *id_vydav*, rok, zanr, jazyk, zeme, anotace, cena)

Exemplar (prir_cis, *ISBN*)

Autor (id_autor, jmeno_autor) ... číselník autorů

Napsal (*id_autor, ISBN*, poradi) ... vazební tabulka co kdo napsal s pořadím autorů

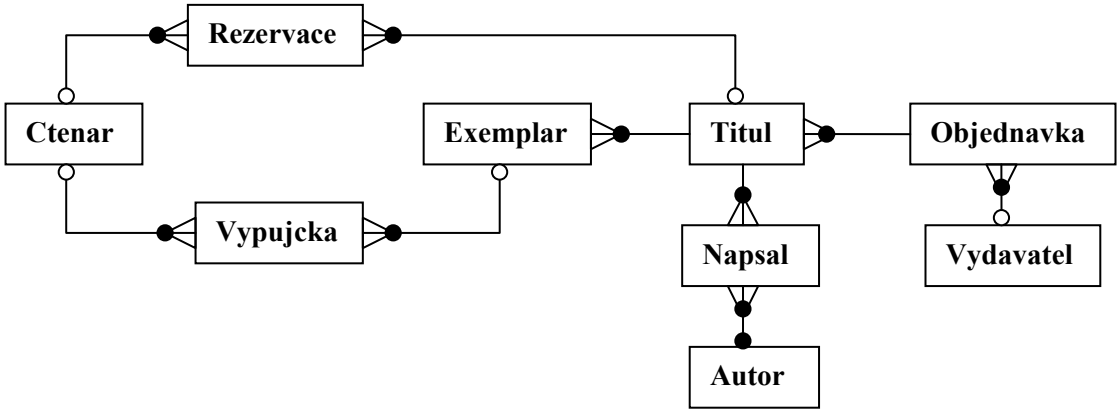
Vypujcka (*prir_cis, id_ctenar*, dat_od, dat_do, dat_vraceno, pocet_upom) ... historie výpůjček

Rezervace (*ISBN, id_ctenar*, dat_rezer, dat_vypuj) ... historie rezervací

Objednavka (id_objed, *id_vydav*, *ISBN*, dat_objed, pocet, dat_dodavky)

Vydavatel (id_vydav, naz_vydav, ulice_vydav, obec_vydav, psc_vydav, mail_vydav)

ERD této databáze je



Pro stručnost uvedeme jen část datového slovníku:

tabulka	atribut	dattyp	délka	klíč	NULL	index	IO	význam, poznám
Ctenar	id_ctenar	num	6	A	N			inkrement
	jmeno_ctenar	char	30	N	N		*1	celé jméno
	...							
Titul	ISBN	char	20	A	N			
	nazev	char	100	N	N			hlavní název
	id_vydav	num	6	N	A			cizí klíč Vydavatel
	rok	num	4	N	A			rok vydání
	...							
...								

*1 ... úplné jméno ve tvaru Příjmení Křestní, Titul

2.3. Funkční analýza

Když je hotova datová analýza, přistoupíme k funkční analýze. Ta má za úkol popsat všechny operace, které je zapotřebí s daty v navržené databázi provádět – všechny funkce IS. Obecně to je ukládání, modifikace a rušení dat, výpočty, třídění, vyhledávání informací, formátování výstupních informací apod. Funkční analýza opět vychází ze zadání IS, z požadovaných vstupů, výstupů a funkcí. Je výhodné, když je v zadání zpracována úplná tabulka událostí a reakcí.

Z nich vytváří analytik funkční model, vyjadřující logický sled a podstatu transformací údajů do systému vstupujících a ze systému vystupujících. Funkční model obsahuje tyto 2 úrovně:

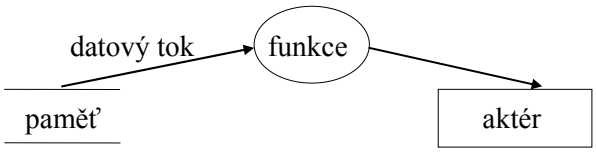
- vnější pohled je hrubý **grafický náhled na strukturu a hierarchii funkcí** systému,
- vnitřní pohledy jsou podrobně rozpracované **algoritmy** (minispecifikace) pro jednotlivé akce.

❑ Grafický model - diagram datových toků DFD

První vnější pohled na funkce se vytváří pomocí tzv. diagram datových toků (Date Flow Diagram, DFD). Je to grafický prostředek pro návrh a zobrazení funkčního modelu systému. Podobně jako ERD u datové analýzy má být DFD dostatečně jednoduchý a názorný i pro uživatele a může sloužit i k upřesňování zadání.

DFD zobrazuje algoritmy systému, vyjadřuje transformace dat z jedné formy do druhé; modeluje funkce systému pomocí grafu a přitom používá následujících základních prvků:

- funkce
- paměti
- aktéři
- datové toky



Funkce (proces, transformace) provádí transformaci dat vstupních na data výstupní, realizuje nějakou funkci nad daty. Zakresluje se kruhovým uzlem v grafu (někdy elipsou, obdélníkem), v uzlu je zaznamenán název funkce.

Každý proces v DFD má svůj název a jednoznačné číslo. Čísla je vhodné přidělovat hierarchicky: součástí čísla je číslo nadřízené funkce, do jejíhož rozkladu popisovaná funkce patří, druhou část tvoří jednoznačné číslo v rámci úrovně rozkladu (nemusí mít žádný vztah k pořadí provádění funkce).

Datový tok vyjadřuje přesun dat nebo informací z jedné části systému do jiné, z okolí systému do systému nebo ze systému do jeho okolí. Znázorňuje se hranou (úsečkou, obloukem) opatřenou šipkou, znamenající směr toku dat. Je možné použít šipky i oboustranně, když jde o dialog, stejná data tečou oběma směry.

Datový tok musí mít známý obsah a je zase pojmenovaný. Datové toky obsahují data, která jsou do systému vkládána, systémem zpracovávána nebo ze systému vypouštěna. U programových systémů jsou obsahem datových toků zprávy, čísla, znaky, záznamy, bity. Datové toky jsou jednou z forem propojení (komunikace) procesů.

Datová paměť je místo dočasného nebo trvalého uchování dat pro jejich pozdější využití. Je to obecnější pojem než soubor. Může být implementován jako pole, soubor textový, soubor databázový, kniha, šanon a leccos jiného. Používá se tam, kde mezi procesy existuje časové zpoždění při předávání dat. Znázorňuje se pomocí dvou rovnoběžek, mezi nimi je název paměti.

Pro každou paměť musí existovat alespoň jeden datový tok směřující do paměti a jeden směřující z paměti. Datový tok může vyjadřovat 1 výskyt dat, více výskytů dat, část jednoho výskytu či část z více výskytů. Paměť je pasivní prvek, data do paměti i z paměti musí vždy procházet přes proces. Paměť je další formou propojení procesů.

Aktér znázorňuje externí zdroj nebo cíl dat, objekt vně systému, s nímž systém komunikuje. Může to být člověk, skupina lidí (oddělení, instituce), jiný systém apod. Platí pro ně :

- aktéři jsou vně systému, toky mezi nimi a systémem představují rozhraní mezi systémem a vnějším světem,
- analytik nemá možnost měnit organizaci a chování entit vně systému ani změnit chování aktérů,
- vztahy mezi aktéry se v DFD nezachycují; mohou sice existovat, ale nejsou součástí navrhovaného systému; pokud by měli být aktéři a vztahy mezi nimi zahrnuty do systému, je třeba DFD reorganizovat.

Graficky se znázorňuje obdélníkem či čtvercem a opět je pojmenovaný.

Hierarchie DFD

Model systému vyjádřený pomocí DFD má obvykle hierarchickou strukturu. Jen velmi malý systém je možno vykreslit jediným diagramem. Proto dle podrobnosti rozkladu obvykle rozlišujeme několik úrovní DFD: vrchní = kontextový, střední, koncový = elementární. Pokud se IS vyvíjí postupem shora dolů, začíná se od kontextového diagramu a pokračuje se ke stále detailnějším diagramům.

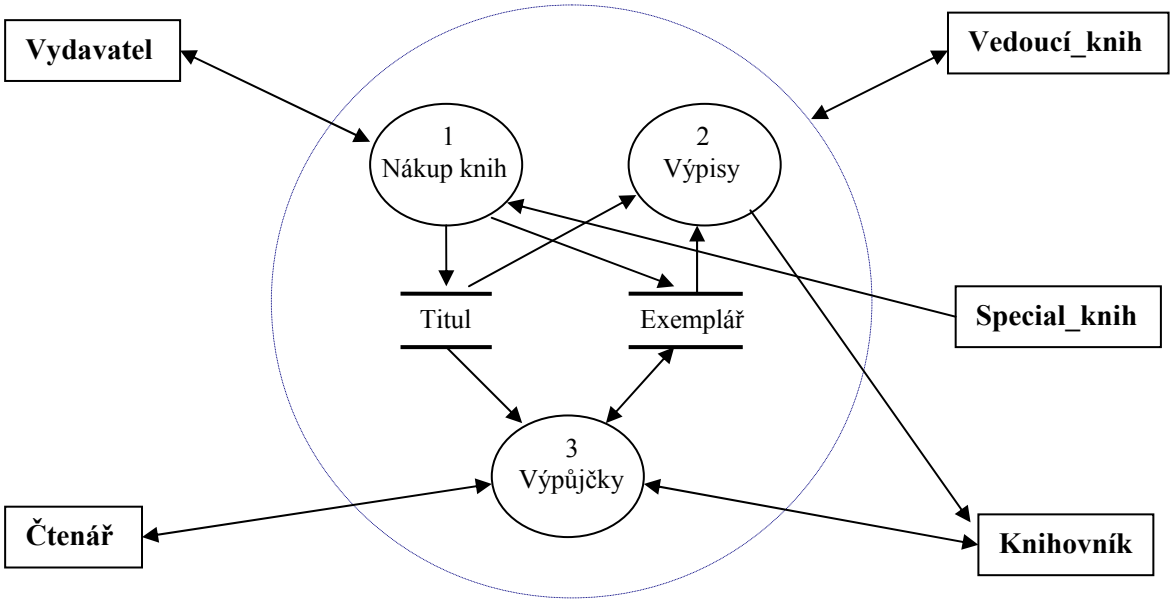
Na vrcholu hierarchie je pouze jeden DFD zvaný **kontextový** (tentýž, co známe ze zadání). Ten obsahuje celý systém jako jednu funkci, definuje hranice systému a všechny aktéry - zdroje systému a cílová místa dat. Systém je zde černá skříňka s definovanými vstupy a výstupy.

Bezprostředním rozkladem kontextového diagramu je DFD **úrovně 0**. Obsahuje základní funkce systému (obvykle rozklad na subsystémy) a jejich vztahy vyjádřené prostřednictvím datových toků a pamětí. Aktéři systému jsou totožní s kontextovým diagramem.

Dále se postupuje v rozkladu funkcí obdobně, vznikají DFD **1., 2. a dalších úrovní**. Každá funkce, kterou je možno dále rozložit, se rozkresluje novým diagramem nižší úrovně až na **elementární funkce**. Ty obsahují uživatelsky dále nedělitelné funkce, které se provádějí vždy celé najednou (co je elementární funkce a co dále dělitelná funkce je věcí analytika).

Příklad:

*Výše uvedený kontextový diagram **Knihovna** se do úrovně 0 zvětší a tak se zviditelní subsystémy pro nákup nových knih, evidenci knih a jejich výpůjček a pro inventurní a jiné výpisy pro knihovní potřeby.*



Modrá"bublina" je zvětšeninou celého IS. Po zvětšení se objevily detailnější funkce – zde subsystémy Nákup, Výpisy a Výpůjčky, očíslovány 1-3. Datové paměti, které spolupracují s víc než jednou funkcí, jsou vně těchto funkcí. Jsou to Tituly a Exempláře, které z funkce Nákup jsou zapisovány do evidence, do Výpisů poskytují informace a ve Výpůjčkách se z nich čtou informace. Případně při ztrátě knihy se do Exempláře zapisuje její stav.

Ostatní paměti jsou dosud „schovány" uvnitř svých funkcí: Vydavatelé a Objednávky ve funkci Nákup; Čtenáři, Výpůjčky, Rezervace, Autoři a Napsal ve funkci Výpůjčky.

Datové toky se upřesnily: Aktér Vydavatel komunikuje jen s funkcí Nákup – odebírá objednávky a posílá knihy s fakturou. Záznam o nových knihách jde do Titulu a Exempláře. Aktér Čtenář

komunikuje jen s Výpůjčkami – tam dodá své údaje a pak dává informace o požadovaných výpůjčkách a rezervacích, případně knihy vrací. „Obyčejný" knihovník jen realizuje výpůjčky nebo získává různé výpisy. Specialista knihovník dopisuje do nových knih speciální zařazovací údaje (žánr, ...) a konečně Vedoucí knihovník má přístup ke všem funkcím: vydává objednávky, používá výpisy, může zasáhnout do výpůjčního systému. Aby všechny tyto datové toky „nepočáraly" graf, zakreslí se jen jeden k okraji modré celkové bubliny a ten znamená tok ke ***všem*** *funkcím.*

Je zřejmé, že každá vnitřní funkce se bude dále po zvětšení rozpadat na několik dalších. Ty pak budou číslovány v rámci své nadfunkce. Například v Nákupu budou funkce 1.1 Objednávky, 1.2.Nové knihy, 1.3.Doplnění spec_informací atd. Pokud se některá z těchto vnitřních funkcí rozpadne ještě na další podfunkce, jsou číslovány v rámci ní dalším číslem. Například funkce 1.1.Objednávky bude mít další podfunkce 1.1.1.Příjem nabídky, 1.1.2.Nová objednávka (vystavení nové objednávky na základě nabídky od vydavatelů).

Jakmile rozklad funkcí dojde k elementárním, které se provádějí celé najednou (na jednu volbu uživatele v menu), dále se v rozkladu nepokračuje. Podrobný popis každé elementární funkce se již zapíše jako algoritmus.

Pokud si celou hierarchii DFD představíme jako strom rozkladů, elementární funkce tvoří listy stromu.

♦

❑ Pravidla tvorby prvků DFD

Obecná pravidla pro DFD

1. Složitost jednoho DFD má být taková, aby formát nepřesahoval velikost A4, aby neobsahoval velké množství uzlů (někdy se doporučuje jen 6 uzlů, rozhodně počet funkcí v rozmezí 3 - 9) a aby byl srozumitelný pro uživatele, analytika a návrháře.
2. Diagram má být technicky správný, srozumitelný, přehledný a esteticky uspořádaný. Bubliny mají být stejně velké *(jinak větší bubliny třeba jen díky delšímu názvu funkce bývají chápany jako důležitější)*, datové toky se nekříží apod. Doporučuje se překreslovat graf až do grafické dokonalosti.
3. Musí být dodržena konzistence DFD, logická soudržnost diagramu. Ta není samozřejmá, protože jedna skutečnost je díky hierarchickému rozkladu rozkreslena více či méně podrobně na několika DFD. Kontrolou je porovnávání vstupů a výstupů mezi funkcemi téže úrovně i mezi jednou úrovní a její „zvětšeninou", rozkladem do detailnější úrovně.

Pravidla pro funkce

1. Při číslování procesů se identifikuje jednak úroveň rozkladu, do něhož funkce patří, jednak proces v rámci úrovně (např. 2.4.3).
2. Názvy procesů mají být stručné, výstižně vyjadřovat funkční náplň procesu, ne příliš obecné a tak nic-neříkající *(například správný název: Vystavení faktury, nesprávný: Zpracování dat, Editace)*.
3. Žádné dvě funkce nesmí mít stejný název.
4. Nesmí existovat proces generující výstupy bez pomoci vstupů (perpetum mobile).
5. Nesmí existovat proces, který má pouze vstupy a žádné výstupy (černá díra).
6. Neznázorňují se žádné inicializační ani závěrečné procedury.
7. Neznázorňují se cykly mezi funkcemi.

Pravidla pro datové toky

1. Datové paměti smějí být propojeny jen prostřednictvím funkce, tedy datový tok do / z paměti musí vycházet z / do procesu.
2. Datový tok z / do terminátoru musí procházet přes proces.

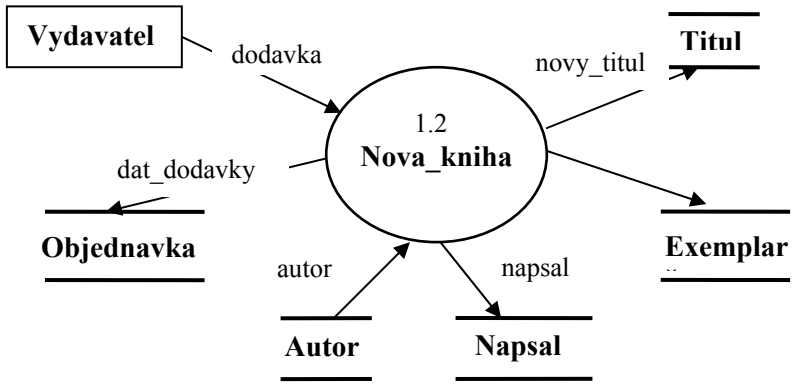
3. Datové toky mezi funkcemi znázorňují pouze přenášená data, nevyjadřují volání jedné funkce druhou ani předávání řízení.
4. Datový tok s týmž názvem může být v DFD použit na více místech pokud název znamená skutečně tentýž datový tok se stejným obsahem.
5. Doporučuje se dodržovat označení datového toku z/do datové paměti :
   - bez označení = přenáší se jeden výskyt
   - označen datovou pamětí = přenáší se jeden nebo více výskytů
   - označen jednoznačně jinak = přenáší se část výskytů.

Pravidla pro datové paměti

1. Paměti se objeví až na té úrovni, kde jsou viditelné funkce do pamětí zapisující nebo z pamětí čtoucí.
2. Vyhledání pro aktualizaci paměti se chápe jako součást zápisu do paměti, nevyznačují se zvláštní šipkou; šipka dovnitř znamená jakékoliv provádění změn (vkládání dat, aktualizaci, rušení).
3. V paměti jsou uloženy výskyty dat se stejnou strukturou. Jestliže tok z / do paměti přenáší celý výskyt, nemusí se pojmenovávat, je určen obsahem a názvem paměti.

Příklad:

V IS Knihovna je například elementární funkcí 1.2. Nové knihy. *Její DFD už bude mít pojmenovány všechny* ***datové toky****.*



Vydavatel dodá objednané knihy a s nimi fakturu, na ní jsou základní údaje o knihách a cena.

Na faktuře jsou uvedeny údaje:vydavatel, id_objed, ISBN, název, autoři, počet exemplářů. Tato struktura zatím neexistuje, proto ji pojmenujeme a zapíšeme do datového slovníku:

dodavka (vydavatel, id_objed, ISBN, název, autoři:multi, počet exemplářů)

Funkce 1.2. vytvoří pro každý Titul nový záznam – neúplný, zatím bez anotace a některých dalších údajů, ty doplní jinou funkcí specialista knihovník. Datový tok je dosud nedefinovaný, pojmenujeme jej a zadáme strukturu a zapíšeme do datového slovníku.

novy_titul (ISBN, nazev, id_vydav, cena)

Dále zapíše několik nových záznamů Exemplářů (dle zadaného počtu). Protože jde o úplný záznam tabulky Exemplář, nemusí se datový tok pojmenovávat.

Dále zapíše několik nových záznamů do tabulky Napsal (dle počtu autorů), opět vždy celý záznam, ale pro jeden vstupní záznam o titulu obecně několik autorů. Proto je datový tok pojmenován stejně jako tabulka, tedy

napsal.

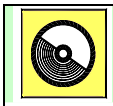
Ale id_autora pro tabulky Napsal *je nutné vyhledat v číselníků autorů* Autor*, odtud se čtou opět celé záznamy – datový tok je pojmenován* autor.

Konečně do Objednavky *podle* id_objed *funkce doplní* dat_ dodavky = *splnění objednávky. Jde o datový tok s jediným atributem, je tedy zbytečné jej pojmenovávat jinak a stačí uvést jméno atributu*

dat_dodavky.

♦

Takovýto elementární DFD dává úplný přehled o všech vstupních a výstupních datech příslušné elementární funkce. Chybí již jen podrobný algoritmus funkce. Ten si popíšeme v následujících odstavcích.



Animace

Na CD-ROMu s tímto výukovým textem jsou animované příklady na DFD.

Jsou to soubory:

- DFD\DFD1 objednavka.exe
- DFD\DFD2 prijem.exe
- DFD\DFD3 vydej.exe
- DFD\DFD4 vypisy.exe

Minispecifikace

Minispecifikace je popis elementární funkce = funkce na nejnižší úrovni hierarchického rozkladu. Popisuje podrobně její algoritmus. Funkce na vyšších úrovních nemá smysl specifikovat, protože jsou jen množinou funkcí nižší úrovně.

K minispecifikaci lze použít mnoho forem popisu - od přirozeného jazyka až po formální nástroje popisu algoritmu. Vždy je však třeba dodržet následující

Formální pravidla pro minispecifikace

1. Existuje jedna minispecifikace pro každou elementární funkci z množiny DFD.
2. Vyjadřuje postup, jak jsou datové toky do funkce vstupující transformovány na výstupní.
3. Vyjadřuje, co funkce znamená věcně, nemusí vyjadřovat způsob implementace funkce (proto není vhodný programovací jazyk). Popisují však všechny podrobnosti transformace dat včetně základního formátování dat na vstupu a výstupu.
4. Nezavádí redundandní popisy, nevyjadřuje znovu, co je zobrazeno v DFD nebo popsáno v DD.
5. Množina výrazů pro popis je jednoduchá a nepříliš rozsáhlá, má používat standardní vyjadřování.
6. Algoritmus musí být srozumitelný analytikovi, programátorovi i uživateli.
7. Popis procesu musí být strukturovaný.

Pro minispecifikaci by mohl sloužit běžný programovací jazyk nebo vývojový diagram, ale pro analytickou práci a pro konzultace se zadavatelem to nejsou vhodné prostředky.

Minispecifikace jsou vstupem pro etapu návrhu implementace, tam jsou doplněny řadou implementačních podrobností.

❑ Strukturovaný jazyk

Strukturovaný jazyk je často používaný prostředek pro popis minispecifikací. Původní návrh tzv. strukturované angličtiny pochází od DeMarca.

Strukturovaný jazyk je přirozený jazyk doplněný o omezující pravidla tvorby vět (syntaxe), aby výsledný popis nepřipouštěl několik různých výkladů. V případě angličtiny lze strukturovaný jazyk přirovnat jazyku Pascal s tím, že lze používat rozšířených „funkcí“, kdy některé činnosti programu jsou jen vhodně pojmenovány. Při dodržení pravidel lze definovat libovolné funkce dle potřeby. V literatuře se často vyskytuje strukturovaná angličtina. Pro popis programátorům je vhodná, ovšem pro komunikaci s českým uživatelem ne. Jak jsme již zdůrazňovali vícekrát, je nutné s uživatelem komunikovat jemu co nejbližšími prostředky. Pokud ho budeme nutit k dodržování pravidel formulování svých tvrzení v češtině, tak se s tím smíří. Používání anglických slov ale bude považovat za programování a řekne si, proč vlastně toho programátora platí.

Slovník jazyka je složen z

- rozkazovacího způsobu sloves
- pojmů (podstatných jmen) z datového slovníku
- rezervovaných slov pro formulaci logiky procesu

Syntaxe jazyka obsahuje následující řídicí struktury pro definování procesu:

- jednoduché rozkazovací věty (pro příkazy, které dělá program)
- jednoduché oznamovací věty (pro činnosti, které dělá uživatel)
- větvení:

  (1) JESTLIŽE <podmínka>
  PAK <činnost pro platnou podmínku>

  (2) JESTLIŽE <podmínka>
  PAK <činnost pro platnou podmínku>
  JINAK <činnost pro neplatnou podmínku>

  (3) VYBER PŘÍPAD
  PŘÍPAD 1: <podm1>
  <činnost pro platnou podmínku 1>
  PŘÍPAD 2: <podm2>
  <činnost pro platnou podmínku 2>
  . . .
  JINAK <činnost pro neplatnou žádnou podmínku>

- cykly

  (1 PRO KAŽDÝ <záznam> DĚLEJ <opakovaná činnost >

  (2) DOKUD <podmínka> DĚLEJ <opakovaná činnost >

  (3) DĚLEJ <opakovaná činnost > DOKUD <podmínka>

Příklad:

Funkce, která rozhodne u každého studenta o přidělení prospěchového stipendia takto: student může dostat stipendium jen s minimálním počtem kreditů 60/rok. Má-li průměrný počet bodů za předmět >85, dostane maximální stipendium, jinak s průměrem <80,85> střední a s průměrem <75,80) nejnižší. Ostatní studenti nedostanou stipendium.

Následující minispecifikace není zcela přesná, není tam v bodě 3.1.správně uvedeno načítání údajů, protože zřejmě v tabulce Student budou jména a v tabulce Stud_vysledky *budou výsledky studentů za jednotlivé předměty. Je tedy nutné bod 3.1. rozepsat podrobněji. Ale pro názornost formulování algoritmu tento zápis zatím stačí, uživatel - neinformatik by mu dobře rozuměl.*

Tučně jsou uvedeny názvy proměnných a tabulek, velkým písmem příkazy a klíčová slova příkazů.

1. ZOBRAZ formulář pro zadání max, střed a min stipendium

Zadejte hodnoty stipendia maximální: střední: minimální:

2. Uživatel – studijní referentka povinně vyplní hodnoty **max, stred, min**

3. PRO KAŽDÉHO studenta z tabulky **Stud_vysledky** DĚLEJ

   3.1. PŘEČTI z tabulky **Stud_vysledky** hodnoty **login, jmeno, predmet, kredity, body**

   3.2. SPOČÍTEJ součet kreditů **Suma_kredit** za aktuální akademický rok

   3.3. JE-LI **Suma_kredit** >= 60
   PAK
   SPOČÍTEJ průměrný počet bodů za předmět **prum**
   VYBER ALTERNATIVU:
   PŘÍPAD 1: **prum** > 85
   stip = **max**
   PŘÍPAD 2: **prum** <80, 85>
   stip = stred
   PŘÍPAD 3: **prum** <75, 80)
   stip = min
   JINAK: **stip** = 0
   JINAK
   stip = 0

   3.4. JE-LI **stip**>0
   PAK ULOŽ do tabulky **Stipendia** hodnoty **login, jmeno, stip**

   KONEC CYKLU

4. VYTISKNI tabulku Stipendia ve tvaru

Přiznaná prospěchová stipendia za rok ...		
Login	**Jméno**	**Výše stipendia**
...	**...**	**...**
...	**...**	**...**

Všimněme si, že minispecifikace je slovně psaná logika programu = algoritmus, jen případně bez některých zřejmých technických detailů (zde výpočet průměru apod.).

♦

❑ Pravidla pro formulování algoritmu

- Algoritmus musí být strukturovaný, používat standardní programové řídicí struktury:
  sekvenci
  větvení
  cykly
  procedury

- Algoritmus musí rozlišovat jako samostatné body:
  příkazy, které provádí program (rozkazovací způsob),
  činnosti, které provádí uživatel (uvedením Uživatel provede ...),
  příkazy manipulující s databází (čtení záznamů, ukládání, modifikace a rušení),
  příkazy prováděné v paměti počítače (výpočty, testování podmínek, tisky, načítání údajů od uživatele atd.).
- Další zásady
  je vhodné rozlišovat identifikátory údajů v databázi a v paměti,
  pracovně se zobrazují i obrazovky pro komunikaci s uživatelem a výstupní sestavy (alespoň orámováním jejich vzhledu, aby byly snadno rozpoznatelné pro návrh komunikace).

Datový slovník

Důležitou součástí datové analýzy a nástroj k provázání datové, funkční i časové analýzy je datový slovník (Data Dictionary - DD). Slouží ke slovnímu formalizovanému popisu dat systému z pohledu uživatele.

Obsahuje

- **složení dat v datových pamětech** (viz závěr kapitoly Konceptuální schéma)
- **složení dat v datových tocích** (pokud struktura datového toku není totožná s některou popsanou datovou pamětí, pak popis struktury dat, případně i rozložení na atomické položky)

Pro datové paměti i datové toky se mimo standardní syntaktické údaje uvádějí:
významu datových pamětí z DFD
významu datových toků z DFD
specifikace domén a měrných jednotek dat v datových tocích a pamětech
údaje o vztazích mezi entitami z ERD jako cizí klíče
všechna další integritní omezení.

Struktura datového slovníku je stejná, jak ho známe z konceptuálního schématu, doplněny jsou struktury nově pojmenovaných datových toků.

Příklad:

Datový slovník Knihovny (pro stručnost opět neúplný) doplněný o dva nové datové toky z příkladu DFD Nové knihy:

dodavka (vydavatel, id_objed, ISBN, nazev, autor:multi, pocet exemplářů)
novy_titul (ISBN, nazev, id_vydav, cena)

tabulka/tok	atribut	dattyp	délka	klíč	NULL	index	IO	význam, poznám
...								
Titul	ISBN	char	20	A	N			
	nazev	char	100	N	N			
	...							
...								
dodavka	vydavatel			N	N			viz Vydavatel
	id_objed			A	N			viz Objednavka
	ISBN	char	20	N	N			viz Titul
	nazev	char	100	N	N			viz Titul
	autor			N	N			viz Autor
	pocet			N	N			
novy_titul	ISBN	char	20	A	N			viz Titul
	nazev	char	100	N	N			viz Titul
	...							

♦

Popis struktury domény složitějšího atributu

Někdy se v datové analýze rozhodneme, že složitější atribut není potřeba rozkládat na atomické části, protože je samostatně nebudeme využívat *(typicky například jméno nebo adresa)*. Chápeme tedy takový atribut jako atomický, ale požadujeme dodržení jakýchsi pravidel při jeho vyplňování. Pokud doménu atributu je potřeba popsat podrobněji, notace zápisu pro složení dat obvykle vychází ze zjednodušené BNF a používá symbolů:

=	skládá se z, je definován jako
+	a
(. . .)	volitelný člen, výskyt vůbec, jednou nebo vícekrát
{. . .}	opakovaný člen jednou nebo vícekrát
[. . .|. . .]	výběr z možností

Příklad:

Pro naši evidenci jména a adresy stačí jediný složený atribut, který budeme považovat za atomický, ale požadujeme jeho přesný formát:

```
adresa    =  [(titul) + jmeno + prijmeni + ulice | poštovní schránka] + mesto + (PSC) + (zeme)
titul     =  [pan | paní | slečna | Dr. | Ing. | Prof.]
jmeno     =  {povoleny znak}
prijmeni  = {povoleny znak}
. . .
povoleny znak  =  [A - Z | a - z |  | - | ' ]
```

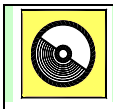
♦

❑ Vztah datové a funkční analýzy

Při tvorbě algoritmů funkcí (minispecifikací) se může objevit potřeba definovat další datové struktury, dosud v databázi neexistující. Mohou to být další atributy už existujících entit, mezivýsledné tabulky nebo data, jejichž nutnost se objevila až při zpřesnění algoritmů. V tom případě je nutné doplnit ERD, strukturu databáze a datový slovník o tyto nové tabulky.

Dalším doplňkem jsou již zmíněné datové toky.

Později uvidíme, že se databáze může doplňovat i při dynamické analýze. Výsledkem celé analýzy je tak ERD 3. úrovně.



Animace

Na CD-ROMu s tímto výukovým textem jsou animované příklady na minispecifikace.

Jsou to soubory:

- Minispec\M1 prijem jednoho zbozi.exe
- Minispec\M2 vraceni zbozi.exe
- Minispec\M3 prijem z cele faktury.exe
- Minispec\M4 vyhledani stavu zbozi.exe
- Minispec\M5 inventura skladu.exe
- Minispec\M6 zpetna inventura.exe
- Minispec\M7 nova karta zbozi.exe
- Minispec\M8 zruseni zbozi ze skladu.exe

Řešený příklad - Projekt – Funkční analýza

Ze zadání IS Knihovna a s výslednou datovou analýzou provedeme funkční analýzu.

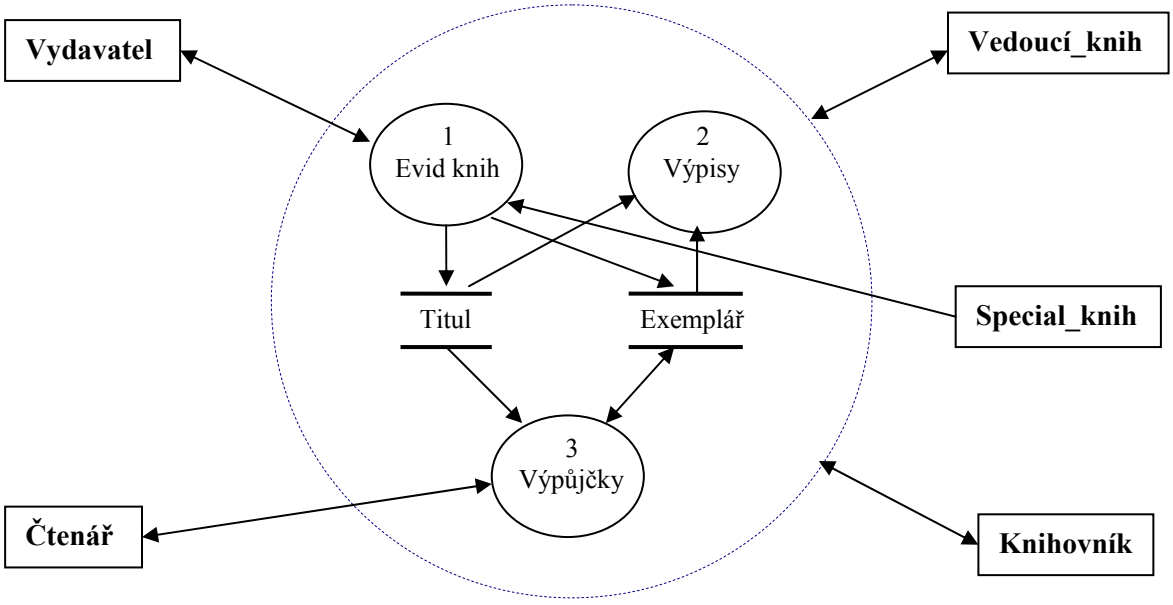
Vycházíme z požadovaných funkcí v zadání:

Nový čtenář, změny čtenářů. Místo zrušení bude archivace kvůli pozdějším statistikám.
Nová kniha, doplnění jejího zařazení do oddílu.
Zrušení knihy z evidence do archivu kvůli ztrátě, zničení nebo vyřazení.
Záznam výpůjčky a vrácení knihy, tisk výpůjčního lístku, tisk upomínek, pokuty.
Záznam nové objednávky pro vydavatele nebo knihkupce.
Možnost výběru knih pro čtenáře podle názvu, autora, žánru, vydavatele, jazyka, roku.
Výběr knih nevypůjčených alespoň 6 měsíců.
Provedení a tisk inventurních seznamů. Výpočet měsíční statistiky výpůjček.

Rozepíšeme je po jednotlivých elementárních funkcích (prakticky se to provede již ve specifikaci zadání) – zde pro stručnost neuvádíme celou tabulku:

číslo fce	událost v realitě	funkce systému	uživatel	subsystém
F1	nový čtenář, změny	edit_ctenar	knih, ctenar	Vypujcky
F2	odhlášení čtenáře	archiv_ctenar	knih, ctenar	Vypujcky
F3	nová kniha	nova_kniha	knih	Evid_knih
F4	doplnění údajů o knize	dopln_kniha	special_knih	Evid_knih
F5	ztráta, zničení knihy	zrus_kniha	knih	Evid_knih
F6	objednání knih u vydavatele	objed_kniha	ved_knih	Evid_knih
F7	výpůjčka knihy čtenářem	vypuj_kniha	knih, ctenar	Vypujcky
...				
F30	měsíční statistika	mesic_stat	ved_knih	Vypisy

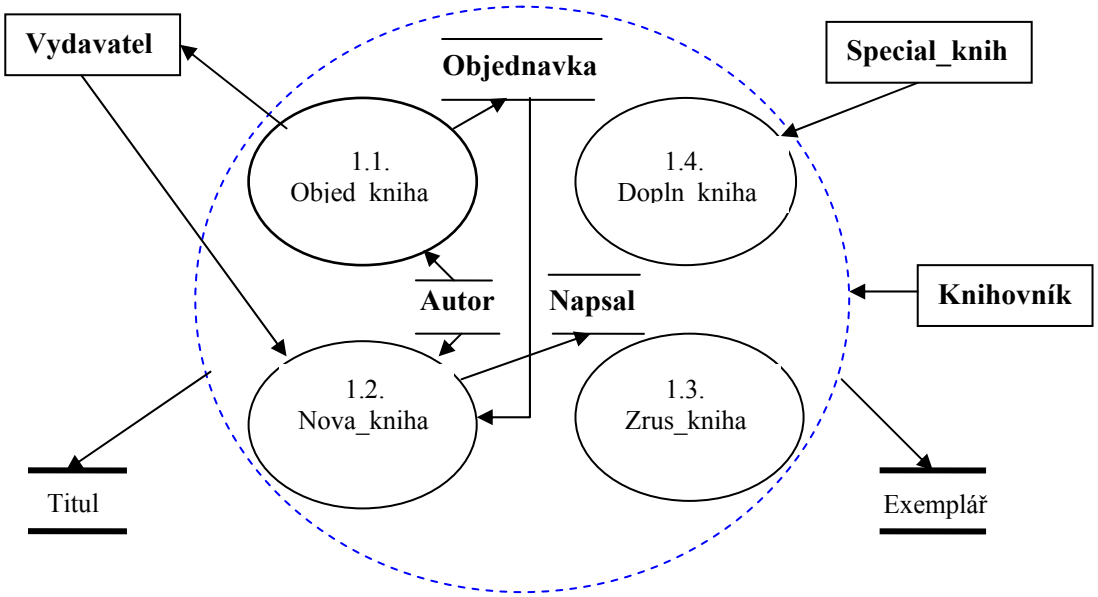
Nejprve rozmyslíme rozdělení všech funkcí do subsystémů – navrhneme 3 subsystémy:



- objednávky, nákup nových knih a jejich evidence, ztráty a zničení = Evid_knih
- evidenci čtenářů a jejich výpůjček = Vypujcky
- prohledávání knih, rešerše, inventury, statistiky = Vypisy

DFD této 0.-té úrovně již známe z příkladu v kapitolce 2.3.

Každý subsystém dále rozkreslíme na jednotlivé funkce. Protože jich již není mnoho, nepoužijeme další meziúrovně. Zde si zobrazíme jen DFD subsystému Evid_knih a pak elementární DFD pro funkci Nova_kniha, kterou již také zníme z kapitolky 2.3.

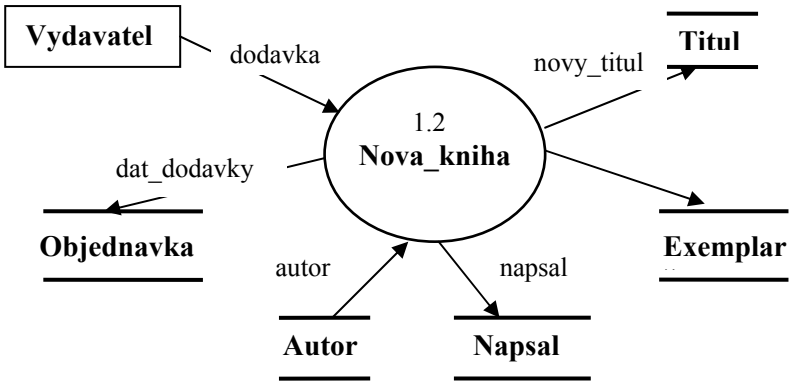


Konečně uvedeme jednu minispecifikaci pro elementární funkci Nova_kniha. Struktura tabulek je:

Titul (ISBN, nazev, *id_vydav*, rok, zanr, jazyk, zeme, anotace, cena)
Exemplar (prir_cis, *ISBN*)
Autor (id_autor, jmeno_autor) ... číselník autorů
Napsal (*id_autor, ISBN*, poradi) ... vazební tabulka co kdo napsal s pořadím autorů
Objednavka (id_objed, *id_vydav*, *ISBN*, dat_objed, pocet, dat_dodavky)
Vydavatel (id_vydav, naz_vydav, ulice_vydav, obec_vydav, psc_vydav, mail_vydav)

Minispecifikace funkce F3 Nova_kniha

Funkce zapíše novou knihu do evidence Knihovny, tedy do Titul, Exemplar a Napsal. Dále poznamená do Objednavky její splnění pro zaevidovanou knihu.



Algoritmus:

1. Zobraz prázdný formulář nové knihy ve tvaru

Nová kniha	
ISBN:	
Název:	
Autoři:	výběr z číselníku Autor (id_autor, jmeno_autor)
Vydavatel:	výběr z číselníku Vydavatel (id_vydav, naz_vydav)
Rok vydání:	
Cena:	

2. Uživatel-knihovník zadá ISBN, název, rok a cenu, z číselníků vybere autory a vydavatele.
3. Zapiš do Titul nový záznam (ISBN, nazev, *id_vydav*, rok, cena)
4. Urči nové prir_cis jako inkrement z tabulky Exemplar.
5. Zapiš do Exemplar nový záznam (prir_cis, ISBN).
6. Nastav poradi = 1
7. Pro každého zadaného autora
   7.1 vyhledej podle jména v Autor odpovídající id_autor
   7.2 zapiš nový záznam do Napsal (id_autor, ISBN, poradi)
   7.3 poradi = poradi + 1
8. Zobraz dotaz

nová kniha	konec

9. Jestli uživatel zvolí nová kniha, vrať se na bod 1
   jinak konec funkce

Ostatní minispecifikace zde popisovat nebudeme.

2.4. Dynamická analýza

Funkční analýza popíše algoritmy elementárních funkcí, ale nic neuvádí o jejich časových návaznostech. Funkce jen popisují postupy jak - když dostanou příslušná data vstupní - je zpracují na data výstupní. Protože není obecně možné spouštět kdykoliv jakoukoliv funkci (například *dokud nemám přijatý materiál na sklad, nemohu jej vydávat do výroby*), musí časové návaznosti definovat další typ modelu – model dynamický. Uvedeme si 2 dynamické modely vhodné pro IS. Bude to obecný stavový diagram a životní cyklus entity, vhodný pro nejnižší úroveň popisu stavů entit.

❑ Stavový diagram STD

Stavový diagram (State Transition Diagram) slouží k modelování chování systému v časových návaznostech, tedy v závislosti na čase nebo na pořadí funkcí. Popisuje časové následnosti procesů, které DFD nezaznamenává. Modeluje chování systému v závislosti na působení **vnějších událostí** nebo na základě **vnitřních změn stavů**.

Definují se stavy, v nichž se systém může nacházet *(např. základní denní režim účtování, měsíční uzávěrka, zjištěné manko)* nebo vnitřní stavy čekání na událost *(nová faktura: přijata, dán příkaz bance, zaúčtována, ...).*

Definice:

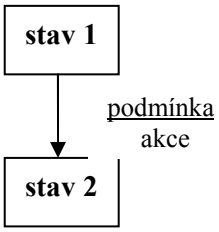
Stav je podmnožina hodnot atributů jednoho nebo více objektů (typů entit). Za určitého stavu má objekt jeden druh chování, při změně stavu se mění i jeho chování.

Definice:

Přechod mezi stavy je taková změna hodnot atributů, že objekt přejde z jednoho stavu do druhého. Je to buď **modifikace hodnot atributů** nebo **změna časová** nebo vnitřní **impuls systému** či **impuls vnější**. Změna stavu nastane při **rozpoznání, že je splněna nějaká podmínka**. Ze stavu do stavu přejde systém provedením určitých akcí.

Akce je provedení (elementární) operace nad objektem.

STD znázorňuje jednotlivé stavy a přechody mezi nimi opět pomocí grafu. Uzly tvoří stavy systému, hrany znamenají změny stavů. Stavy jsou znázorněny obdélníkovými uzly. Stav systému vyjadřuje interval mezi jednotlivými akcemi, který platí v daném okamžiku (*systém ve stavu čekání na heslo, systém připraven k přijetí příkazu apod.).*



Změna stavu nastane při rozpoznání, že je splněna nějaká podmínka *(příkaz přijat, heslo zadáno, uplynul zadaný čas apod.)*. Ze stavu do stavu přejde systém provedením určitých akcí *(vyhledej informaci, zapiš nového zaměstnance ap.)*. Podmínky a akce se zaznamenávají v STD jako popis orientovaných hran (změn stavů). Popis má tvar zlomku, nahoře je podmínka přechodu do následujícího stavu, dole je název akce tento přechod realizující.

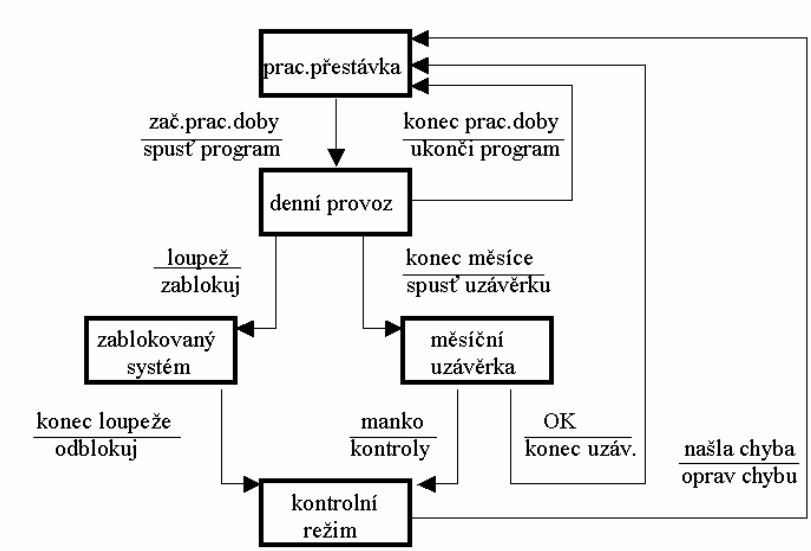
Někdy může být podmínka složená nebo akce není jediná, ale skládá se ze sekvence dílčích akcí. Pak má zlomek tvar

podm1; podm2; … ; podmn	...	přechod způsobí kterákoliv z nich
akce1;akce2; … ; akcem	...	akce se provedou sekvenčně

Význačné jsou počáteční a koncové stavy. Systém musí mít definován jeden počáteční stav a jeden nebo více koncových stavů. Výchozímu stavu žádný stav nepředchází, po koncových stavech žádný nenásleduje.

Příklad:

Hrubý STD pro IS Banky. Stav Denní provoz je možno na podstavy rozkreslit dále.



Je zřejmé, že stavy denní provoz, měsíční uzávěrka, kontrolní režim *jsou stavy systému, v nichž se systém chová vždy jistým způsobem a v každém z nich jinak, než v jiném. Při měsíční uzávěrce se například nemohou provádět denní operace výběrů a vkladů peněz, v denním provozu zase není možno provádět kontrolní operace, protože se hotovosti neustále mění.*

Ale každý tento stav je možno detailněji rozdělit na podstavy, případně i v několika úrovních, až se dostaneme na stavy jednotlivých entit. Například jednou z evidovaných entit je konkrétní účet klienta, který může mít stavy kladný < 100000 s účet s úročením 1%, kladný >100000 s úročením 3%, zablokovaný soudním příkazem, záporný > -10000 bez úročení, záporný <-10000 s úrokem 5% apod.

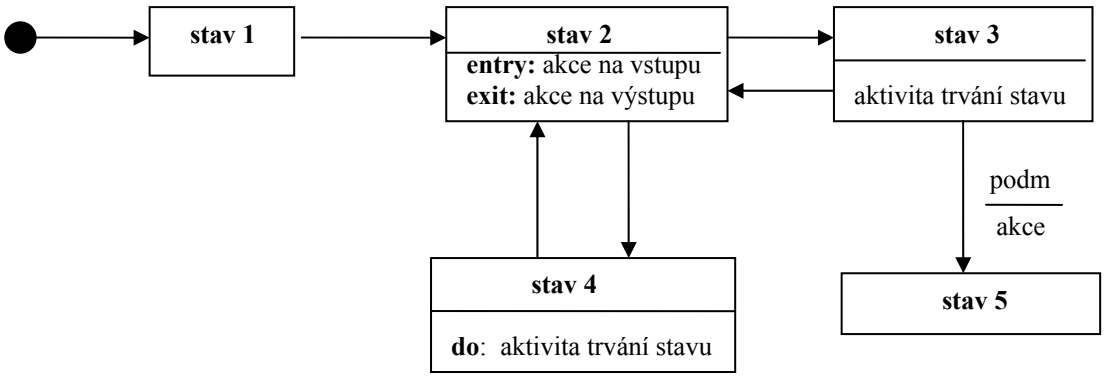
♦

Vnitřní přechody znamenají zpracování události s případnými připojenými akcemi beze změny stavu. Zapisují se do stavu a mají stejný tvar, jako událost zapsaná u přechodu.

entry je událost, ke které dochází při každém vstupu do daného stavu; zapisuje se tak jednou místo na hrany na každém vstupu;

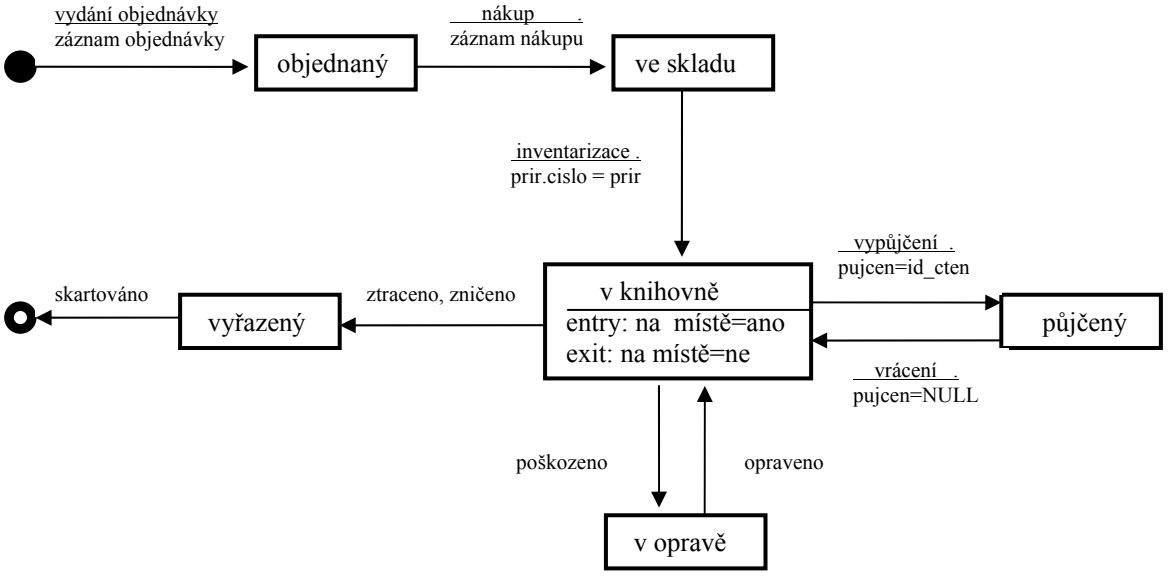
exit je událost, ke které dochází bezprostředně před každým opuštěním daného stavu; zapisuje se jednou místo na výstupní hrany;

do je aktivita trvání stavu; stav může skončit „sám od sebe“, je-li jeho trvání spojeno s nějakou činností; pak je podmínka trvání formulována za **do** *(není zaplacena celá částka faktury, systémový čas < 24:00 ...)* a stav má jediný výstup bez popisu.



Příklad:

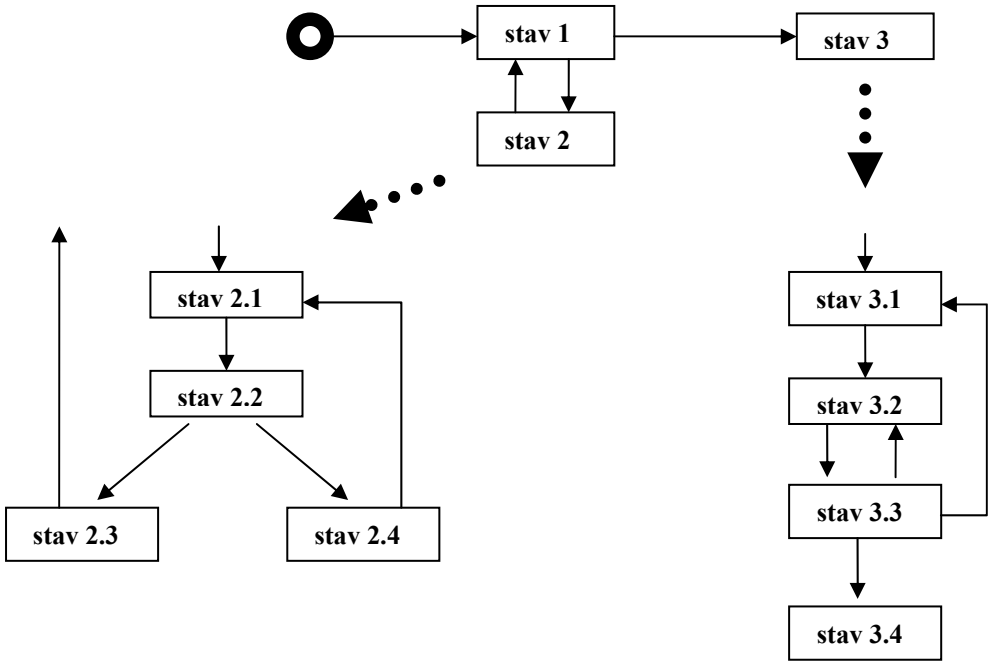
V Knihovně jeden z typů entit je Exemplář. Každá entita = každý exemplář knihy může být v různých stavech, které je třeba v IS evidovat.



Stav je tedy charakterizován hodnotami některých atributů každého exempláře knihy. Pro jeden stav je možno používat jen některé funkce. Například pro vypůjčení je možno nabízet jen knihy ve stavu „v knihovně“, žádné jiné. Při vrácení se nabízejí knihy jen knihy ve stavu „vypůjčený“ atd.

♦

Reálný systém mívá obvykle desítky stavů, které se na jeden diagram nevejdou, nebo by byl nepřehledný. V tom případě se používá členění diagramů do hierarchické struktury, obdobně jako u DFD. Každý stav vyšší úrovně může být popsán samostatným STD nižší úrovně,vazbu mezi úrovněmi je vhodné zviditelnit číslováním stavů podle podobných pravidel, jako u DFD.



❑ Kontrola konzistence stavů

Jedním z problémů při tvorbě STD je, zda je graf správný a úplný. Pokud zapomeneme na některý stav nebo některý přechod, může ve výsledném IS chybět některá přechodová funkce nebo nebude rozeznatelný některý stav.

Jsou 2 přístupy ke kontrole:

1. Identifikujeme = pojmenujeme všechny stavy, zakreslíme je každý samostatně a pak hledáme všechny možné přechody mezi nimi a ty zakreslíme.
2. Vyjdeme z počátečního stavu, hledáme všechny možné změny tohoto stavu, zakreslíme přechody a následující stavy; pokračujeme stejně pro nově vzniklé stavy.

Součástí kontroly by měly být odpovědi na následující otázky:

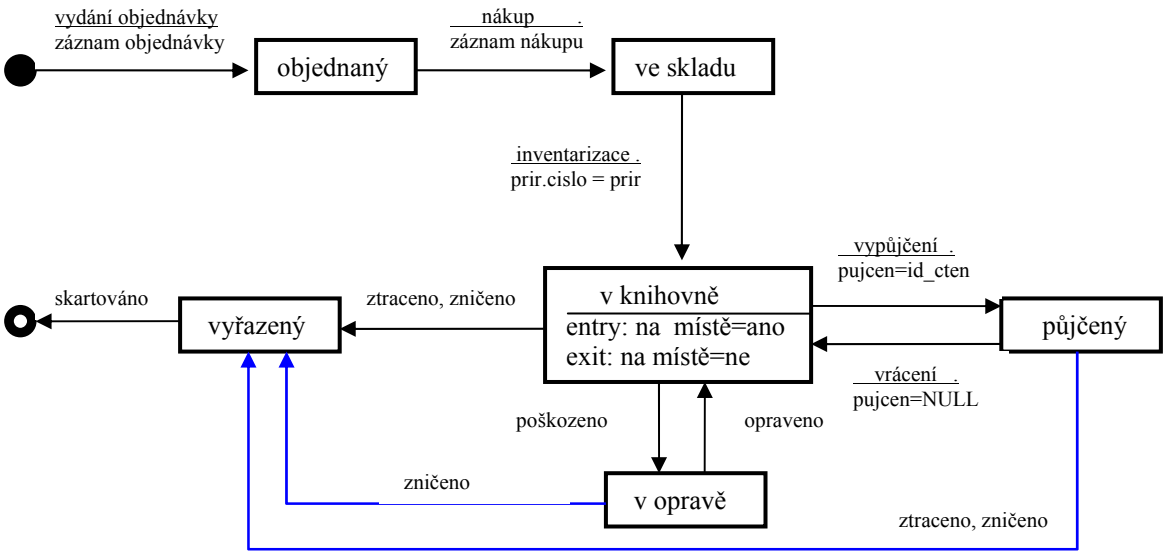
- byly definovány všechny stavy? (jaké všechny situace potřebujeme rozeznávat?)
- jsou všechny stavy dosažitelné? (existuje ke každému stavu reálný vstup?)
- lze všechny stavy opustit? (existuje ke každému nekoncovému stavu reálný výstup?)
- odpovídá chování systému v každém stavu všem možným podmínkám?

Kontroly lze prověřit například pomocí matice stavů. Nakreslíme matici, do jejího levého sloupce zapíšeme všechny stavy, do 1. řádku všechny události či podmínky, při nichž dochází ke změně stavu. Pak vyplňujeme jednotlivá pole matice dvojicemi akce/následující stav = v definovaném stavu (vlevo) při definované události (nahoře) se pomocí zapsané akce přejde do zapsaného nového stavu. Pokud při přechodu akce chybí, zaznamená se jen nový stav.

	událost 1	**událost 2**	**...**	**událost 8**
stav 1	akce 1/stav 3	akce 2/stav 4		**---**
stav 2	**---**	--- /stav 5		akce 5/stav 6
...				
stav 7	**---**	**---**	akce 8/stav 4	**---**

Systematickým vyplněním celé tabulky projdeme všechny možné přechody a máme jistotu, že na některý přechod nezapomeneme. Ovšem stále nemáme jistotu, že jsme nezapomněli na některý stav nebo událost.

Příklad: *V IS Knihovna tvoříme STD exempláře knihy. Zobrazíme si znovu jeho STD:*



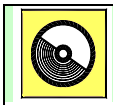
Vytvoříme odpovídající matici stavů. Do ní místo názvu stavu zapisujeme pro stručnost jen jeho číslo, události poškozeno a opraveno zde chybí, názvy akcí jsou zkráceny.

Matice stavů je

	objed	nákup	invent	vypůj	vrácení	ztrac	...	skartac
1 počáteční	záz obj/2	---	---	---	---	---		---
2 objednaný	---	záz ex/3	---	---	---	---		---
3 ve skladu	---	---	prir/4	---	---	---		---
4 v knihovn	---	---	---	puj=cte/5	---	**/ 7**		---
5 půjčený	---	---	---	---	puj=ne/4	**/ 7**		---
6 v opravě	---	---	---	---	---	**/ 7**		
7 vyřazený	---		---	---	---	---	---	**/8**
8 skartovan	---	---	---	---	---	---	---	---

Systematickým vyplněním tabulky zjistíme případné chybějící přechody – například ze stavu 5 do stavu 7 při podmínce ztraceno, podobně při neopravitelném poškození ze stavu 6 do 7 apod.(označeno modře).

♦



Animace

Na CD-ROMu jsou další animované příklady na stavové diagramy.

- STD\STD bankovni ucet.exe
- STD\STD exemplar knihy.exe
- STD\STD faktura vydana.exe
- STD\STD jizda.exe
- STD\STD matrika.exe
- STD\STD penezni uver.exe
- STD\STD portal.exe
- STD\STD portal.exe
- STD\STD reklamace.exe
- STD\STD student.exe
- STD\STD tiket.exe

❑ Vztahy mezi analytickými modely

Každý model je zaměřen na jiný aspekt systému: datový model zobrazuje statickou strukturu databáze, funkční model popisuje algoritmy, které vstupní data transformují na výstupní data a dynamický model zobrazuje možné přechody stavů celé databáze nebo jejích částí do jiných stavů.

Jednotlivé skutečnosti se zpravidla projevují ve více modelech systému, proto je nutné prověřovat konzistenci každého modelu uvnitř i mezi modely navzájem.

Ověření konzistence návrhu - jeho různých modelů je daní za použití strukturalizace popisu.

Dobrý CASE systém by měl takovou konzistenci kontrolovat nebo ji pomáhat udržovat.

Řešený příklad - Projekt – Dynamická analýza

S výslednou datovou a funkční analýzou ještě ověříme úplnost řešení pomocí dynamické analýzy. Definujeme a zakreslíme stavy systému a stavy některých entit. Databáze je:

Ctenar (id_ctenar, jmeno_ctenar, adr_ctenar, telef_ctenar, mail_ctenar)
Titul (ISBN, nazev, *id_vydav*, rok, zanr, jazyk, zeme, anotace, cena)
Exemplar (prir_cis, *ISBN*)
Autor (id_autor, jmeno_autor) ... číselník autorů
Napsal (*id_autor, ISBN*, poradi) ... vazební tabulka co kdo napsal s pořadím autorů
Vypujcka (*prir_cis, id_ctenar*, dat_od, dat_do, dat_vraceno, pocet_upom) ... historie výpůjček
Rezervace (*ISBN, id_ctenar*, dat_rezer, dat_vypuj) ... historie rezervací
Objednavka (id_objed, *id_vydav*, *ISBN*, dat_objed, pocet, dat_dodavky)
Vydavatel (id_vydav, naz_vydav, ulice_vydav, obec_vydav, psc_vydav, mail_vydav)

Již jsme uvedli STD pro Exemplar. Proberme, které entity dále se budou vyskytovat v různých stavech:

Ctenar má jen stavy, které nazveme například „aktivní“ a „ukončený“. Prvního se dosáhne zápisem nového čtenáře a žádná změna jeho atributů nemá na tento stav vliv. Druhého se dosáhne zrušením (=archivací) čtenáře, STD tedy není nutný. Podobně zvážíme entity Autor, Napsal, Vydavatel.

Titul má stavy: objednaný (nejsou k němu exempláře), nový (nevyplněny zanr, anotace), aktivní (vyplněno vše), archivovaný (přesunut do archivu).

Vypujcka má stavů více: aktivní (dat_vraceno IS NULL), zpožděná neupomínaná (syst_datum > dat_do AND dat_vraceno IS NULL AND pocet_upom = 0), upomínaná nevrácená (syst_datum > dat_do AND dat_vraceno IS NULL AND pocet_upom > 0), vrácená (dat_vraceno IS NOT NULL) a archivovaná (přesunuta do archivu).

Obdobně zvážíme entity Rezervace a Objednavka.

Závěrem zvážíme stavy celého systému a subsystémů. Zřejmě je možné provádět kdykoliv všechny funkce s jedinou výjimkou – při inventuře aktuálního stavu by neměly být prováděny výpůjčky a vrácení. Proto stavy systému budou dva: aktivní a inventura, případně neaktivní (vypnutý IS).

2.5. Komunikace s uživatelem

❑ Co patří do komunikace člověk - počítač

Součástí analýzy by měl být i návrh komunikace s uživatelem – uživatelský vzhled programu.

Uživatelský vzhled programu je součástí specifikace; jeho vhodný návrh je důležitý zvláště s rozvojem možností grafiky, použití myši, hlasových výstupů ap. Měl by být konzultován s uživatelem a případně s odborníky na psychologickou stránku komunikace, ergonomii apod.

Uživatel většinou není profesionálem v počítačových profesích a musí se teprve zaškolit. Z hlediska didaktického je důležité, aby první kroky ve styku s počítačem nebo novým SW byly jednoznačné, dobře pochopitelné a jednoduché. Dojde-li k omylům již při vkládání údajů a vydávání příkazů, uživatel snadno podlehne dojmu, že použití počítače a programu je příliš složité a pro něj nezvládnutelné. Je demoralizován, zanevře na počítače, přestává být aktivní. Proto je nutné navrhovat programové systémy z hlediska člověka intuitivně, aby byl dostatečně motivován, znal svou úlohu v celém systému, uměl v každé situaci reagovat, uvědomoval si stav rozpracovanosti své úlohy.

Hlavním vstupem pro návrh komunikace je množina minispecifikací – jejich obrazovková okna (ovládací a řídicí prvky - menu, vstupní formuláře, výstupní sestavy, dotazovací okna, informační a chybová hlášení apod.)

Komunikací rozumíme způsob a formu vedení dialogu počítače a uživatele, tedy

- volbu akcí uživatelem (**menu**-systém, jeho formát a ovládání),
- způsob a formát ukládání a modifikace dat (**vstupní formuláře**),
- možnosti **výběrů informací**, formulaci požadavků (varianta QBE),
- formát **výstupů** (sestavy, jejich standardní hlavičky a patičky, označení),
- formát **dotazů** programu uživateli,
- formát **informačních a chybových hlášení** uživateli**.**

Úkolem je zpracovat návrh jednotného uživatelského vzhledu programu.

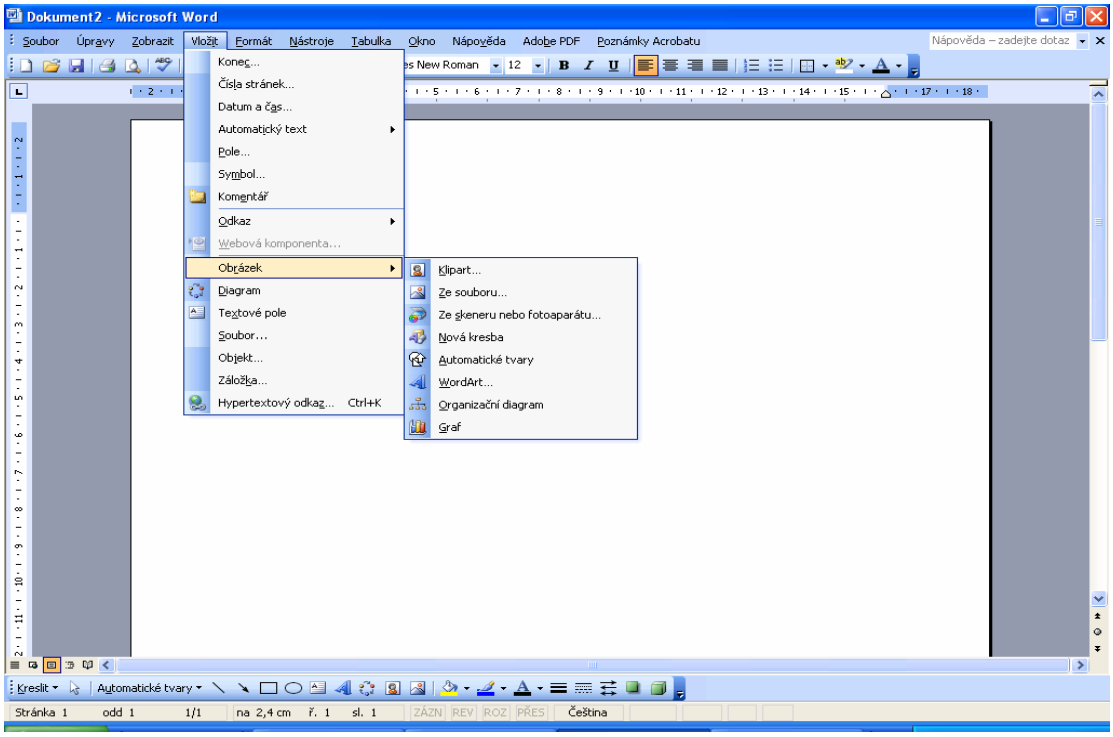
Nástroje pro návrh komunikace

- základní grafická úprava prostředí, použití kláves, myši, tlačítek, ...
- styl komunikace, styl dotazů, informačních a chybových hlášení, styl nápověd, vzhled uživatelských oken,
- formát vstupních formulářů a podformulářů, formát výstupních sestav,
- ovládání všech těchto prvků,
- použití fontů textů, použití barev pro písmo a pozadí, vzhled a umístění oken, ...

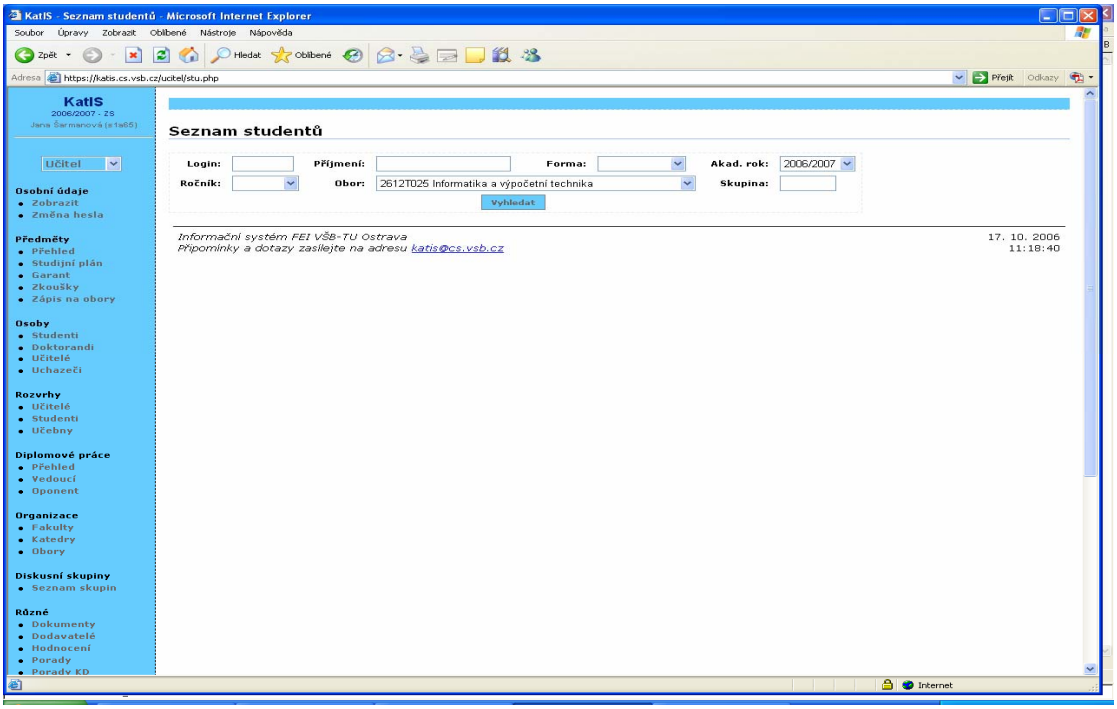
Příklady menu:

Pro menu informačního systému - souboru funkcí nabízených uživateli na výběr – je standardně používáno několik typů, většině zkušenějších uživatelů známých. Je vhodné vybrat jeden z nich, byť vzhled nemusí být totožný (volbou barev, písma apod.):

1. *Klasický menu systém se základní lištou nahoře (často s podlištou ikon pro nejčastější volby) a s roletovými podmenu i více úrovní; mimo jiné používaný základními programy MS:*

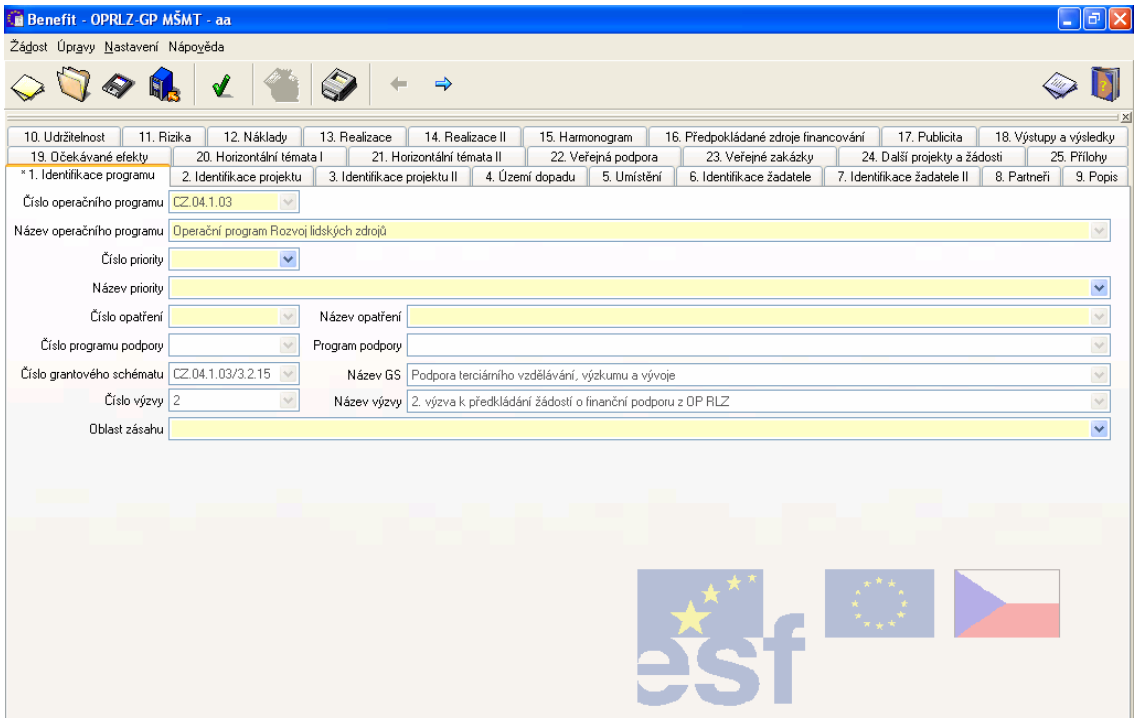


2. *Menu s lištou a ikonami nahoře + podmenu vlevo nebo vpravo ve svislém okně, případně rozbalitelné, nebo nahoře v okně další podmenu, nebo nahoře v okně volby pro formulování výběru (selekce) z dat:*



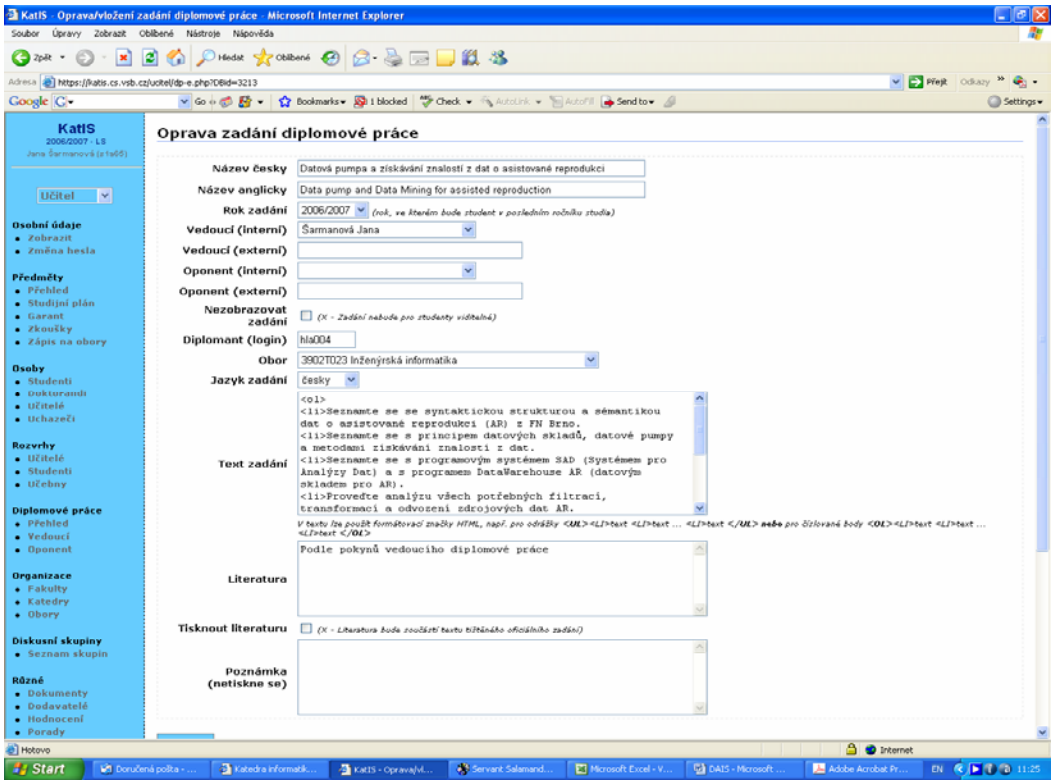
♦

3. *Menu se záložkami, lištou a ikonami nahoře.*

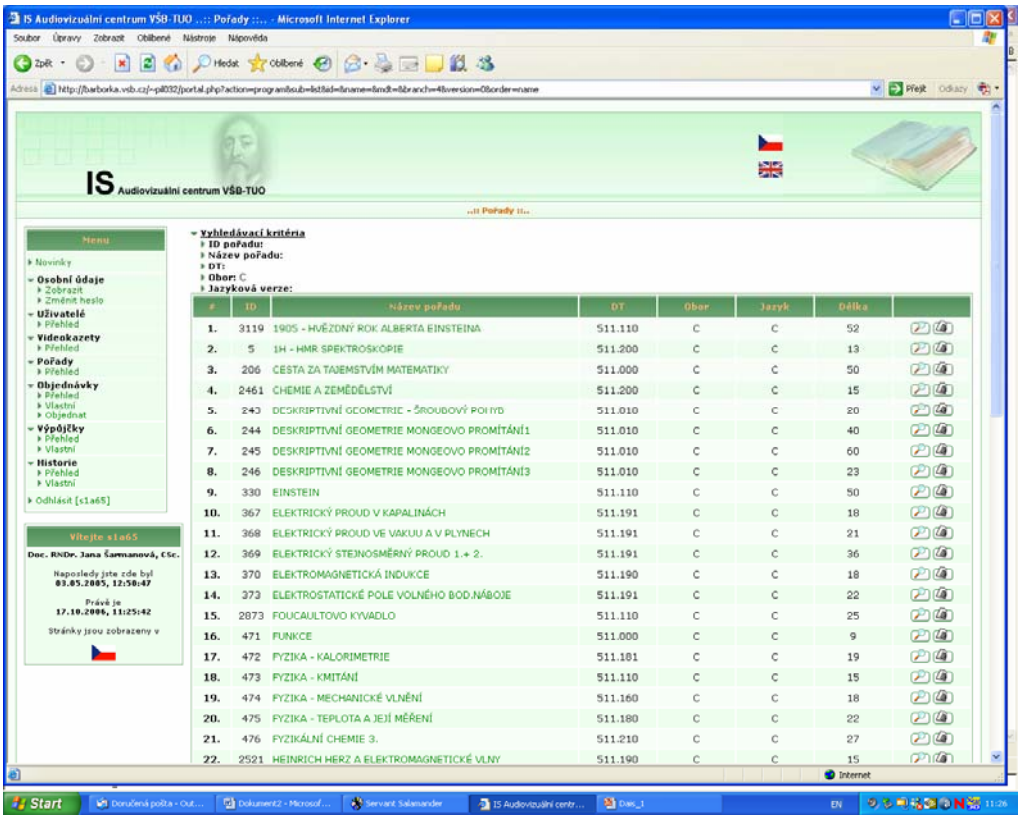


Příklady formulářů a reportů

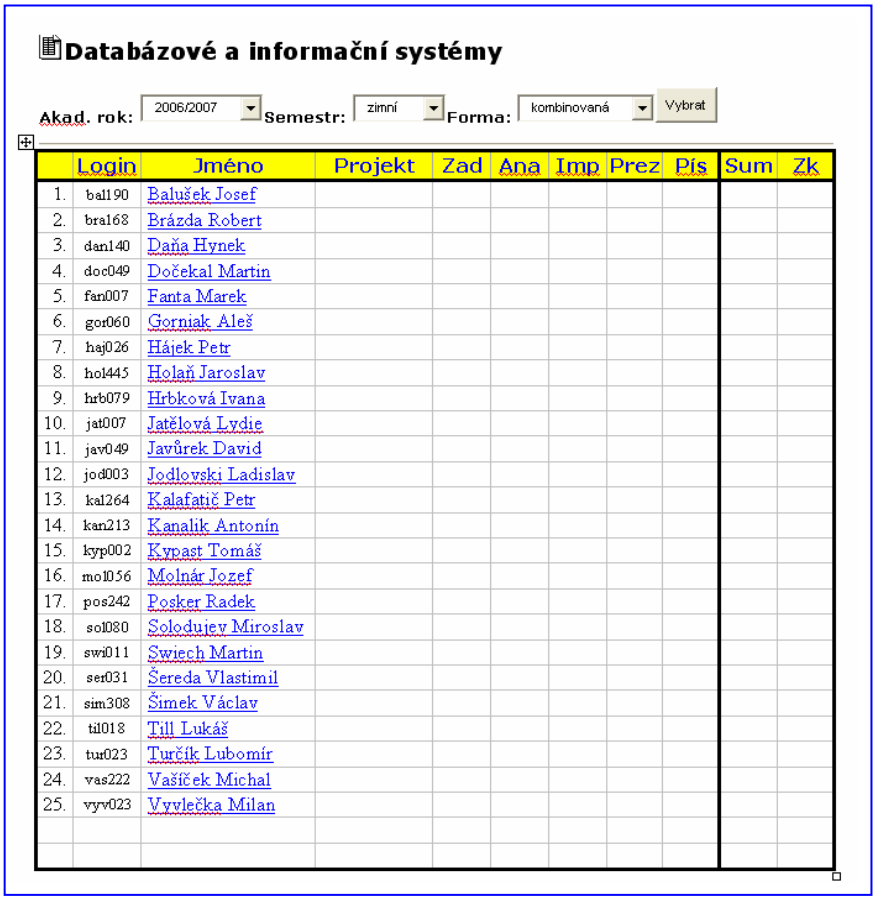
Vstupní formulář pro interaktivní ukládání a editaci záznamů:



Výstupní sestava na obrazovku s možností manipulací s daty



Klasická pracovní výstupní sestava = report ve tvaru tabulky:



Databázové a informační systémy

Akad. rok: 2006/2007 Semestr: zimní Forma: kombinovaná Vybrat

	Login	Jméno	Projekt	Zad	Ana	Imp	Prez	Pís	Sum	Zk
1.	bal190	Balušek Josef								
2.	bra168	Brázda Robert								
3.	dan140	Daňa Hynek								
4.	doc049	Dočekal Martin								
5.	fan007	Fanta Marek								
6.	gor060	Gorniak Aleš								
7.	haj026	Hájek Petr								
8.	hol445	Holaň Jaroslav								
9.	hrb079	Hrbková Ivana								
10.	jat007	Jatělová Lydie								
11.	jav049	Javůrek David								
12.	jod003	Jodlovski Ladislav								
13.	kal264	Kalafatič Petr								
14.	kan213	Kanalik Antonín								
15.	kyp002	Kypast Tomáš								
16.	mol056	Molnár Jozef								
17.	pos242	Posker Radek								
18.	sol080	Solodujev Miroslav								
19.	swi011	Swiech Martin								
20.	ser031	Šereda Vlastimil								
21.	sim308	Šimek Václav								
22.	til018	Till Lukáš								
23.	tur023	Turčík Lubomír								
24.	vas222	Vašíček Michal								
25.	vyv023	Vyvlečka Milan								

Klasická jednoduchá výstupní sestava s „firemní“ hlavičkou a patičkou

hlavička sestavy

sloupcové nadpisy

řádky sestavy

patička sestavy

VŠB-TU Ostrava

Výplatní listina za únor 2007

Zpracováno 5.3.2007

Jméno	Mzda hrubá	Daň z příjmu	Mzda čistá
Adam Karel	12 000.-	1 000.-	11 000.-
Beneš Josef	32 200.-	10 200.-	22 000.-
...			
Žižka Jan			
Celkem	345 600.-	145 600.-	200 000.-

Klasická vícestránková výstupní sestava s hlavičkou, patičkou a stránkováním.

logo firmy

hlavička sestavy

sloupcové nadpisy

řádky sestavy

patička stránky

VŠB-TU Ostrava Strana 1

Výplatní listina za únor 2007

Zpracováno 5.3.2007

Jméno	Mzda hrubá	Daň z příjmu	Mzda čistá
Adam Karel	12 000.-	1 000.-	11 000.-
Beneš Josef	32 200.-	10 200.-	22 000.-
...			
Mezisoučet	**127 400.-**	**27 400.-**	**100 000.-**

hlavička stránky

VŠB-TU Ostrava Strana 2

Výplatní listina za únor 2007

sloupcové nadpisy

řádky sestavy

patička sestavy

Jméno	Mzda hrubá	Daň z příjmu	Mzda čistá
Novák Pavel	12 000.-	1 000.-	11 000.-
Novotný František	32 200.-	10 200.-	22 000.-
...			
Celkem	**345 600.-**	**145 600.-**	**200 000.-**

Klasická vícestránková výstupní sestava s hlavičkou, patičkou, s grupami a podgrupami.

hlavička sestavy

VŠB-TU Ostrava Strana 1

Výplatní listina za únor 2007

Zpracováno 5.3.2007

sloupcové nadpisy

Fakulta katedra	Jméno	Mzda hrubá	Daň z příjmu	Mzda čistá
hlavička grupy **Fakulta 100**				
hlavička podgrupy **Katedra 101**				
	Adam Karel	12 000.-	1 000.-	11 000.-
	Beneš Josef	32 200.-	10 200.-	22 000.-
	...			
patička podgrupy **Celkem katedra 101**		98 400.-	38 400.-	60 000.-
hlava další podgrupy **Katedra 102**				
	Adámek Cyril			
	Barnabáš jan			
	...			
pata další podgrupy **Celkem katedra 101**				
	...			
	...			
	...			
Celkem katedra 199				
pata grupy **Celkem fakulta 100**				
Fakulta 200				
Katedra 201				
	...			
	...			
...				
...				
Celkem katedra 999				
Celkem fakulta 900				
patička celé sestavy **Celkem**		**345 600.-**	**145 600.-**	**200 000.-**

řádky sestavy

Samozřejmě je možné nastavit stránkování i tak, že každá grupa nebo podgrupa začíná na nové stránce.

♦

❑ Pravidla pro návrh komunikace člověk – počítač

Současný informatik je sice zvyklý na mnohé SW systémy a jejich vzhled, ovládání, nápovědy, hlášení apod. Přesto při psaní svých prvých programů si mnohé zásady neuvědomuje a bude vhodné si je shrnout a trochu popsat. Pokud píše program pouze pro sebe, je na něm, zda se bude řídit níže uvedenými zásadami. Ovšem program pro zákazníka je povinen následující pravidla dodržovat.

- Princip **prvořadosti uživatele,** tzv. true image při návrhu formátu formulářů a výstupních sestav. Pokud to neodporuje věcně (jiná data, jiný postup), je vhodné zachovat všechny dosavadní zvyklosti uživatele ve vzhledu a uspořádání formulářů.
- Princip **jednotnosti** (jednotný styl **vzhledu i ovládání** v celém systému, obdobné situace zobrazovat obdobně); součástí tohoto principu je návrh jednotného, jednoznačného a jednoduchého **komunikačního jazyka**, vstup/výstupní zprávy jednotné, podléhají stejným syntaktickým a sémantickým pravidlům *(například chybová hlášení „není připojena síť“, „tato akce není povolena“ nebo informační hlášení „nevybrán žádný záznam“, „vybráno 2000 záznamů“, „vše OK“, „dosud nerealizováno“)*; jednotné využití ovládacích kláves *(F1, ESC, Enter, PgUp, PgDn)* nebo ovládacích tlačítek myši *(ne například obdobná tlačítka pojmenovávat různě - OK, Odeslat, Uložit, Zavřít,Potvrdit, ...)*.

  S tím souvisí již zmíněný jednotný vzhled základních komunikačních prvků – menu, formulářů, sestav, dialogů, chybových a informačních hlášení. Jednotný styl komunikace se odrazí příznivě i v jednotném stylu programování;
- Princip **vlídnosti:** realizované helpy, nápovědy, přesně formulované otázky, jemná upozornění na chyby; zprávy pozitivní, ne negativní *(ne „Chyba ...“, ale „Zadejte údaj jako celé číslo“)*. Žádný druh humoru, opakovaný vtip není vtip, může být i nepříjemný a odporovat principu vlídnosti.
- Zprávy vydávané systémem musí **respektovat kontext**, ve kterém se uživatel nachází, mají být dostatečně podrobné a informující, co dál *(ne všude jen "chybný údaj" a už vůbec ne „ERROR“, ale například "Záporný příjem nelze evidovat")*.
- Respektovat **úroveň zkušeností uživatele** a respektovat **zaměření uživatele.** Jinak se dávají zprávy úřednici, jinak vedoucímu, jinak programátorovi.
- Minimalizovat čas pro vstupní zprávy uživatele
  - optimalizovat počet kroků, pomocí nichž se uživatel dostane k akci, kterou chce realizovat - minimalizovat počet úderů na klávesnici, kliků myši,
  - zprávy vkládané uživatelem mají být co nejstručnější, aby se omezilo množství překlepů, nepřesnosti, aby se urychlila komunikace.
- Zajistit **úplnost a správnost** vstupní informace
  - podrobit každý vstup všem v úvahu přicházejícím kontrolám,
  - umožnit v odůvodněných případech zdůvodnění či opakované potvrzení odpovědi.
- Maximalizovat **spolehlivost komunikace**
  - odlišit zprávy a data uživatele od zpráv systému - barvou, umístěním na obrazovce, písmem, sytostí ap.
  - dát signál o přijetí každého požadavku, aby uživatel věděl, co se děje, když se dlouho nic neděje: pípnutím o přijetí požadavku, zprávou „pracuji“, chybovou zprávou apod.
  - nepředpokládat, že si uživatel něco pamatuje z předcházejícího kroku.
- **Poskytnout nápovědu** v každé situaci, když uživatel neví, jak dál, co má odpovědět, jak dál pokračovat. Zabudovat uživatelskou příručku do programu.
- Umožnit kdykoliv **návrat** v komunikaci. Kromě chyby uživatele by systém měl vždy umožnit změnu názoru uživatele nebo vycouvání při chybné volbě.

- Optimalizovat **množství výstupních informací**
  - před výstupem spočítat množství výstupních zpráv, v extrémních případech vydat o tom zprávu *(„vašemu dotazu vyhovuje 12 566 záznamů, chcete je vypsat všechny?“)*
  - řešit případy zjevného i skrytého nedostatku informací (*„vašemu dotazu nevyhovuje žádný záznam“)*.

❑ Analýza příčin chyb a principy správné reakce na ně

- jako příkazy a odpovědi uživatele musí být systém schopen **přijímat jakákoliv data**; musí rozpoznat data správná od chybných a musí o tom dát zprávu uživateli; samozřejmě nesmí havarovat při zadání chybných dat, neboť - jak známo - uživatel je schopen všeho;
- hlášení chybových stavů musí být ve formě srozumitelné uživateli a odpovídající konkrétní situaci *(ne „přetečení rozsahu pole“, ale například „položek je mnoho, povoleno je jen 5“)*
- zařadit **zápis chyb do souboru** chyb; dát uživateli možnost zapsání zprávy do tohoto souboru chyb;
- **chyby automaticky neopravovat**, i když je systém umí rozeznat a opravit; vhodné řešení je buď oprava – zpráva uživateli o chybě a její opravě – potvrzení uživatele - akce nebo chybové hlášení - nové zadání od uživatele; uživatel si tak nezvykne na chybná zadání;
- nechat si opakovaně **potvrdit závažná a nebezpečná rozhodnutí** *(o výmazu informací, o nevratných změnách v údajích apod.)*;
- neumožnit **nevhodnou kumulaci** příkazů tam, kde by mohlo dojít k nedorozumění; pokud je před akcí vymáháno více odpovědí, je nutno jednoznačně dát najevo, co znamenají jejich kombinace; umožnit nevyplnění některých odpovědí a předem definovat, co to znamená *(například opakovaný dotaz „setřídit dle:“ jméno, „setřídit dle:“ katedra ... platí poslední údaj?, platí všechny zadané údaje? apod.)*.

Řešený příklad - Projekt – Návrh komunikace

Když máme jasno o funkčnosti IS Knihovna, navrhneme uživatelský vzhled jeho implementace a necháme odsouhlasit zadavateli.

Shrnutí pojmů 2.

Analýza datová, konceptuální schéma, ERD, datový slovník.

Analýza funkční, kontextový diagram, diagram datových toků DFD, minispecifikace funkcí, strukturovaný jazyk.

Analýza dynamická, stavový diagram STD.

Komunikace s uživatelem, menu systém, vstupní formuláře, výstupní sestavy, nápovědy.

Otázky 2.

1. Co je analýza informačního systému?
2. Které typy analýzy IS rozeznáváme?

3. Co analyzuje datová analýza IS a jaké k tomu používá nástroje?
4. Co analyzuje funkční analýza IS a jaké k tomu používá nástroje?
5. Co je strukturovaný jazyk a k čemu je určen?
6. Jaká jsou pravidla pro zápis algoritmů strukturovaným jazykem?
7. Co analyzuje dynamická analýza IS a jaké k tomu používá nástroje?
8. Co je komunikace programu s uživatelem a jaké k tomu používá nástroje?
9. Jaká pravidla má dodržovat komunikace s uživatelem?
10. Jaké jsou zásady pro ošetření uživatelových chyb?

Úlohy k řešení 2.

Z úloh 1. kapitoly nebo následujících úloh si vyberte 2 a zpracujte pro ně zadání a úplnou analýzu.

Nejprve napište zadání slovně, zadejte seznam událostí a reakcí systému, nakreslete kontextový diagram a přiřaďte jednotlivé akce systému jednotlivým aktérům.

Dále proveďte datovou analýzu a výsledek popište jako úplné konceptuální schéma (lineární zápis tabulek databáze, ERD a datový slovník, integritní omezení).

Potom proveďte funkční analýzu, rozkreslení DFD alespoň nulté a nejnižší elementární úrovně a napište minispecifikace některých funkcí strukturovaným jazykem.

Na závěr vykreslete stavový diagram IS a některých jeho entit.

1. Zahradnictví **Zahrada** potřebuje evidovat své zakázky. Sleduje informace o nabízených rostlinách (katalogové číslo, název český, název latinský, nepovinný popis, cena) - v katalogu mohou být i dosud nepoužité rostliny. Dále eviduje zákazníky, kterým vytvořila alespoň jednu zahradu (jméno, adresa, telefon) a realizované zahrady (zákazník, adresa_zahrady, datum_realizace, číslo_zahrady, seznam použitých rostlin, jejich počet a druh). Mimo základní práce s databází, jako nové záznamy, jejich úpravy a rušení, je s daty třeba provádět následující operace: vytvoření nové zakázky se seznamem objednaných rostlin, výpočet a tisk závěrečného účtu za vytvořenou zahradu, výpis katalogu nabízených rostlin.
2. Navrhněte strukturu databáze pro informační systém ABC soukromého **zdravotnického střediska** s několika lékaři. Je potřeba evidovat lékaře (osobní údaje a specializaci), pacienty (osobní údaje, pojišťovna), objednané pacienty a uskutečněné návštěvy u lékaře i lékařů u pacientů (datum a čas, diagnóza, výkon lékaře, cena pro pojišťovnu). Mimo základní funkce manipulace s databází je třeba posílat měsíční výpisy výkonů příslušným pojišťovnám a zaznamenávat jejich proplacení.

Příprava na tutoriál - Zadání semestrálního projektu

Pro vlastní úlohu informačního systému proveďte úplnou analýzu datovou a funkční a alespoň jeden stavový diagram pro důležitou entitu.

Dále navrhněte uživatelský vzhled celé aplikace.

3. NÁVRH IMPLEMENTACE INFORMAČNÍHO SYSTÉMU

Čas ke studiu kapitoly: 16 hodin (4 x 2 hodiny + 4 x 2 hodiny řešení úloh)

Cíl V této kapitole se dozvíte

- co všechno patří do etapy návrhu implementace informačního systému,
- jakými nástroji se provádí návrh,

a budete schopni

- provést návrh implementace menšího informačního systému,
- provést.indexovou analýzu informačního systému,
- navrhnout systémová data a systémové funkce pro zabezpečení všech dalších úloh návrhu IS,
- provést.transakční analýzu funkcí informačního systému,
- s dodržením dvoufázového protokolu navrhnout zámky objektů databáze a zabezpečit tak sériovost transakcí,
- rozpoznat, je-li možné, že dojde k uváznutí při paralelním běhu 2 transakcí
- navrhnout pro všechny funkce vhodné modulové schéma.

Výklad

3.1. Obsah a dělení etapy návrhu implementace

❑ Co je návrh implementace

Výsledkem analýzy je několik modelů budoucího systému. Ty popisují, **co** se bude v IS evidovat a **co** se bude s daty dělat. Všechny modely popisují věcně budoucí IS a prozatím nezohledňují ani použitý HW a SW, ani organizační, ekonomické, časové a další záležitosti. Tato nezávislost analýzy na budoucí implementaci má mj. výhodu v tom, že je možná implementace v jakémkoliv prostředí – například při potřebě přechodu stejného IS na jiný SW.

V etapě návrhu implementace se upřesňuje, **jak** se to vše bude dělat. Řeší se jednak další podrobnosti v již zpracovaných modelech, aby implementace měla optimální vlastnosti, jednak se nyní berou v úvahu dosud nepoužité nefunkční požadavky ze zadání.

Vstupem pro návrh je

- výsledek analýzy = data, funkce, stavy; odtud se doplní další systémové funkce, další data,
- návrh komunikace, ovládání, formáty; odtud další systémové funkce,
- nefunkční požadavky; odtud návrh HW, SW prostředí, zohlednění legislativy, smluvní podmínky projektu.

Výstupem návrhu bude

- detailní zadání pro rutinní implementaci = definice databáze, úplné algoritmy funkcí

Dělení návrhu

U velkých IS se v etapě návrhu implementace řeší úlohy 2 úrovní:

- **systémový návrh =** koncepce řešení, návrh HW a SW prostředí, harmonogram, cena apod.
- **vlastní návrh implementace** = upřesnění, doplnění a optimalizace algoritmů, doplnění dat, rozdělení funkcí do modulů.

3.2. Systémový návrh

Úvodní koncepční část vychází z výsledků analýzy a z nefunkčních požadavků zadání. Z analýzy je jasný rozsah systému a jeho dělení na funkční celky – subsystémy.

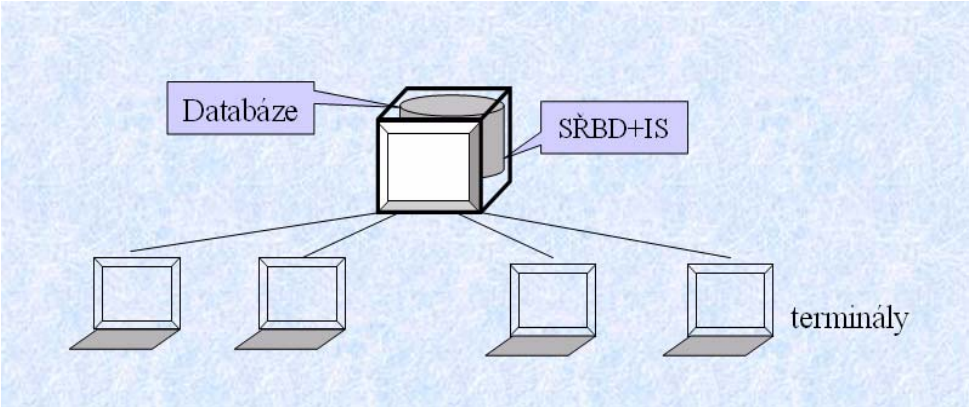
Z nefunkčních požadavků se dále berou v úvahu

- případné požadavky na **použitý HW a SW** prostředí; pokud není v požadavcích zadavatele, pak rozhodnutí o HW a SW;
- požadované prioritní **vlastnosti** výsledného IS *(zda je pro zadavatele nejdůležitější rychlost zpracování nebo nejvyšší zabezpečenost dat nebo nejrychlejší provoz apod.)*; odtud budou brány požadavky na optimalizaci algoritmů z funkční analýzy nebo úprava harmonogramu, bezpečnostních funkcí apod.;
- zařazení do **kontextu ostatních IS**, které jsou aktéry budovaného systému: definování vstupních a výstupních formátů, dohody na předávání dat;
- kontrola, doplnění a zpřesnění požadavků na potřebnou **legislativu** *(dodržování zákonů a vnitrofiremních směrnic, zohlednění v odpovídajících algoritmech),*
- návrh **architektury IS** - rozložení dat databáze i instalovaného SW na jednotlivá média *(na jednotlivé servery nebo dokonce jednotlivé uzly distribuované sítě),*
- **harmonogram** zpracování *(vybuduje se celý IS a pak zprovozní nebo se bude realizovat a zavádět do provozu po subsystémech nebo se některé části systému nakoupí, protože jsou na trhu SW nabízeny apod. – odtud konkrétní časový plán);*
- stanovení **ceny** systému, uzavření smlouvy.

❑ Architektury informačních systémů

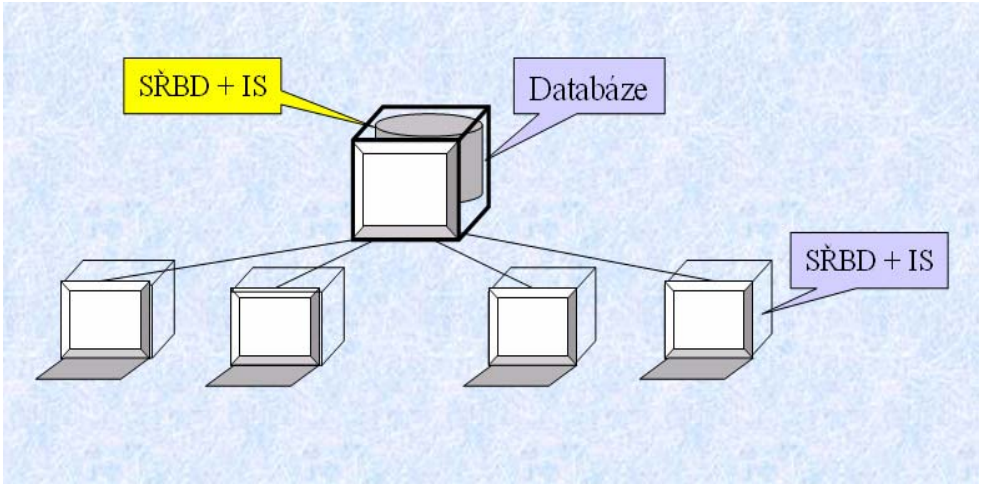
Podle použitého typu SŘBD, případně dalších SW prostředí, podle rozmístění částí IS a podle toho, kde probíhá zpracování dat obvykle rozlišujeme následující architektury výsledného IS:

1. **Architektura centrální**



Systém je zabezpečen centrálním počítačem s terminály (obvykle „neinteligentními“ konzolami tj. bez vlastního procesoru, paměti a vlastní aplikace). Většinou to jsou terminálové sítě na velkém centrálním počítači, ale mohou to být i systémy na bázi PC s centralizovaným řízením. Databáze, SŘBD i IS jsou umístěny v centrálním počítači. Aplikace i zpracovávaná data se volají z terminálů, zpracování všech úloh probíhá na počítači. Jediný počítač je silně zatížen, proto toto řešení nebývá používáno u aplikací pro velký počet uživatelů. V současnosti je těchto aplikací již velmi málo.

2. Architektura file – server



Klasická lokální síť pro aplikace menšího rozsahu a malý počet uživatelů. Tvoří ji HW server, na který je připojeno několik PC jako pracovní stanice.

Databáze je umístěna na HW serveru (proto file-server), aby k ní měli přístup všichni uživatelé. SW = SŘBD i aplikační úlohy mohou být instalovány na každém PC (což má výhodu při práci, protože se programy nepřenášejí, ale nevýhodu při všech změnách SW, protože se musí provádět aktualizace na všech stanicích). Nebo jsou SŘBD i aplikace umístěny na HW serveru a při práci se volají z jednotlivých stanic (výhoda při změnách SW, protože se provádí na jediném místě, nevýhoda při běžné práci, protože se každý používaný modul stále přenáší ze serveru na PC). Aplikace zpracovávají vstupy uživatelů, výstupy na obrazovku i přístup k datům na disku.

Možnosti SŘBD jsou velké, aplikace flexibilní, ale rychlost přenosů dat, bezpečnost dat a zabezpečení integrity dat jsou sníženy. Všechny aplikace musí mít naprogramován víceuživatelský přístup k datům, obvykle pomocí zamykání nebo transakcí. Většinou je používán relační model dat, který někdy umožní i přístup k datům mimo aplikační úlohy, a tak není možno zaručit integritu dat. Pro větší počet uživatelů a rozsáhlé databáze klesá výkonnost a stoupá komplikovanost systému.

Aplikace zpracovávají vstupy uživatelů, výstupy na obrazovku i přístup k datům na disku.

Příklad:

Představme si jednoduchou databázovou úlohu – nalezení platu pana Nováka v databázové tabulce Zam, a to ze stanice A. Po SQL příkazu

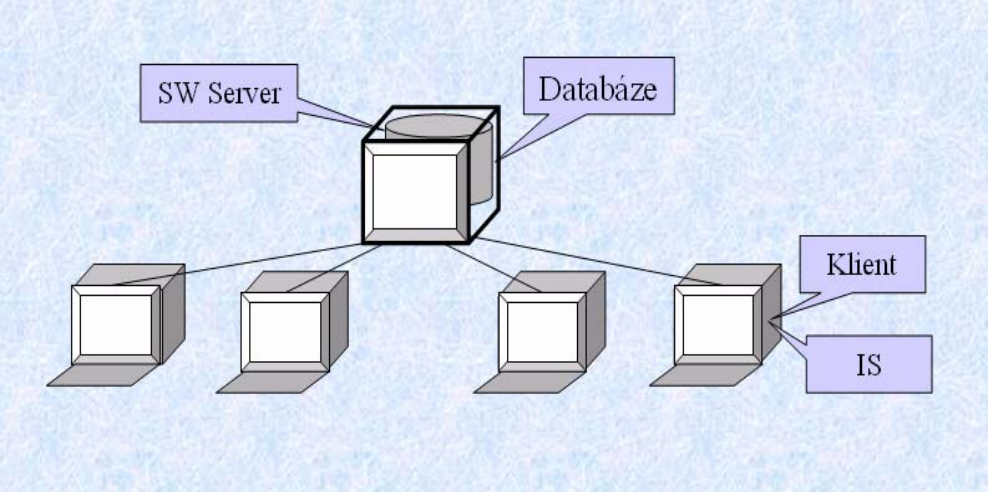
SELECT plat FROM Zam WHERE jmeno = „Novák“;

se provádí následující akce (představme si pro větší efekt, že se v tabulce Zam hledá sekvenčně): aplikace ze stanice A vyhledá na serveru v Zam 1. záznam, přenese na A, tam otestuje podmínku jmeno = „Novák“, zjistí neshodu; požádá o 2., 3., atd. záznam, dokud není nalezen hledaný záznam. Teprve teď v nalezeném záznamu najde hodnotu plat a zobrazí jej uživateli. Mezi serverem a stanicí tak proteče mnoho zbytečných dat.

U příkazu „setřiď tabulku Zam“ je situace ještě zřetelnější: tabulka se přenese na stanici, tam setřídí, výsledek se přenese zpět na server. Oba přenosy tabulky jsou zbytečné, kdyby třídění mohlo proběhnout na serveru, odpadly by.

♦

3. Architektura klient - server



Hlavní nevýhodou minulé architektury byly zbytečné přenosy dat mezi serverem a pracovní stanicí. Tu odstraňuje architektura klient – server.

Princip spočívá v tom, že dosavadní integrovaný SŘBD, obsahující jak databázové operace, tak prostředí pro vývoj aplikace a její spouštění, je rozdělen na 2 části, 2 spolupracující systémy:

SŘBD = Server + Klient

Klient se instaluje na PC, v něm je vytvořena aplikace. Na datovém HW serveru je instalován SW server, který realizuje všechny databázové operace. Pokud aplikace potřebuje provést databázovou operaci, požádá SW server, ten ji provede na HW serveru bez zbytečného přenášení dat, po síti se přenášejí jen požadovaná data. Zpracování dotazů a přístupů do databáze si řídí server. Protokol mezi klientem a serverem je nejčastěji SQL, proto mohou mezi sebou komunikovat i různé SŘBD.

Databázový server může být umístěn na témže PC, jako klient, častěji však běží na samostatném počítači.

Příklad:

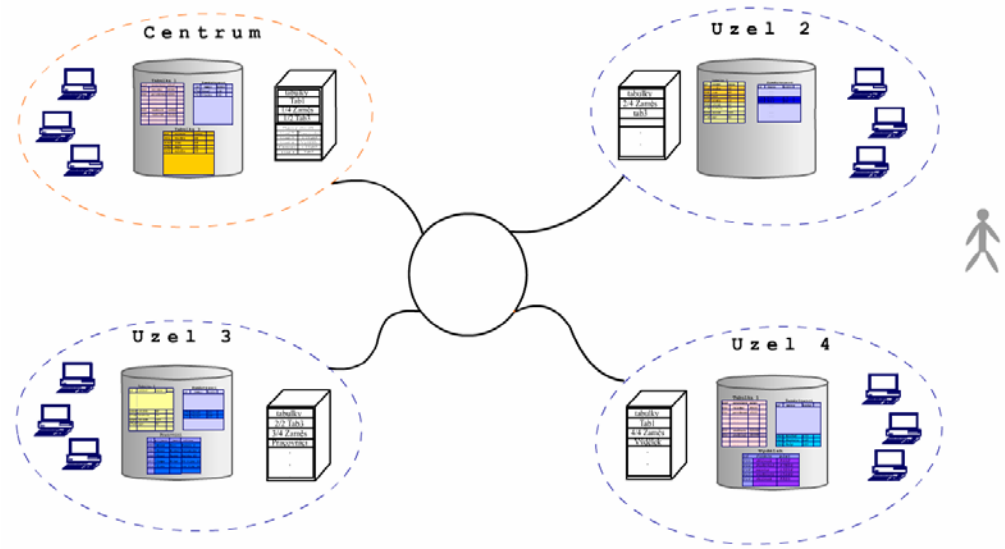
Mějme stejnou úlohu – nalezení platu pana Nováka v databázové tabulce Zam, a to ze stanice A. Opět se v tabulce Zam hledá sekvenčně: aplikace ze stanice A požádá server o záznam s podmínkou jmeno = „Novák“; SW server na HE serveru vyhledá tento záznam a jen tento jediný pošle zpět na stanici A.

U příkazu „setřiď tabulku Zam“ stanice pošle příkaz k setříděn na server, ten příkaz vykoná přímo na HW serveru a zpět pošle jen oznámení o ukončení transakce.

♦

3. Distribuované databáze

Dosavadní typy vyžadují data soustředěná na jednom počítači. Některé rozsáhlejší aplikace mohou mít data **územně rozložená** na **lokální báze** dat nebo jsou databáze částečně nebo úplně **sdíleny** s jinými lokálními bázemi. Přitom každou lokální bázi zabezpečuje lokální IS. Pro vzájemné propojení bází a jednotný uživatelský přístup k nim je nutný další jednotící SW - distribuovaný databázový systém.
Jednoduché typy mohou pouze přenášet modifikovaná data mezi lokálními bázemi a udržovat je vzájemně aktuální. Často prostřednictvím centrálního počítače a např. po skončení práce s databází se všechny soubory aktualizují. To ale neřeší případ, kdy potřebná data pro aplikaci jsou na nepřístupném lokálním PC. Tyto i další problémy řeší distribuované zpracování. Uživatel požádá o data lokální počítač. Pokud tam nejsou, lokální PC zjistí po síti, kde data jsou, vyžádá je a předá uživateli, který ani nemusí vědět, odkud data pocházejí.
Takovéto distribuované IS jsou mnohem složitější a tedy náročnější na řešení i provoz, než klasické IS.



Podrobněji se seznámíme s architekturou distribuovaných databází a jejich novými problémy v kapitole 6.

3.3. Vlastní návrh implementace

Po koncepčních záležitostech pro řešení projektu se řeší vlastní návrh implementace, nazývaný též objektovým návrhem. Jeho úkolem je projít všechny modely z analýzy a doplnit je řadou implementačních detailů. Často se přitom doplňují i další data a další systémové funkce. Systémovými je zde nazýváme proto, že neřeší přímo některou z požadovaných uživatelských funkcí, ale zabezpečují jejich správné fungování a zabezpečují řadu dalších organizačních operací systému.

Co všechno tedy k návrhu implementace patří:

I. úprava a doplnění funkcí z funkční analýzy:

- **upřesnění algoritmů** minispecifikací,
- **optimalizace přístupů k datům, analýza časové a prostorové složitosti algoritmů**,
- **indexová analýza,**
- **řešení odvozených atributů,**
- kontrola **rozpoznatelnosti stavů** z dynamické analýzy,
- kontrola úplnosti **podmínek pro spouštění** jen přípustných funkcí podle stavů,
- **analýza podobnosti** (afinity) funkcí,
- **transakční analýza;** protože jde o rozsáhlou oblast, základní problémy transakční analýzy si uvedeme v následujících kapitolkách 3.3. – 3.5.,
- **návrh rozmístění dat**, pokud jde o distribuovaný IS; tato oblast bude probrána v kapitole 6;

II. zabezpečení **mezního provozu:**

- počáteční **instalace a inicializace databáze,**
- **ukončení práce se systémem** a úklid,
- **analýza zálohování dat,**
- **pád systému** a **obnova databáze po havárii,**
- **analýza archivování dat;**

III. doplnění systémových funkcí:

- doplnění dat i funkcí pro evidenci **rolí uživatelů, přidělování přístupových práv a jejich realizaci,**
- zařazení **evidence chyb** do IS,
- zařazení **evidence využívání funkcí** do IS..

Podívejme se na jednotlivé body podrobněji.

❑ Upřesnění algoritmů

V minispecifikacích nemusí být všechny detaily algoritmů dostatečně podrobně nebo přesně rozepsány. Například jsou uvedeny jen odkazy kontrol domén atributů na datový slovník, kontroly nebo výpočty jsou jen pojmenovány atd. Zde se navrhuje realizace kontrolních a výpočtových funkcí například formou triggerů, definováním procedur, doplňují se detailní vztahy nebo algoritmy pro výpočty.

Příklad:

Část minispecifikace Záznam nového pacienta *pro entitu*

Pacient (id_pac, rod_cis_pac, jmeno, adresa, datum_zapisu)

je formulována takto:

1. Zobraz prázdný formulář pro Pacient, vyplň automaticky **id_pac** a **datum_zapisu**
2. Uživatel zadá rod_cis_pac, jmeno, adresa

...

V návrhu se doplní (dle použitého SŘBD a podle toho, jestli podporuje datový typ serial = automaticky se inkrementující hodnotu atributu) způsob naplnění atributu id_pac *a systémovým datem se naplní* datum_zapisu.

♦

Příklad:

V minispecifikaci pro výpočet výplatní listiny je bod

x. Vypočti čistou mzdu, daň z příjmu, zdravotní a sociální pojištění z aktuální hrubé_mzdy.

Bod se musí doplnit o vzorce pro výpočet pojištění a daně. Protože se ze zákona pravidla pro jejich výpočet čas od času mění, je vhodné definovat nějaké globální proměnné – například do pomocné systémové tabulky s aktuálními hodnotami konstant – procenta pro výpočet pojištění a daně. Ty se v rámci tohoto bodu musí načíst (zřejmě před cyklem pro výpočet mzdy pro konkrétní zaměstnance), dosazovat do příslušných vzorců a počítat uvedené hodnoty. Tedy se upraví celý algoritmus výpočtu výplat.

♦

Může se stát, že některé funkce budou obsahovat složitější zpracování údajů a budou v průběhu tohoto zpracování vytvářet pomocné **dočasné tabulky**, které budou mít jinou strukturu, než dosud definované tabulky databáze. V tom případě se dočasné tabulky pojmenují, definuje se jejich struktura a vše se **přidá do datového slovníku**.

Tyto tabulky buď jsou „vnitřní“ součástí jedné elementární funkce, pak se neobjeví v DFD, ale jen v minispecifikaci. Nebo dočasnou tabulku vytvoří jedna funkce a jiná ji později použije. Pak by již měla být datovou pamětí i v DFD viditelnou. Zde jen zkontrolujeme, že je její struktura skutečně uvedena v datovém slovníku.

❑ Optimalizace přístupu k datům, časová a prostorová náročnost algoritmů

V minispecifikaci je obvykle formulace „vyhledej, změň nebo zruš záznamy splňující xxx“ bez detailů pro formulaci dotazu. Nyní je nutné formulovat SQL příkaz, který příkaz realizuje. Jde o zvážení optimální formulace dotazů, protože již víme, že mnohé dotazy lze formulovat několika způsoby a ne všechny jsou stejně rychlé. Je vhodné nejprve formulovat dotaz pomocí relační algebry (tím si uvědomit vhodné pořadí operací nad tabulkami, které je nutné s daty provádět) a pak teprve formulovat odpovídající dotaz SQL.

Příklad:

V databázi Knihovna jsou mimo jiné tabulky

Ctenar (id_ctenar, jmeno_ctenar, adr_ctenar, telef_ctenar, mail_ctenar)	s 300 záznamy
Titul (ISBN, nazev, *id_vydav*, rok, zanr, jazyk, zeme, anotace, cena)	s 10 000 záznamy
Rezervace (*ISBN_cis, id_ctenar*, dat_rezer, dat_vypuj)	s 500 záznamy
Exemplar (prir_cis, *ISBN*)	s 30 000 záznamy
Vypujcka (*prir_cis, id_ctenar*, dat_od, dat_do, dat_vraceno, pocet_upom)	s 200 000 záznamy

V minispecifikaci je příkaz

x. Vytiskni názvy a ISBN knih, které má rezervovány zadaný čtenář Xyz

Jde zřejmě o spojení 3 tabulek, protože ze zadaného jména určíme v tabulce Ctenar *id_čtenáře, v* Rezervace *jeho rezervovaná ISBN a v* Titul *jejich názvy. Předpokládejme, že výsledkem je 5 rezervovaných titulů.*

Odpovídající SQL dotaz může být formulován několika způsoby. Rozeberme si 2 z nich:

1. pomocí
```
SELECT ISBN, nazev
  FROM Ctenar C, Rezervace R, Titul T
  WHERE C.id_ctenar = R.id_ctenar AND R.ISBN = T.ISBN
        AND C.jmeno_ctenar = „Xyz“;
```

pro řešitele pohodlné řešení, ale při realizaci se provádí kartézský součin 3 tabulek => testuje se

300 x 10 000 x 500 =**1 500 000 000** *případů, z nich je výsledkem 5 záznamů.*

Asi si dovedeme představit, že i na rychlých počítačích bude takový dotaz trvat dlouho.

2. pomocí poddotazů

```
SELECT nazev
    FROM Titul
    WHERE ISBN IN (SELECT ISBN
                   FROM Rezervace
                   WHERE dat_vypuj NOT NULL AND
                         id_ctenar IN (SELECT id_ctenar
                                       FROM Ctenar
                                       WHERE jmeno_ctenar = „Xyz“)))
```

Pro řešitele jen málo složitější řešení, přitom počet testů (ve smyslu nejhoršího případu a sekvenčního hledání) je:

Z tabulky Ctenar *se hledá v 300 záznamech, výsledkem je jeden záznam. Ten se porovnává s 500 rezervacemi, výsledkem je hledaných 5 aktuálních rezervací. Ty se porovnávají s 10 000 záznamy titulů, výsledkem jsou úplné informace o titulech. Celkem je tedy*

300 + 500 + 5 x 10 000 = **50 800** *porovnání, z nich je výsledkem 5 záznamů.*

Rozdíl v počtu porovnávacích testů je zřejmý.

♦

Součástí optimalizace je tedy zvážení počtu průchodů tabulkou či počtu přístupů do databáze. Existují i případy, kdy je vhodné dokonce nepoužít SQL dotaz, ale naprogramovat vlastním algoritmem některé funkce, protože se může její provedení mnohonásobně zrychlit.

Příklad:

Uveďme si teoretický příklad pro realizaci spojení tabulek, a to na jednoduchém příkladě jen 2 tabulek, každá o 10 000 záznamech. Máme tabulky

X(a,b,c), Y(b,d,e)

Úkolem je realizovat jejich přirozené spojení

Z = X [*] Y

pomocí SQL se příkaz zapíše jednoduše

SELECT * FROM X, Y WHERE X.b=Y.b

ovšem tento příkaz je v SŘBD „přeložen“ do algoritmu, který obvykle bere v úvahu, zda jsou tabulky setříděny nebo indexovány podle společného atributu **b**.*Podívejme se na 3 modelové případy, kdy předpokládáme postupně setřídění žádné, jedné a obou tabulek podle* b.

1. Obě tabulky X, Y *nejsou setříděny ani indexovány podle* b. *Algoritmus je realizován přehledným dvojitým cyklem (v algoritmu je počet záznamů tabulek obecně* **m** *a* **n**).

PRO každý záznam z X dělej

```
  PRO každý záznam z Y dělej
    je-li X.b = Y.b PAK  ZOBRAZ  a, X.b, c, d, e
  KONEC CYKLU X
KONE CYKLU Y
```

*Časová složitost příkazu je m*n (pro každý řádek* X *se prochází všechny řádky* Y*)*

*Předpokládáme-li že 1 operace načtení a porovnání shody údajů trvá 0.001 sekundy, pak pro m = 10 000, n = 10 000 je celkový čas = 10000*10000=(100 000 000/1000) sec =* ***27 hod.***

2. *Tabulka* Y *je setříděna nebo indexována podle* b. *Pak pro každý řádek* X *se v tabulce* Y *hledá binárně odpovídající hodnota* b. *Časová složitost příkazu je* $m*\log_2 n$, *pro m = 10 000, n = 10 000 je celkový čas =* $(10000* \log_2 10000 / 1000)$ *sec =* ***20 min****.*

3. *Obě tabulky jsou setříděny nebo indexovány podle* b. *Pak o něco málo složitějším algoritmem je možno realizovat spojení takto:*

```
i=1; j=1
DOKUD  i<m
  DOKUD
     X.b <> Y.b  PAK  j=j+1;
  KONEC CYKLU
  jj=j
  DOKUD X.b = Y.b
        ZOBRAZ  b, X.b, c, d, e;
        j=j+1;
  KONEC CYKLU
  i=i+1;
  j=jj+1;
KONEC CYKLU
```

Časová složitost je m+n, pro m = 10000, n = 10000 celkový čas = (10000+10000)/1000 sec = ***20 sec****.*

Celkový rozdíl v čase pro stejnou úlohu dostatečně výmluvný.

Některé SŘBD mají vlastní optimalizátory SQL dotazů, které vyhledají nejlepší překlad, jiné ne. Někdy se tedy vyplatí často prováděné dotazy nad velkými tabulkami „naprogramovat" a nespoléhat na mechanický překlad SQL dotazu.

♦

S optimalizací přístupu k datům bezprostředně souvisí i několik následujících bodů.

❑ Indexová analýza

Indexová analýza znamená zvážení na základě analýzy jednotlivých minispecifikací, ke kterým atributům a ve kterých tabulkách bude vhodné vytvořit index. Tedy projdeme všechny minispecifikace pracující s některými databázovými tabulkami a zvážíme pro každý atribut použitých tabulek, jestli se podle něj

- hledá,
- pořizuje setříděný seznam – ve výpisu, v nápovědě apod.,
- realizuje spojení s jinou tabulkou,
- ověřuje jednoznačnost v téže tabulce,
- ověřuje existence v jiné tabulce.

Určíme četnost používání každé takové funkce a podle celkového výsledku navrhneme způsob indexování každého atributu (udržovaný, dočasný). Udržovaný index je stále aktuální, ale jeho údržba zatěžuje průběžně zpracování a zabírá kapacitu na disku. Neudržovaný index se vytváří až v okamžiku potřeby jej použít a po ukončení funkce se opět zruší. Rozhodnutí zřejmě závisí na tom, jak často se index využívá. K tomu je nutné projít všechny minispecifikace a rozhodnutí o existenci indexu i jeho typu provádět tak, aby vyhovovalo optimálně všem funkcím systému.

Někdy je vhodné realizovat index částečný (pokud to umožňuje SŘBD), jen na některé záznamy celé tabulky. Takový index je menší – má méně záznamů a tedy se v něm o to rychleji vyhledává.

Informace o navržených indexech udržovaných doplníme do datového slovníku, protože tak se vytvoří současně s definicí tabulky. Informace o dočasných indexech doplníme do příslušných minispecifikací, v nichž se index použije: tam se doplní příkazy CREATE INDEX a DROP INDEX.

Účinnost existujícího indexu jsme viděli v předcházejícím odstavci o optimalizaci dotazů.

Příklad:

Existuje tabulka faktur

Faktura_prijata (cis_fakt, rok_fakt, typ, ci_objed, dodavatel, fakt_dodavatele, dat_vyd, ter_splat, cena, proc_DPH, DPH, suma, zaloha, k_uhrade, dat_prik, dat_zapl, storno, dat_stor, odvod_DPH)

a existuje minispecifikace Nová faktura. *V ní provedeme následující úvahy:*

cis_fakt … *inkrementální hodnota, automaticky setříděno*

dodavatel … *nabídka při vyplňování* ***dle abecedy****, indexována tabulka* Dodavatel *podle názvu firmy*

fakt_dodavatele … ***kontrola na jednoznačnost*** *v tabulce* Faktura_prijata *pro téhož dodavatele, indexována podle* {**dodavatel, fakt_dodav**}

zaloha … *ověření zálohy dle čísla objednávky, jen pro* typ = zalohova, *index podle* {**dodav, ci_objed**} *částečný pro* typ = zalohova.

Protože se fakturování provádí denně mnohokrát, všechny indexy budou udržované.

♦

Příklad:

V minispecifikaci Měsíční seznam faktur *se vypisuje seznam faktur setříděný podle* **typ, dodavatel, fakt_dodavatele.** *Protože jde o využití funkce jednou měsíčně, bude složený index* **{typ, dodavatel, fakt_dodavatele}** *dočasný a do této minispecifikace se na začátek doplní příkaz* CREATE INDEX ... *a na konec* DROP INDEX.

♦

❑ Odvozené atributy

Někdy se v databázi potřebují nejen prvotně uložené údaje, ale i údaje z nich vypočtené či jinak odvozené. Nyní tedy jde o zvážení, zda se budou vypočtené atributy jednorázově počítat a ukládat nebo se budou po případných změnách zdrojových dat přepočítávat periodicky, případně jestli je nebudeme ukládat v databázi, ale použijeme explicitní příkaz (trigger, funkci, proceduru) pro jejich výpočet vždy až v okamžiku jejich použití.

Příklad:

U pacientů se eviduje rodné číslo. Z něj je možno spočítat datum narození a pohlaví, z data narození a aktuálního data při návštěvě u lékaře také věk pacienta.

Otázkou je, zda se tyto odvozené údaje budou počítat a ukládat jednou při zadání rodného čísla, nebo se budou periodicky přepočítávat a ukládat, nebo se nebudou ukládat a spočítají se až v okamžiku (v té funkci), kdy budou potřebné.
Předpokládejme, že v minispecifikacích se jen u dvou málo používaných funkcí vyskytuje datum narození (většinou stačí rodné číslo), často se však pořizují výpisy podle pohlaví. Věk pacientů se používá často pro různé statistiky i jiné funkce při rozhodování o způsobu léčby.
Rozhodneme se tedy následovně: datum narození se ukládat nebude, ale bude definována procedura pro jeho výpočet, použitá u zmíněných 2 funkcí. Pohlaví se vypočte jednorázově po uložení rodného čísla. Věk se však ročně mění a pacienti mohou být evidováni dlouhodobě. Proto bude optimální věk poprvé spočítat a uložit po záznamu nového pacienta a pak jednou ročně (například při roční uzávěrce a přechodu na nový rok) přidat funkci aktualizace atributu věk.

♦

❑ Kontrola rozpoznatelnosti stavů

V dynamické analýze se formulují podmínky, podle nichž se rozeznají definované stavy. Je-li tam vše správně a úplně provedeno, jsou doplněny tabulky o případné další „stavové“ atributy. Je vhodné ještě ověřit, že jsou definovány všechny funkce, které realizují akce přechodů mezi stavy. Pokud ne, zde se doplní, zvláště v možných krajních a dosud nezformulovaných situacích (viz též bod řešení mezních podmínek provozu).

Příklad: kontrola rozpoznatelnosti stavů

Minispecifikace Měsíční účet pojišťovně *z tabulky* Pacient *a ze zjednodušené tabulky*

Navsteva (rod_cis_lek, rod_cis_pac, cis_pojist, datum, id_vykonu, cena_vykonu)

je dosud formulována:

1. Zobraz dotaz uživateli:

Zadejte měsíc pro účtování [mm.rrrr]:	

2. Uživatel zadá měsíc
3. Vyber z Navsteva záznamy daného měsíce
4. Setřiď vybrané záznamy podle cis_pojist, rod_cis_pac, datum
5. Vypiš vybrané záznamy ve formátu

```
Pojišťovna: název
        Pacient: rod_cis_pac    jmeno   ¨
                        datum           id_vykonu       cena_vykonu
                        ...
        Pacient: rod_cis_pac    jmeno   ¨
                        datum           id_vykonu       cena_vykonu
                        ...
                ------------------------------------------------------------

        Suma                                            celkova_suma
Pojišťovna: název
        Pacient: rod_cis_pac    jmeno   ¨
                        ...
                ------------------------------------------------------------

        Suma                                            celkova_suma
```

Bod 3 předpokládá, že při této funkci bude vždy 100% úplný seznam návštěv zapsán do databáze. Může se však stát, že z jakýchkoliv důvodů bude nějaký záznam zapsán dodatečně a pak by takové návštěvy zůstaly nevyúčtované. Proto se upřesní bod 3 o formulaci „+ nezaúčtované“. Aby se rozpoznaly dosud nezaúčtované záznamy, bude nutné přidat do tabulky Navsteva *další logický atribut, například nazvaný* uctovano.

Poznamenejme, že při procházení jinou minispecifikací Platba od pojišťovny *bude dále potřeba zaznamenat, které návštěvy už byly proplaceny. Pak buď dodáme další atribut logický atribut* zaplaceno, *nebo změníme atribut* uctovano *na atribut* stav *a budeme v něm zaznamenávat hodnoty např. 0 = dosud neuskutečněná objednaná návštěva, 1 = uskutečněná, 2 = vyúčtovaná, 3 = zaplacená pojišťovnou. Obdobné doplnění atributu by mělo vyplynout z kontroly stavového diagramu entity* Navsteva.

Dále v bodu 3.ani 4. není uvedeno, jak se vybrané záznamy setřídí. Zřejmě by nebylo vhodné třídit celou velkou tabulku Navsteva, *proto záznamy vybereme do nové dočasné tabulky* Ucet. *Do ní můžeme současně doplnit z číselníku pojišťoven jejich názvy a z číselníku výkonů ceny výkonů. Proto body 3.-5. zpřesníme takto:*

3. Vyber z Navsteva záznamy daného měsíce **+ nezaúčtované minulé a zapiš je do tabulky Ucet (**rod_cis_lek, rod_cis_pac, jmeno, cis-pojist, nazev_pojis, datum, id_vykonu, cena_vykonu)
4. Setřiď vybrané záznamy podle cis_pojist, rod_cis_pac, datum
5. Vypiš Ucet ve formátu .....

♦

❑ Kontrola úplnosti podmínek pro spouštění přípustných funkcí podle stavů

V dynamické analýze byly definovány stavy systému a entit. Je nutné ještě propojit funkční a dynamickou analýzu - definovat, které funkce (minispecifikace) se mohou používat ve kterém stavu systému a které ne. Máme-li tento seznam funkcí povolených v konkrétních stavech hotov, je třeba zvolit nějaký systém kontroly, že se nepovolené funkce nebudou uživateli nabízet v menu nebo se nebudou spouštět s upozorněním uživateli. Tato kontrola je buď součástí řídicího programu IS, nebo jsou kontroly součástí minispecifikací. V obou případech je třeba tyto kontroly realizovat – doplněním kontrolní systémové funkce do řídicího programu nebo doplněním kontroly na začátku všech odpovídajících minispecifikací.

Příklad:

Funkci Zaplacení faktury *dává smysl spouštět jen pro faktury dosud nezaplacené. Jde o stav jednotlivých entit, proto není možné zakázat celou funkci, ale jen některé záznamy z tabulky* Faktura. *Stav faktury* nezaplacená *poznáme podle hodnoty atributu* datum_zaplac *(je NULL nebo není). Proto na začátek minispecifikace této funkce přidáme podmínku*

1. podm=“datum_zaplac IS NULL“

a dále se tato podmínka přidá k příkazu

2. zobraz seznam faktur ze souboru Faktura splňujících podm

Zbytek minispecifikace je stejný.

♦

❑ Analýza afinity

Praktickým hlediskem, jak se budou programovat jednotlivé funkce systému, se zabývá analýza afinity = podobnosti funkcí. Často je v IS řada funkcí velmi podobná. Například formuláře pro záznam

nových entit, jejich různé modifikace nebo rušení, mají všechny zobrazen stejný formulář, ale pokaždé se v něm vyplňují jiné hodnoty atributů. Bylo by zbytečné je programovat několikrát jen s malými rozdíly funkčnosti. Vhodnější je rozpoznat tuto podobnost, naformátovat formulář jako jednu proceduru a uvnitř rozlišit, kterou funkci právě bude realizovat. Toto řešení je vhodnější i v případě změn, prováděných ve společné části – změna se provede na jednom místě a ne ve všech podobných funkcích.

Jiný častý případ podobných funkcí je u reportů nebo u některých výpočtových funkcí se společnými mezivýsledky a jiným zakončením. Výsledkem je návrh společných procedur pro implementaci těchto podobných funkcí.

Příklad:

V IS Sklad materiálu jsou tabulky

Sklad(cis_karty, nazev_mater, cena_jedn, mnozstvi, ..., cena)

Pohyb(cis_karty, datum, typ_zmeny, mnoz_zmeny, id_zakazka, ...)

kde typ_zmeny je O = počáteční stav roku, P = příjem, V = výdej, R = vrácení.

Mezi funkcemi tohoto IS jsou také:

Nová skladová karta... *pro nově evidovaný materiál se vyplní* cis_karty, nazev_mater, cena_jedn; *do* Pohyb *se zapíše nový záznam s množstvím 0;*

Příjem na sklad ... *na vybranou kartu se vyplní pomocná hodnota* datum *a* mnoz_prijate*, to se přičte k aktuálnímu množství a přepočte se* cena*, do* Pohyb *se zapíše nový záznam;*

Výdej ze skladu ... *obdobně jako příjem, jen množství se odečte a zapíše se číslo zakázky, pro kterou se materiál vydává;*

Vrácení zboží do skladu ... *obdobně jako příjem, ale jinak se zaznamená* typ_změny.

Všechny tyto funkce používají stejný formulář pro kartu materiálu, při každé funkci se ale vyplní jiné atributy a trochu jinak se zapíše záznam do Pohyb*. Místo 4 funkcí se tedy navrhne jedna, která nejdříve zobrazí formulář pro kartu materiálu, potom se rozvětví podle 4 funkcí, v rozvětvení se naplní hodnoty pro nový záznam do Pohyb, nakonec se tam opět pro všechny funkce stejně zapíše nový záznam.*

♦

❑ Mezní provoz

Většina funkcí z analýzy se týká ustáleného procesu zpracování, běžných uživatelských funkcí. Mimo ně se však musí zpracovat i návrh tří základních mezních provozů – inicializace, ukončení a pádu systému, případně některých dalších mezních situací. Jejich základní typy probereme v samostatných bodech.

Stane se, že při funkční analýze se zapomene na některé okrajové situace, které se nevyskytují často, ale jejich zabezpečení je nutné. Ukážeme si je na příkladech, ale obdobných i dalších situací se může vyskytnout mnoho.

Příklad:

Ve firemním systému je řada tabulek s autoinkrementálními klíči. Jejich počáteční hodnota však je odvozena od kalendářního roku. Například číslování faktur (číslo faktury je jediné číslo, používané jako variabilní symbol při platbě bankovním převodem) je definováno jako

RRRRnnnni

kde RRRR je aktuální rok a nnnn je vlastní inkrementální číslo jednoznačné v rámci roku. Na začátku každého kalendářního roku je nutné nastavit počítadla na novou počáteční hodnoty (je to jednorázově provedeno a je to lepší řešení, než pro každou fakturu „vypočítávat" rok a k tomu inkrement).

Podobně mohou být číslovány zakázky, objednávky, pokladní a účetní doklady atd.

Aby nemusel správce databáze nastavovat počitadla ručně, je vhodné ***vytvořit systémovou funkci****, která to provede automaticky k 1.1. nebo na příkaz správce.*

♦

Příklad:

Jiná situace je například ve firmě na přelomu každého měsíce. Přicházející faktury jsou zařazovány do měsíce, kdy byly vystaveny a na konci měsíce je z nich vyúčtováno DPH na finanční úřad. Ty, které chodí poštou, mohou mít několik dnů zpoždění a tak se může stát, že přijdou až po odeslání vyúčtování. Je tedy nutné nejen rozlišit faktury podle měsíce, ale také podle toho, zda už byly vyúčtovány finančnímu úřadu. Toto vyúčtování nesouvisí se zaplacením faktury a tedy se mohlo zapomenout na evidenci tohoto ***dalšího stavu*** *faktury. Pokud stav není rozlišitelný dosavadními atributy, doplníme jej.*

♦

❑ Instalace IS a inicializace databáze

Poslední součástí IS jsou programy pro počáteční instalaci IS a inicializaci databáze.

Instalační program se realizuje podle možností konkrétního SŘBD.

Mimo to je však zapotřebí napsat funkce pro definování struktury databáze (CREATE DATABASE a řada příkazů CREATE TABLE) s případným naplněním některých tabulek *(číselníků apod.)*. V kapitole o předání IS do provozu se k tomuto bodu ještě vrátíme a popíšeme si podrobněji postup inicializace a naplnění databáze.

Příklad:

Zpracováváme IS ABC soukromého zdravotnického střediska, kde jsou definovány tabulky

Lekar (RC_L, jmeno_L, spec)
Pacient (RC_P, jmeno_P, adresa_P, id_pojis)
Navsteva (id_navst, RC_L, RC_P, datum, hodina, id_diag, id_vykon)
Cisel_vykonu (id_vykon, cena)
Cisel_diagnoz (id_diag, diagnoza)
Cisel_pojistoven (id_poj, pojistovna)

Po instalaci SŘBD a aplikačního programu je třeba založit databázi, vytvořit tabulky a naplnit obecně platné číselníky pomocí připravené inicializační sekvence příkazů

```
CREATE DATABASE ABC
  CREATE TABLE Lekar ( ... )
  .....
  CREATE TABLE Cisel_pojistoven ( ... )
  INSERT INTO Cisel_vykonu SELECT * FROM Vykony
  .....
  INSERT INTO Cisel_pojistoven SELECT * FROM Pojistovny
```

kde Vykony, ..., Pojistovny *jsou předem připravené nebo i upravené celostátní číselníky.*

Ostatní tabulky zatím neobsahují žádná data, takže je třeba ověřit, jestli na začátku nebudou některé funkce s nimi pracující havarovat. Například pokud by uživatel začal zapisovat do tabulky Navsteva, pokud by tabulky Lekar nebo Pacient byly prázdné. Jde opět o otestování jistého mezního stavu databáze.

♦

❑ Ukončení práce se systémem a úklid dat

Když je hotova analýza záloh a archivace, musíme vytvořit funkce, které to budou realizovat.

Pokud se s IS pracuje „během pracovní doby“ a pak se vypíná, je vhodné zařadit před ukončením, po skončení práce posledního uživatele, další systémové funkce, jako zálohování databáze, archivaci a „úklid“ databáze = výmaz neplatných záznamů, reindexace neudržovaných indexů apod. Zřejmě stav celého systému se přepne stavu „úklid“, který bude nepřístupný běžným uživatelům a přístupný případně jen správci databáze.

Pokud je IS v provozu nepřetržitě, je vhodné odhadnout dobu s nejnižším provozem a do ní zařadit spuštění záloh, archivací, úklidu. Se souběhem běžného provozu a těchto funkcí ale souvisí některé další problémy, které probereme až v kapitolách o transakční analýze.

Do tohoto bodu patří také návrh opatření a realizace funkcí pro obnovu databáze po pádu systému. Pokud náš SŘBD má programy pro obnovu databáze z poslední zálohy a log souboru (co to znamená probereme v kapitolách o transakcích), pak se jen navrhnou funkce pro jejich spouštění. Pokud to SŘBD nemá, je nutno tyto funkce navrhnout a realizovat.

Příklad:

V menu IS Knihovna je jedna z voleb nazvaná například „Konec“. Do této funkce můžeme zařadit postupně posloupnost několika elementárních funkcí: Uklid, Archivace, Zaloha.*a spouštět je tak automaticky. Pak každá z těchto funkcí má v sobě zabudované podmínky (plynoucí z příslušných analýz), pro které tabulky nebo entity se tentokrát provádějí.*

♦

Příklad:

Jiná možnost je vytvořit samostatný systém pro správce databáze a provedení těchto jednotlivých funkcí úklidu, archivace, záloh nechat na správci.

♦

❑ Analýza zálohování databáze

Ne všechny databázové tabulky je nutné zálohovat se stejnou periodou. Je nutné provést analýzu toho, jak často se mění obsah tabulek a podle toho navrhnout periodu jejich zálohování nebo rozpoznání stavů, kdy se mají zálohovat. K zálohování se vytvoří další systémové funkce. Spouští se buď automaticky například při ukončení práce s IS nebo v definovaný čas *(denně o půlnoci apod.)*, nebo jsou spouštěny na příkaz správce databáze.

Perioda zálohování nebo definování stavů, kdy se mají jednotlivé tabulky zálohovat, se určuje **podle rychlosti změn v tabulkách**. Zřejmě mnohé číselníky, které se nemění téměř vůbec, stačí zálohovat jednou. Statická data, jako různé seznamy *(studentů, zaměstnanců, vyučovaných předmětů, čtenářů knihovny apod.)* se mění zřídka a stačí je zálohovat buď vždy po (málo časté) změně, nebo pravidelně v řídkých intervalech *(studenty po semestru, zaměstnance měsíčně, předměty ročně, čtenáře knihovny po změně apod.)*. Konečně dynamická data, která se mění nebo doplňují často, je třeba zálohovat podle okolností s krátkou periodou *(příjem a výdej materiálu na skladě denně, výpůjčky knih denně, účetní operace v bance i několikrát denně, výsledky zkoušek studentů během semestru vůbec, ve zkouškovém období denně apod.)*.

Příklad:

V IS Knihovna jsou tabulky Ctenar, Titul, Exemplar, Autor, Napsal, Vypujcky, Rezervace, Vydavatel, Objednavky.

Knihy se nakupují přibližně jednou týdně a do IS zapisují během několika dnů po nákupu. Není tedy přesný den, kdy je přírůstek knih zapsán do evidence, proto se budou tabulky Titul *a* Exemplar *zálohovat vždy po změně. K tomu musíme rozpoznat, jestli došlo ke změně nebo ne – definovat další stav systému. Totéž bude platit pro tabulky* Autor *a* Napsal, *protože jen při příjmu nových knih může přibýt nový autor a záznam o tom, co napsal.*

Tento stav vyjmenovaných tabulek je nutné nějak rozpoznat a zatím není rozpoznatelný podle hodnot existujících atributů. Proto například založíme novou systémovou tabulku Stav_tabulek *(systémovou, protože slouží jen pro zabezpečení správných funkcí systému, neeviduje uživatelská data) a do ní uděláme záznam o stavu tabulek. Schéma tabulky bude například*

Stav_tabulek (tabulka, stav) ... stav = 0 znamená beze změny, stav = 1 byla změna

a do minispecifikací pro záznam nových entit nebo modifikaci existujících entit přidáme na konec příkaz

x. Nastav v tabulce Stav_tabulek pro tabulka = „xxx“ hodnotu stav = 1

kde „xxx“ je název měněné tabulky.

Současně na konci funkce pro zálohování této tabulky dáme podobný příkaz měnící zpět stav = 0. *Obdobně můžeme řešit změny v tabulkách* Čtenář, Vydavatel, Objednávky.

Ovšem tabulky Vypujcky a Rezervace *se mění denně, proto budou zálohovány pravidelně denně.*

♦

Příklad:

U IS Praktický lékař jsou mimo jiné tabulky – číselníky léků a jejich cen, lékařských výkonů a jejich ohodnocení. Oba číselníky vydává ministerstvo zdravotnictví a jen občas se mění. Takové číselníky není nutné zálohovat vůbec, v případě havárie jsou dostupné na internetu. Jen pokud konverze takového číselníku do použitého IS je náročnější, je vhodné udělat při jeho instalaci jednou i zálohu.

♦

❑ Pád systému

Pád systému je neplánované ukončení práce systému. Důvodem mohou být neodchytnuté chyby uživatele, chyba SW na kterékoliv úrovni – operačního systému, SŘBD nebo aplikace, chyba HW, nehlídané vyčerpání kapacity paměti nebo disku apod. Dobrý návrhář očekává pády a plánuje jejich ošetření. Důležité je co nejvíce pádů rozpoznat a vydat o jejich příčinách zprávu do chybového protokolu i uživateli.

Nejjednodušší je případ, kdy jde jen o chybu, která nezpůsobila ztrátu dat a pak je možné systém spustit znovu. Často však nastává složitější situace, kdy jsou částečně porušena data u právě probíhajících nedokončených operací. Konečně může nastat nejhorší situace, kdy je zničena část nebo celá databáze.

Zkušenosti vývojářů říkají, že zpracování správných dat zabírá jen zlomek práce při tvorbě IS, většinu práce tvoří zpracování chyb. Ovšem současné SŘBD již výskyt chyb předvídají a velkou většinu ošetření chyb podporují. Přesto jim návrhář musí dodat potřebné informace. Tyto situace a jejich řešení budeme podrobně probírat v transakční analýze v následujících kapitolkách. Tam se také dozvíme, jak se pomocí zálohy databáze a dalšího souboru změn dá obnovit aktuální stav databáze i

v případě, že záloha není zcela „čerstvá“, že od poslední zálohy už proběhly v databázi některé další změny.

❑ Analýza archivace dat

Podobně jako u zálohování se provádí analýza archivování dat. Data v databázi se archivují podle různých pravidel, někdy celé tabulky, jindy jednotlivé entity - záznamy. Různé entity se archivují buď v pravidelných intervalech nebo při změně stavu na koncový stav = archivace.

Archivací obvykle rozumíme překopírování archivovaných dat do samostatné tabulky nazvané například Archiv_Xxx, kde Xxx je název původní tabulky. Někdy je vhodné zabezpečit přístup do archivovaných dat alespoň pro čtení. Pak se do IS přidá další subsystém nazvaný například Archiv a v něm budou funkce pro prohlížení nebo výpisy z archivovaných tabulek.

V těchto případech bychom neměli zapomenout na zálohování archivních tabulek.

U tabulek s malým počtem entit archivovaných je možné jen nastavit u entity stav = archivováno a ponechat ji v základní tabulce. Statistiky se pak provádějí pohodlněji z jediné tabulky, ne z původní a archivu současně. Ovšem pak ve všech minispecifikacích pracujících s takovou tabulkou je potřeba přidat podmínku, se kterými entitami se pracuje (viz též bod kontrola stavů).

Příklad:

IS Knihovna nebude archivovat celé tabulky, ale jen některé jejich entity. Z dynamické analýzy by mělo být známo (ze stavů jednotlivých entit), které entity se budou archivovat ze statistických důvodů.

Čtenáři, kteří si již nechodí vypůjčovat a vrátili průkaz čtenáře; u každého čtenáře by již měl být přidán atribut stav, *kde je například* stav = 0 ... *aktivní čtenář,* stav = 1 *ukončený čtenář. Není zřejmě nutné, aby byl každý odhlášený čtenář archivován ihned po odhlášení. Navíc takových čtenářů není mnoho. Bude tedy jistě stačit, když se archivace čtenářů provede jednou měsíčně.*

Realizované výpůjčky, tedy ty, kdy je kniha již vrácena, mohou být archivovány ihned po vrácení knihy. Opět ale není nutné je archivovat okamžitě, opět bude stačit archivace s pravidelnou periodou například týden, měsíc apod. – bude záležet na četnosti výpůjček, aby případně nezpomalovaly prohledávání tabulky výpůjček.

Totéž bude platit pro tabulku Rezervace.

U exemplářů se budou archivovat jen vyřazené nebo ztracené knihy, stačí s řídkou periodou, například 1 rok, opět podle četnosti vyřazování. U exempláře ale musí být opět atribut stav, označující například knihu v knihovně, vypůjčenou, zničenou, v opravě, ztracenou atd.

Některé tabulky není potřeba archivovat, stačí jejich zálohy, například Autor *nebo* Napsal.

♦

❑ Evidence chyb

Jistě známe jeden ze základních programátorských zákonů, že „v každém odladěném programu je chyba“. Zkušenost ukazuje, že zákon platí, i když se ta chyba odstraní.

Zvláště pro dobu po předání do provozu je velmi výhodné evidovat všechny chyby, které se při provozu objeví. Nemusí jít jen o chyby aplikačních programů, může jít i o chyby uživatelů, které systém umí rozpoznat a ošetřit.

Některé SŘBD mají zabudovány nástroje, pomocí nichž evidují chyby aplikací a řešitel je může pro ladění využít. Pokud to použitý SŘBD neumožňuje automaticky, je vhodné při detekci každé chyby zapisovat o ní podstatné údaje například do chybového databázového souboru. Zapisuje se datum a

čas, počítač, uživatel, chybová funkce i přesnější určení řádku kódu programu a typu chyby. Dobře spolupracující uživatele můžeme zaškolit i v tom, že při jakékoliv jejich připomínce nebo chybě sami „ručně“ zaznamenají do tohoto souboru svou poznámku. Zkušenosti ukazují, že i po krátké době uživatelé nejsou schopni přesně popsat situaci, při níž se objevily jejich problémy nebo chyba. Řešitel tak s obtížemi identifikuje problematickou funkci.

Je-li chybový protokol v databázové tabulce, může z ní správce systému snadno vybírat chyby jednotlivých funkcí, uživatelů, jejich četnosti atd. Tak lze mnohem přesněji identifikovat chyby aplikace a odstranit je nebo chyby jednotlivých uživatelů a poučit je.

❑ Evidence využívání funkcí IS

Užitečné je také evidovat spouštění jednotlivých elementárních funkcí. Podobně jako u chybového protokolu se při volání každé funkce uloží záznam o tomto volání do evidenční tabulky: opět datum a čas, počítač, uživatel, funkce. Uživatel o této evidenci nemusí případně ani vědět.

Pak je možné, aby správce systému analyzoval využívání funkcí a navrhoval změny do systému: velmi často využívané funkce nad velkými daty navrhl k další optimalizaci, na nevyužívané funkce upozornil uživatele pod.

❑ Role uživatelů a přístupová práva

Zatím jsme brali v úvahu stavy systému a entit jen z hlediska věcného – kdy která funkce je realizovatelná, případně pro které entity.

Jiné hledisko pro povolení spouštět jednotlivé funkce je rozlišení podle uživatelských rolí. Ne každý uživatel má právo provádět všechny funkce IS. Již při specifikaci zadání IS se provádí přidělení uživatelských funkcí jednotlivým rolím uživatelů. Toto rozdělení je třeba realizovat. Jde opět o řadu systémových funkcí (fungují skrytě, bez možnosti volby uživatelem) a tentokrát i o další data.

Potřebujeme evidovat jednotlivé uživatelské role a jejich práva, evidovat jednotlivé uživatele včetně loginů a hesel, jejich přiřazení k určitým uživatelským rolím. Zřejmě tak přidáme několik dalších systémových tabulek do databáze a k nim řadu dalších funkcí. Funkce musí zabezpečovat

- správci databáze manipulaci s rolemi a s uživateli – záznam nových, editaci, rušení;
- jednotlivým uživatelům během provozu IS variabilní nabídku funkcí podle jejich přístupových práv.

3.3. Transakce

❑ Porušení konzistence databáze

Již u datové analýzy jsme se zabývali konzistencí databáze. Jedno z velkých nebezpečí porušení konzistence databáze však hrozí i při provozu IS. Provoz databázového systému musí být bezpodmínečně zajištěn i proti možným chybám systému, technickým i programovým.

Zopakujme si, že **konzistence databáze je vzájemný soulad údajů v databázi.**

Konzistence databáze může být porušena vlivem chyb

- **při datové analýze** vlivem špatného návrhu struktury databáze: odtud může plynout redundance a z ní dále nekonzistence;

- **při funkční analýze** chybnými funkcemi nad databází: nedostatečnou kontrolou dat vkládaných uživateli nebo dokonce chybně implementované funkce pro manipulaci s daty;
- **během běhu aplikační úlohy vlivem systémové chyby HW nebo SW;** to se řeší v etapě návrhu implementace a nazývá se transakční analýzou;
- **během uložení dat na vnějším médiu;** to se opět řeší v návrhu implementace systémovými funkcemi pro zálohování databáze, případně evidováním změn databáze.

První dva případy jsme již probrali v kapitolách o analýze, pro další dva případy se musíme nejprve seznámit s některými novými pojmy.

❑ Porušení konzistence dat během běhu aplikační úlohy

Příčinou porušení konzistence dat při provozu mohou být poruchy při provozu počítače, diskové paměti, výpadky napětí a s tím spojená ztráta informací v operační paměti, chyby programového vybavení na úrovni operačního systému, SŘBD nebo aplikačních úloh.

Zvláště u velkých IS s velkým množstvím procesů není možné ponechat kontrolu nebo vyhledání takových chyb na „ručním“ přístupu uživatele. Databázový systém musí být schopen takovou chybu rozpoznat a opravit.

Uveďme si nejprve, ve kterých situacích může k nekonzistenci systému dojít. Datové soubory jsou uloženy na disku po blocích. Bloky (stránky) jsou jednotkou přenosu informace mezi vnější diskovou pamětí a operační pamětí počítače. Aplikační programy nad datovými soubory provádějí různé operace. Víme, že databázové operace jsou 4 typů: Insert, Update, Delete a Select. Tyto operace se však přeloží do jakýchsi elementárních operací čtení a zápisu dat. Insert znamená pouze zápis do databáze (write), Select pouze čtení (read), Update a Delete nejprve data přečtou, v paměti je změní a zapíší zpět do databáze (read + write).

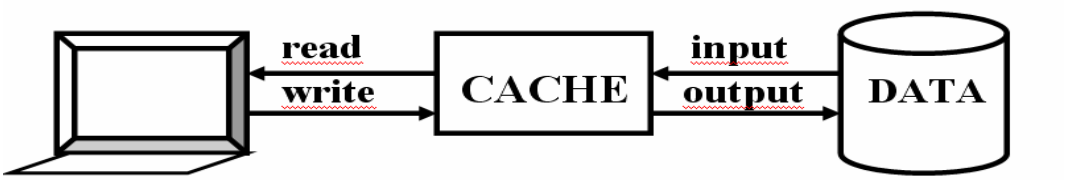
Přitom čtou data z disku po blocích do operační paměti, někdy je modifikované opět zapisují zpět na disk, opět po blocích. Tento proces používá dvě schematické operace:

- **read (X, PX)**, která načítá databázový záznam X do lokální proměnné PX; není-li blok, ve kterém je uložen záznam X na disku, právě umístěn v operační paměti, provede se operací **input(X)** přesun bloku z disku do vyrovná vrací paměti; záznam X se z vyrovnávací paměti přenese do PX.
- **write (X, PX)**, která hodnotu lokální proměnné PX zapíše do databázového záznamu X, který se nalézá ve vyrovnávací paměti; pokud blok, ve kterém je X umístěn, není v operační paměti, provede se nejprve operací input (X) jeho přesun z disku do vyrovnávací paměti; pak se provede přenos záznamu proměnné PX do X.

Žádná z obou operací nevyžaduje zápis na disk příkazem **output(X)**. Blok s daty je na disk zapsán tímto příkazem teprve v jedné z následujících situací:

- správce vyrovnávací paměti potřebuje v operační paměti místo pro jiný blok,
- program končí práci s datovým souborem příkazem close, pak správce vyrovnávací paměti zapíše všechny bloky tohoto souboru na disk,
- program dá příkaz k zápisu změn provedených ve vyrovnávací paměti na disk; pak jde o tzv. vynucený výstup, který provádí instrukce output (X).

Tato 3 operace tedy nemusí vždy bezprostředně následovat za operací write (X, PX), protože v bloku, kde se X nalézá, mohou být umístěny další položky, se kterými chce program pracovat. Odsud plyne, že když dojde k chybě mezi operací write a output, ztratí se nová informace uložená ve vyrovnávací paměti a dosud nezapsaná na disk.



vnitřní paměť vyrovnávací paměť disková paměť

Řešeni je v tom, že IS musí být schopen takovou chybu rozpoznat a vrátit stav databáze do konzistentního tvaru. Obvykle to znamená obnovit původní hodnoty dat, pro které ještě konzistence platila.

Ukážeme si tuto situaci na velmi jednoduchém příkladě. Obecně však databázové transakce mohou být složeny z mnohem většího počtu změn v databázových údajích. Transakce se vždy týkají jen změn v databázi.

Příklad:

Příklad nazývaný „algoritmus bankéře". V bance jsou evidována klientská konta v jednoduché tabulce Konto (klient, suma). Jednoduchou operací se převádí částka 100.- Kč z jednoho konta klienta A s hodnotou A = 500.- Kč na druhé konto klienta B s hodnotou B = 500.- Kč.

Podívejme se detailně na algoritmus této operace a průběžné hodnoty uložené jednak v databázi, jednak v paměti počítače.

program	vnitřní paměť	cache	databáze
			A=500, B=500
read(A,a)	a=500	a=500	
a:=a-100	a=400		
write(A,a)		a=400	
output(A			A=400, B=500
read(B,b)	b=500	b=500	
b:=b+100	b=600		
write(B,b)		b=600	
----------	---------- chyba systému ???	----------	----------
output(B)			A=400, B=600

Za porušení konzistence zde považujeme zřejmě stav, kdy součet A+B se změní. Dojde-li k poruše před první operací output, není ještě porušena konzistence, jen integrita (nesoulad se skutečností, částka již patří kontu B). To se dá vyřešit zopakováním celé transakce, až je chyba systému odstraněna. Dojde-li však k chybě mezi oběma operacemi output, je porušena konzistence i integrita a přitom není možné celou operaci spustit znovu, protože hodnota v A by se opět odečetla.

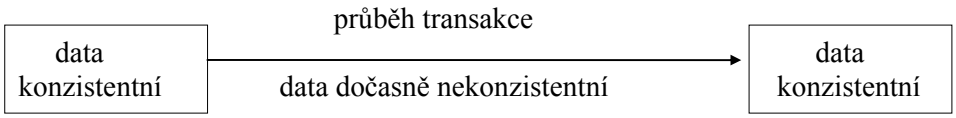
♦

Transakce

Pro řešení těchto situací se definuje pro nás nový pojem transakce.

Definice:

Transakcí nazýváme základní logickou jednotku zpracování. Aby byla zachována konzistence dat, musí transakce proběhnout buď celá, nebo je nutné obnovit původní stav a spustit transakci znovu. Říkáme, že transakce musí být atomická, tj dále logicky nedělitelná.

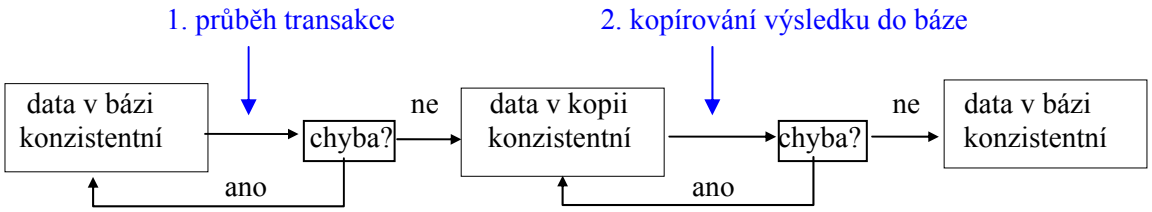


Uvedeme si, jakými metodami se tato vlastnost transakcí zajišťuje. Platí pro ně, že

- společnou vlastností všech metod je to, že **pracují s kopiemi dat** tak dlouho, dokud není jasné, že transakce proběhla bezchybně celá nebo že je nutné ji zopakovat.
- databázové transakce mohou být složeny z vysokého počtu operací.
- transakce se vždy týkají jen **změn v databázi**.

Metody pro zabezpečení průběhu transakcí

Metoda zpožděné aktualizace zapisuje data do diskového souboru teprve když je jisté, že transakce proběhla celá úspěšně. Výsledky transakce nezapisuje přímo do databázového souboru, ale do pomocného systémového souboru, kterému se říká **log**. Pokud dojde při transakci k chybě, může se celá provádět znovu, protože původní data nebyla změněna. Přitom se obsah pomocného souboru začne vytvářet znovu, původní je ignorován. Skončí-li transakce úspěšně, obsah souboru log se překopíruje do skutečného datového souboru. Pokud by došlo k chybě při kopírování, může se spustit znovu tolikrát, dokud neskončí tato druhá etapa úspěšně. Operace, které je možno spouštět opakovaně, aniž by došlo k porušení konzistence dat, nazýváme operacemi typu **redo**.



Původní transakce, která nebyla znovuspustitelná (nemůže se opakovat, aniž by došlo k porušení konzistence databáze), se rozdělila na dvě znovuspustitelné části. Ty se opakují, dokud neproběhnou obě až do konce.

Příklad:

Zvažme opět algoritmus bankéře, v němž je oddělena fáze realizace transakce od vlastního zápisu výsledku transakce do databáze:

fáze	program	vnitřní paměť	pomocný log-soubor	databáze
				A=500, B=500
1.	read(A,a)	a=500		
	a:=a-100	a=400		
	write(La,a)			
	output(La)		La=400	
	read(B,b)	b=500		
	b:=b+100	b=600		
	write(Lb,b)			
	----------	chyba systému, opakovat od 1.	----------	----------
	output(Lb)		La=400, Lb=600	
2.	read(La,a)	a=400		
	write(A,a)			
	output(A)			A=400
	read(Lb,b)	b=500		
	write(B,b)			
	----------	chyba systému, opakovat od 2.	----------	----------
	output(B)			A=400, B=600

♦

Metoda přímé aktualizace provádí zápis do výsledného datového souboru přímo, avšak pro případ neúspěšného ukončení transakce si ukládá výchozí hodnoty v pomocném souboru log. Skončí-li transakce úspěšně, obsah pomocného souboru log se ignoruje. Dojde-li v průběhu transakce k chybě, vrátí se zálohované hodnoty zpět do datového souboru. Operace, která obnoví původní hodnoty dat v souboru se nazývá operací typu **undo** a musí mít stejnou vlastnost jako operace redo, tj. že ji lze opakovat tak dlouho, dokud nebude provedena bez poruchy, aniž by to mělo vliv na její výsledek.

Příklad: *Máme opět stejný příklad s jinak řešeným ošetřením transakce:*

fáze	program	vnitřní paměť	pomocný log-soubor	databáze
				A=500, B=500
1.	read(A,a)	a=500		
	write(La,a)			
	output(La)		La=500	
	a:=a-100	a=400		
	write(A,a)			
	output(A)			A=400
	read(B,b)	b=500		
	write(Lb,b)			
	output(Lb)		La=500, Lb=500	
	b:=b+100	b=600		
	write(B,b)			
	----------	chyba, provést 2, pak opakovat od 1.	----------	----------
	output(B)			A=400, B=600
2.	read(La,a)	a=500		
	write(A,a)			
	output(A)			A=500,
	read(Lb,b)	a=500, b=500		
	read(Lb,b)			
	write(B,b)			
	----------	chyba, opakovat od 2.	----------	----------
	output(B)			A=500, B=500

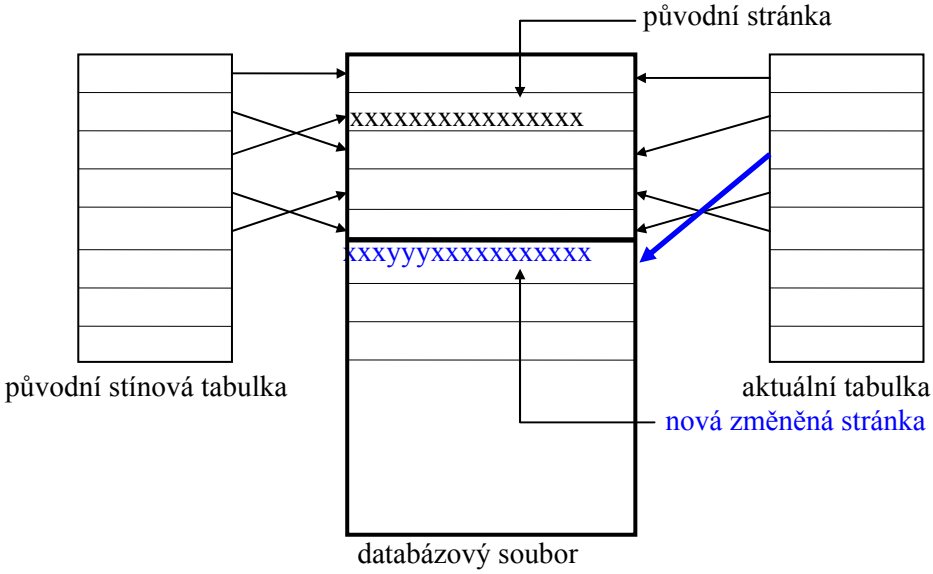
Obě uvedené metody používají systémový soubor log pro ukládání informací o průběhu transakcí. Aby se omezilo zdlouhavé prohledávání celého souboru log, ukládají se pro něj zadávají se tzv. kontrolní body (checkpoint), které způsobí pravidelné ukládání informací z paměti na disk a do souboru log. Pak při obnově hodnot databáze při vrácení transakce není nutné se vracet na začátek souboru log, ale jen k naposledy uloženému kontrolnímu bodu, který označuje "až sem to bylo dobře".

Existují i jiné metody, nepoužívající log-soubor. Jako příklad si uvedeme jednu takovou metodu. Ovšem tyto metody neuchovávají průběžné změny databáze a tak je není možné využít k obnově databáze (jak uvidíme v následujícím odstavci). Proto jejich použití není časté.

Metoda stínového stránkování používá k popisu uložení dat v databázi na disku dvě pomocné tabulky - tabulky stránek. Bloky v databázi na disku označuje jako stránky a v tabulkách stránek jsou postupně diskové adresy těchto bloků databáze.

Před zahájením transakce mají obě pomocné tabulky stejný obsah - aktuální adresy obsazených stránek. Stínová tabulka zůstává stejná. Pokud se během transakce mění obsah některé stránky, nechá se na disku původní stránka beze změny a stránka s novým obsahem se zapíše na prázdné místo na disku. V aktuální tabulce stránek se změní odkaz na novou stránku, ve stínové tabulce odkazů zůstává odkaz na starou stránku. Skončí-li transakce bezchybně, platí aktuální tabulka a místo se starým obsahem stránky na disku se uvolní pro další použití. Skončí-li transakce poruchou, platí stínová stránka, která obsahuje odkazy na stránky před zahájením transakce. Uvolní se naopak stránky se změněnými hodnotami, transakce se může zopakovat.

Problémem při tomto způsobu práce je evidence uvolněných stránek (garbage collection = sběr smetí) a uvolňování pro další použití. To zvyšuje složitost a režii celého systému.



Animace

Na CD-ROMu s tímto výukovým textem jsou animované příklady na metody zabezpečení transakcí.

- Transakce\T01 trans.exe
- Transakce\T02 trans chyba.exe
- Transakce\T03 zpozdena aktual.exe
- Transakce\T04 zpozdena chyba 1 faze.exe
- Transakce\T05 zpozdena chyba 2 faze.exe
- Transakce\T06 zpozdena chyba 1 2 faze.exe
- Transakce\T07 prima aktual.exe
- Transakce\T08 prima chyba.exe
- Transakce\T09 prima chyba 1 2 faze.exe

Úvod do transakční analýzy

Pokud používáme SŘBD, který podporuje transakce (není to každý), pak je zapotřebí doplnit do minispecifikací označení transakcí – jejich začátek a konec.

U výše použitých příkladů jsme za transakci považovali skupinu příkazů, manipulujících (měnících) s obsahem databáze. Ovšem SŘBD nemůže rozeznat, která skupina příkazů tvoří jednu atomickou transakci a které příkazy již patří k jiné transakci. Proto je nutné transakce v programu označit.

Většina SŘBD s podporou transakcí má příkazy (patří i do jazyka SQL):

begin transaction ... označení začátku transakce
end transaction ... označení konce transakce
commit ... dokončení transakce se zápisem do databáze
rollback ... zrušení transakce s vrácením počátečních hodnot

Tyto příkazy se doplní do algoritmu elementární funkce. Kam přesně, to probereme na konci této kapitoly 3 v odstavci o transakční analýze. Zde si uvedeme jen jednoduchou situaci, kdy je transakce zřejmá. Nazveme „tělem transakce“ příslušnou posloupnost příkazů. Pak obvykle doplnění o uvedené 4 příkazy má tvar:

begin transaction
příkaz 1 } tělo transakce
...
if ERROR then **rollback**
else **commit**
end transaction

Pokud tuto konstrukci nepoužijeme, SŘBD s podporou transakcí považuje za transakci každou databázovou operaci samostatně – tedy INSERT, UPDATE, DELETE. Zabezpečí tedy, aby každá samostatně proběhla bezchybně. Ale už z příkladu algoritmu bankéře víme, že to k zajištění konzistence databáze nestačí. Tam musí proběhnout všechny databázové operace správně, přerušení nebo chyba mezi nimi konzistenci nezaručí.

Příklad:

Algoritmus bankéře jako transakce:

```
begin transaction
    read(A,a);
    a:=a-100;
    write(A,a);
    read(B,b);          } tělo transakce
    b:=b+100;
    write(B,b);
    if ERROR then rollback
             else commit;
end transaction;
```

♦

Některým SŘBD stačí označit jen begin a end bez commit nebo rollback, jiné místo začátku a konce používají příkaz **begin work** a **end work**.

V transakční analýze budeme do algoritmu minispecifikace také jen označovat začátek a konec transakcí, bude věcí programátora, aby pak použil správnou syntaxi

3.4. Porušení konzistence dat ve vnější paměti

❑ Zálohování databáze

Dosud popisované metody chrání data před ztrátou informace v paměti počítače, v průběhu modifikací dat v aplikační úloze. Ke ztrátě informace však může dojít také na disku. Ochrana dat na vnějších médiích před ztrátou by měla být součástí každého informačního systému.

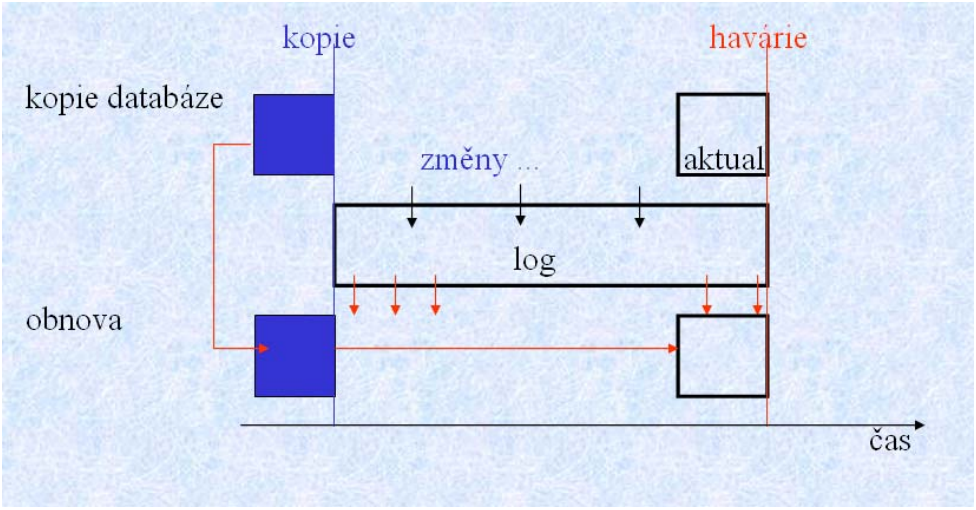
Základní metodou ochrany diskových dat je důsledné a pravidelné **zálohování databáze**. Základní kontrola správnosti kopií se provádí na úrovni operačního systému.

Během ukládání zálohovaných dat nesmí být obvykle prováděny žádné transakce a uloženy musí být také všechny informace, z nichž by bylo možno rekonstruovat databázi v případě poruchy. K tomu opět slouží **změnové soubory log**, je to jejich druhé významné využití.

Analýzu zálohování jsme probrali v návrhu implementace. Pro každou databázovou tabulku se analyzuje, jak často se mění a tedy jak často je nutné ji zálohovat. Podle toho se navrhnou další funkce, které budou zálohování v předepsaných termínech realizovat.

❑ Obnova databáze

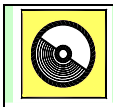
Pokud dojde z jakéhokoliv důvodu ke zničení části nebo celé databáze, je důležité mít jednak kopii databáze, jednak soubor log, do kterého se zaznamenávají průběžné změny hodnot od poslední kopie. Zničená data v databázi pak zrekonstruuje vhodný program obnovením dat ze staré kopie a automatickým provedením všech modifikací ze souboru log. Tak získáme data aktuální v okamžiku jejich zničení.



Samozřejmě je důležité uchovávat jak kopii databáze, tak log-soubory na jiném médiu, než zdrojovou databázi. Jinak hrozí při zničení média, že bude zničena jak databáze, tak její kopie nebo změnový soubor.

Větší SŘBD obsahují prostředky pro průběžný záznam modifikací databáze i pro obnovu databáze z její starší kopie. Do informačních systémů se pomocí nich zabudují moduly pro zabezpečení ochrany dat. SŘBD mívají jak programy pro obnovu celé databáze, tak programy pro obnovu jen jednotlivých tabulek.

Pokud je provedena analýza zálohování a každá tabulka je zálohována v jiných intervalech, je vhodné i obnovovací funkci upravit tak, aby, aby se za zdrojovou kopii braly poslední kopie každé tabulky.



Animace

Na CD-ROMu je animovaný příklad na obnovu databáze z log souboru.

- Transakce\T10 obnova databaze.exe

Ochrana databáze bez podpory transakcí

U malých SŘBD, které takové nástroje nemají (nepodporují transakce, nemají prostředky pro obnovu aktuálního stavu báze pomocí log-souboru ap.), je nutné konstruovat jednotlivé elementární funkce informačního systému jako znovuspustitelné. Pro tyto účely existují vhodné techniky programování, jako.

- psát dávkové elementární funkce jako znovuspustitelné;
- při modifikaci dat uvnitř tabulky provádět nevratné změny na kopii souboru (dávkové změny v datech, třídění) a až po bezchybném výsledku zrušit starý soubor;
- připravovat předem bezchybné dávky vstupních nebo editovacích dat, zkontrolovat jejich bezchybnost testovacím programem a teprve potom provádět ukládání nebo modifikace dat v databázi;
- provádět častější zápis na disk, třeba vynucenými příkazy.

Je zřejmé, že všechny tyto postupy zvětšují paměťové i časové nároky databázového systému, ovšem za konzistenci databáze to nebývá cena vysoká.

3.5. Paralelní procesy nad databází

❑ Víceuživatelský přístup k databázi

Všechny naše dosavadní úvahy o zachování konzistence a integrity databáze se týkaly (aniž jsme to zdůrazňovali) aplikací jednouživatelských. Databázový systém byl provozován vždy jedním uživatelem, databázové operace se prováděly sériově – postupně jedna za druhou. Databázové soubory byly k dispozici v dané chvíli pouze jednomu uživateli.

U převážné většiny informačních systémů však je nutné, aby databáze byla současně přístupná více uživatelům a aby s ní pracovalo současně (paralelně) více aplikací. Typickými příklady jsou velké systémy pro rezervaci místenek, jízdenek či letenek. Tato potřeba však vyvstává např. i v malých firmách, kde se na několika počítačích provozuje účetnictví, skladové hospodářství, osobní evidence, mzdy ap. a jednotlivé aplikace užívají společnou databázi. Obdobně to platí u databází školních, nemocničních, knihovních a mnohých dalších.

V těchto případech nastává nový typ problému: jak zajistit, aby při paralelním zpracování dat v databázi nedocházelo vlivem špatné koordinace zpracování k chybám, k porušení konzistence a integrity.

Pokud programy data jen čtou, je vhodné zajistit co největší míru paralelismu. Hodnoty dat se nemění a nemůže tedy vzniknout nekonzistence. U programů, které databázi modifikují (vkládají, ruší nebo aktualizují data) je však nutné zajistit, aby **v každém okamžiku měl k datům přístup jen jediný program.**

Problémy s řízením paralelních procesů vznikají u aplikací s databází provozovaných na jediném počítači pomocí terminálové sítě, databází provozovaných prostřednictvím lokální počítačové sítě a v největší míře u databází distribuovaných. Jejich řešením se zabývají odborníci – informatici. Zde si je jen stručně popíšeme a naznačíme řešení.

❑ Požadavek sériovosti transakcí

Obdobně jako při zajišťování konzistence dat vlivem chyby systému při jednouživatelském provozu databáze také zde je základní jednotkou zpracování transakce. Každý uživatel pracuje nezávisle na ostatních a spouští své úlohy – transakce v jakémsi náhodném pořadí. Každá transakce sama o sobě je zabezpečena, aby neporušila konzistenci. Pokud by tedy SŘBD zabezpečoval, že každá transakce proběhne celá bez přerušení jinou transakcí, nenastaly by problémy.

Ovšem toto tzv. **sériové** zpracování transakcí by bylo velmi pomalé. Transakce pracují s databází, tedy s pomalými přenosy dat mezi databází a pamětí. V době přenosů dat v jedné transakci je procesor počítače nevyužitý a mohl by zpracovávat části jiné transakce. V tom případě řekneme, že se transakce provádějí **paralelně** nebo souběžně. Budou-li různé transakce pracovat s různými daty, opět nenastane problém. Pokud však různí uživatelé pracují se stejnými daty, mohou nastat problémy nového typu.

Příklad:

Opět použijeme příklad převodu mezi bankovními konty. Mějme dvě transakce T0 a T1, příslušející dvěma paralelně běžícím programům. Počáteční stav kont je A = 1000, B = 2000. Jedna transakce převádí mezi konty 50 Kč, druhá 10% aktuální částky konta.

Transakce **T0:**

read(A)
A:=A-50
write(A)
read(B)
B:=B+50
write(B)

T1:

read(A)
pom:=A*0.1
A:=A-pom
write(A)
read(B)
B:=B+pom
write(B)

Provádíme-li transakce postupně, můžeme je provést v pořadí T0, T1 nebo T1, T0.

V prvém případě je výsledek A = 855, B = 2145.

Ve druhém případě je výsledek A = 850, B = 2150.

Protože pro obě transakce je podmínkou konzistence konstantní velikost součtu A+B (převáděním se celková částka nemůže změnit), při obou výsledcích zůstává konzistence zachována. Je to zřejmě tím, že každá transakce proběhne celá bez přerušení.

♦

Obecně pro n současně běžících transakcí existuje n! možností sériového pořadí. Sériové zpracování transakcí tedy může vést k různým výsledkům, zůstává však vždy zachována konzistence databáze.

Připusťme nyní libovolné **paralelní zpracování** transakcí, tj. takové, že se příkazy obou transakcí (obou souběžně běžících programů) mohou různě střídat. Říkáme, že transakce jsou prováděny podle určitého **schématu**. Takových schémat paralelního zpracování je zřejmě mnoho podle toho, jak se mezi sebou střídají jednotlivé příkazy obou transakcí. Ukážeme si, že některá z nich vedou k porušení konzistence, některá ne.

Příklad:

Schéma 1, při kterém dochází k porušení konzistence dat. Pro jednodušší zápis transakce již nevypisujeme zvlášť operace write *a* output*, ale samozřejmě předpokládáme zabezpečení výsledku každé transakce pomocí některé z dříve popsaných metod.*

proces T0	**proces T1**	**paměť T0**	**paměť T1**	**databáze**
				A=1000,B=2000
read(A) A:=A-50		A=1000 A=950		
	read(A) pom:=A*0.1 A:=A-pom write(A) read(B)		A=1000 pom=100 A=900 B=2000	A=900 ↓ B=2000
write(A) read(B) B:=B+50 write(B)		B=2000 B=2050		A=950 B=2050 ↓
	B:=B+pom write(B)		B=2100	B=2100

Výsledné hodnoty v databázi A=950, B=2100 jsou zřejmě nekonzistentní, došlo totiž vícekrát k vzájemnému přepsání vypočtených hodnot oběma transakcemi.

♦

Příklad:

Schéma 2, při kterém nedochází k porušení konzistence dat

proces T0	proces T1	paměť T0	paměť T1	databáze
read(A) A:=A-50 write(A)		A=1000 A=950		A=1000,B=2000 A=950
	read(A) pom:=A*0.1 A:=A-pom write(A)		A=950 pom=95 A=855	 **A=855**
read(B) B:=B+50 write(B)		B=2000 B=2050	B=2000	B=2000 B=2050
	read(B) B:=B+pom write(B)		B=2050 B=2145	 **B=2145**

Výsledné hodnoty v databázi A=855, B=2145 jsou konzistentní.

♦

U víceuživatelského provozu databáze požadujeme splnění nového požadavku, zabezpečujícího konzistenci databáze, nazývaného sériovostí transakcí.

Definice:

Sériovost transakcí je požadavek, aby výsledek po paralelním provedení řady transakcí byl stejný, jako když by byly provedeny celé transakce postupně za sebou v libovolném pořadí.

Úkolem je zřejmě najít schémata, která splňují požadavek sériovosti.

❑ Sériovost testovaná precedenčním grafem

Je možno vysledovat, že důležité jsou operace read () a write () a jejich pořadí, ostatní operace nemají vliv na konzistenci výsledku paralelního zpracování.

Otázkou tedy je, jak se pozná, kdy paralelní schéma splňuje požadavek sériovosti a kdy ne.

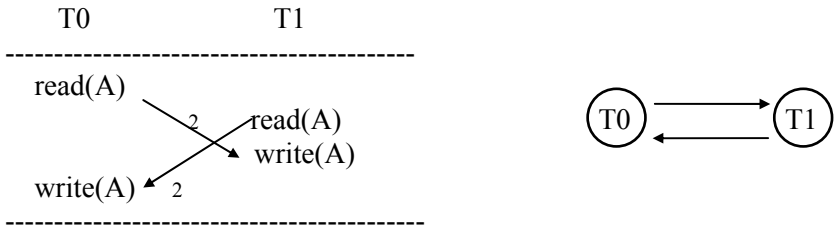
Pro systém, kde každá transakce nejprve přečte objekt operací read() a teprve potom do něj zapíše operací write(), je možno sestrojit tzv. **precendenční graf**. Je to orientovaný graf, jehož uzly jsou transakce a jehož hrany jsou orientovány **Ti → Tj**, jestliže

(1) Ti provede write(A) dříve, než Tj provede read(A)
(2) Ti provede read(A) dříve, než Tj provede write(A).

Příklad:

U předcházejících schémat 1 a 2 dostaneme pro objekt A grafy

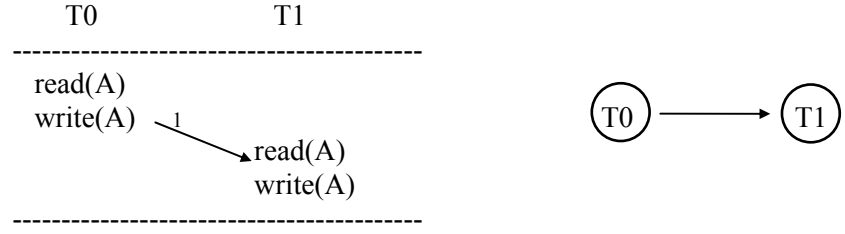
Schéma 1:



Platí, že

T0 provede read (A) dříve než T1 provede write(A), dle (2) platí T0→ T1
T1 provede read (A) dříve než T0 provede write(A), dle (2) platí T1→ T0

Schéma 2:



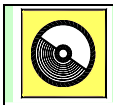
Platí, že

T0 provede WRITE dříve než T1 provede READ, dle (1) platí T0 → T1
T1 neprovede dříve než T0 nic

♦

Jestliže získaný orientovaný graf obsahuje cyklus, pak testované schéma paralelního zpracování transakcí nesplňuje požadavek sériovosti.

Pro systém, kde transakce zapisuje pomocí write(A), aniž by předtím četla operací read(A) lze ukázat, že neexistuje žádný efektivní algoritmus, který by rozhodl, zda dané schéma paralelního zpracování transakcí splňuje požadavek sériovosti. Proto tato metoda by byla použitelná jen pro transakce obsahující pouze operace UPDATE a DELETE (ty nejprve přečtou záznam operací read() a potom jej zapíší operací write()), nikoliv pro transakce s operací INSERT (zde jde jen o zápis operací read()).



Animace

Na CD-ROMu je animovaný příklad na použití precedenčního grafu:.

- Seriovost\S08 seriovost preceden graf T0 T1.exe
- Seriovost\S09 neseriovost preceden graf T0 T1.exe
- Seriovost\S10 seriovost preceden graf T0-T2.exe
- Seriovost\S11 seriovost preceden graf T0-T3.exe

❑ Zamykání

Jedním ze způsobů, jak zajistit požadavek sériovosti, je zpřístupnit data vždy jen jediné transakci. Když jedna transakce získá k údaji výlučný (exklusivní) přístup, pak tento údaj nemůže modifikovat jiná transakce dříve, než první transakce skončí a uvolní přístup k údaji - a to i v případě, že byla při paralelním zpracování několikrát přerušena. Říkáme, že údaje jsou **zamčeny**.

Jediný klíč ke každému zámku při modifikaci přiděluje SŘBD těm transakcím, které o něj požádají.

Zámky údajů v databázi můžeme rozdělit podle několika hledisek: **co** se zamyká, **jak** se zamyká a **kdo** zamyká.

Rozlišujeme zámky dvou základních druhů (JAK se zamyká)

1. Zámky pro sdílený přístup (shared) umožňují údaje jen číst více transakcím současně, ne však do nich zapisovat.
2. Zámky výlučné (exclusive) umožní čtení i zápis vždy pouze jediné transakci.

Existuje několik úrovní zamykání údajů (CO se zamyká)

1. Na úrovni operačního systému definujeme soubor jako typ read-only a tak zakážeme zápis a modifikaci všem. V IS se tento postup nevyskytuje často, ale je možný například pro některé neměnitelné číselníky.
2. Zamknout je možno celou **databázi** pro jedinou transakci a tak ji znepřístupnit ostatním transakcím. Půjde zřejmě o výjimečné situace, například při zálohování databáze a v podobných situacích, kdy je nutné zabezpečit na jistou dobu stabilní obsah databáze.
3. Na úrovni SŘBD v aplikačním programu zamkneme **tabulku** pro buď výlučně (exclusive) a tak k ní zamezíme přístup všem ostatním procesům, dokud náš program neskončí a neuvolní ji. To je vhodné jen v nutných případech. Pro čtení zamykáme tabulku sdíleně a tak ostatní transakce z ní mohou také číst, ale ne zapisovat.
4. V aplikačním programu stačí často zamknout jen jeden nebo několik **záznamů**, ne celý soubor, aby tak byly ostatní záznamy přístupné ostatním uživatelům. Tento postup je nejvhodnější, pokud situace nevyžaduje zámek větší části databáze.
5. Některé SŘBD umožňují zamykat dokonce jen jednotlivé **atributy**. Ovšem evidence toho, co je pro kterou transakci zamčeno, je pak nutná pro každou hodnotu atributu a to znamená velkou režii navíc.

Pokud má jedna transakce údaj (soubor, záznam) uzamčený a další transakce jej chce uzamknout také, může dojít ke kolizi. Proto v SŘBD existují funkce testující, zda je údaj volný. Pokud není, je nutno situaci programově řešit (počkat na uvolnění, zrušit transakci ap.).

Ovšem s používáním zámků mohou nastat nové problémy. Ukážeme si je na příkladech. Zaveďme si následující označení pro příkazy uzamčení a odemknutí:

LS(A) ... zamkni objekt (tabulku, zíznam) A pro sdílený přístup (Lock Shared)
LX(A) ... zamkni objekt A pro výlučný přístup (Lock eXclusive)
UN(A) ... uvolni objekt A (UNlock)

Žádosti LS(A) lze zřejmě vyhovět vždy, není-li na A zámek typu LX(A). Žádosti LX(A) lze vyhovět pouze tehdy, je-li položka A ve stavu po provedení UN(A).

Realizace zámků (KDO zamyká)

Pro zamykání buď v jazyce SŘBD existují speciální příkazy, které používá aplikační programátor v IS (zde budeme pro ně používat zavedené názvy LS, LX a UN), nebo zamykání provádí SŘBD automaticky současně s realizací některého databázového příkazu.

Použití zámků však není jednoduché, nesprávné použití může vést k nesprávným výsledkům, jak ukáží následující příklady.

Příklad:

Mějme dvě transakce T1 a T2 s počátečními hodnotami A = 100, B = 200.

T1: LX(B)	**T2:** LS(A)
read(B)	read(A)
B:=B-50	UN(A)
write(B)	LS(B)
UN(B)	read(B)
LX(A)	UN(B)
read(A)	display(A+B)
A:=A+50	
write(A)	
UN(A)	

Sériová provedení transakcí T1, T2 i T2, T1 dají jako výsledek příkazu display(A+B) hodnotu 300. Při následujícím paralelním schématu 3 je však výsledek jen 250.

***Schéma 3**, u kterého není dodržen požadavek sériovosti.V databázi tentokrát jsou vyznačeny tučně hodnoty nezamčené, které jsou přístupné kterékoliv transakci, netučně hodnoty sice v databázi uložené, ale uzamčené jednou transakcí a nepřístupné jiným transakcím.*

proces T1	proces T2	paměť T1	paměť T2	databáze
				A=100,B=200
LX(B)				
read(B)		B=200		
B:=B-50		B=150		
write(B)				B=150
UN(B)				**B=150**
	LS(A)			
	read(A)		A=100	
	UN(A)			
	LS(B)			
	read(B)		B=150	
	UN(B)			
	display(A+B)		**A+B=250**	
LX(A)				
read(A)		A=100		
A:=A+50		A=150		
write(A)				A=150
UN(A)				**A=150**

Důvodem nesprávného výsledku při zobrazení výsledku transakce T2 je to, že transakce T1 uvolnila položku B příliš brzy.

♦

❑ Protokol o zámcích

S používáním zámků tedy nastávají nové problémy. Pokud nejsou zámky používány správně, obvykle když transakce odemyká své záznamy příliš brzy, kdy ještě nejsou všechny její záznamy modifikovány, konzistence databáze pořád může být porušena. Proto existují pravidla nazvaná **protokoly o zámcích**, které definují, kdy se mají části databáze zamykat a kdy odemykat. Při jejich dodržování je konzistence zaručena i při paralelním zpracování transakcí.

K řešení požadavku sériovosti se tedy používají protokoly o zámcích. Takových protokolů je řada. Uvedeme se základní variantu takové metody, nazvanou **metoda dvoufázového zamykání**.

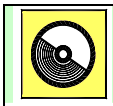
Dvoufázové zamykání spočívá v tom, že transakci rozdělíme na 2 fáze; v první fázi zámky jen zamykáme a neuvolňujeme, ve druhé fázi naopak jen uvolňujeme a nezamykáme. První příkaz

odemknutí zámku tak může být až po posledním příkazu uzamčení. Často se tak odemknou všechny objekty až před ukončením transakce. Kam umístit jednotlivé příkazy uzamčení a odemknutí tabulek nebo záznamů provádí **transakční analýza**. K té se dostaneme v následujícím odstavci.

Příklad:

U předcházejícího příkladu s transakcemi T1 a T2 se schématem 3 transakce T1 nemá dodržený protokol o zámcích. Položka B je uvolněna dříve, než je uzamčena další položka téže transakce A (oba příkazy jsou označeny modře). To je důvodem k nekonzistentnosti výsledku souběžné transakce T2.

♦



Animace

Na CD-ROMu jsou animované příklady na sériové zpracování, použití zámků, 2-fázový protokol:

- Seriovost\S01 seriove T0 T1.exe
- Seriovost\S02 seriove T1 T0.exe
- Seriovost\S03 paralelni chybne schema.exe
- Seriovost\S04 paralelni spravne schema.exe
- Seriovost\S05 paralelni zamykani spravne.exe
- Seriovost\S06 paralel zamykani ne 2-faze.exe
- Seriovost\S07 paralel zamykani 2-faze.exe

❑ Uváznutí

Vhodným zamykáním, které vyhovuje protokolu o zámcích, je tedy zajištěn požadavek sériovosti = je zajištěna konzistence databáze. Může však nastat nový problém, pokud transakce pracují se stejnými záznamy.

Jednoduché řešení, jak zajistit dvoufázové zamykání, je uvolnit položky až po ukončení celé transakce. Následující příklad ukáže, k jakým novým problémům by to mohlo vést.

Příklad:

Mějme upravené transakce T3 a T4 se změnou zařazení příkazů UN().

T3: LX(B)	**T4:** LX(A)
read(B)	read(A)
B:=B-50	LS(B)
write(B)	read(B)
LX(A)	display(A+B)
read(A)	UN(A)
A:=A+50	UN(B)
write(A)	
UN(B)	
UN(A)	

***Schéma 4** paralelního zpracování, které dodržuje protokol, ale uvolňuje položky pozdě.*

proces T3	proces T4
LX(B) read(B) B:=B-50 write(B)	
	LX(A) read(A) **LX(B)**
LX(A) ...	

... T4 čeká na uvolnění položky B, přepne se na T3
... T3 marně čeká na uvolnění položky A, nelze přepnout na T4, marně čeká i T4

Obě transakce T3 a T4 tento protokol dodržen mají.

Jedna transakce zamkne záznam B, pak druhá transakce zamkne záznam A. Dále prvá transakce potřebuje zamknout záznam A, ale musí počkat, až bude odemčen. Na to druhá transakce potřebuje zamknout záznam B, ale musí počkat, až bude odemčen.

*Tak **čekají obě transakce navzájem,** u T3, T4 došlo k uváznutí.*

♦

Definice:

Situaci, kdy transakce čekají navzájem na odemknutí objektu databáze, kdy nelze žádný požadavek uspokojit a celý proces uvázne v mrtvém bodě, nazýváme **uváznutím (deadlock)**.

Problém tedy je v tom, že pokud používáme zámků málo, hrozí nekonzistence, používáme-li zámků mnoho, hrozí uváznutí.

Máme nyní nový problém: řešení uváznutí v mrtvém bodě.

Problém uváznutí se řeší pomocí dvou typů metod

- pomocí **prevence uváznutí**, kdy operace zamykání a uvolňování řídí v transakcích speciální modul - plánovač tak, aby k uváznutí nedošlo; pokud by hrozilo uváznutí, plánovač to předem rozpozná, příslušnou transakci zruší a spustí ji celou znovu; tím se pořadí zámků změní a k uváznutí znovu nemusí dojít;
- SŘBD připustí uváznutí, ale umí jej detekovat – **rozpoznat a vyřešit**; řeší jej opět zrušením některé z uváznutých transakcí; tím ostatní transakce mohou pokračovat; zrušená transakce je opět spuštěna znovu.

Všimněme si, že se u obou typů metod používá některé z metod pro znovuspuštění transakcí, aniž došlo k chybě HW nebo SW, jak jsme si popisovali výše. Pomocí log souborů se zde řeší kolize víceuživatelského přístupu.

❑ Prevence uváznutí

Pro prevenci uváznutí existuje více technik.

1. Nejjednodušší metodou prevence uváznutí je uzamčení všech položek, které transakce používá, hned na začátku transakce, ještě před databázovými operacemi, a jejich uvolnění až na konci transakce. Tak se transakce nezahájí dříve, dokud nemá k dispozici všechny potřebné údaje a nemůže dojít k uváznutí uprostřed transakce. Tato metoda však má dvě velké nevýhody:

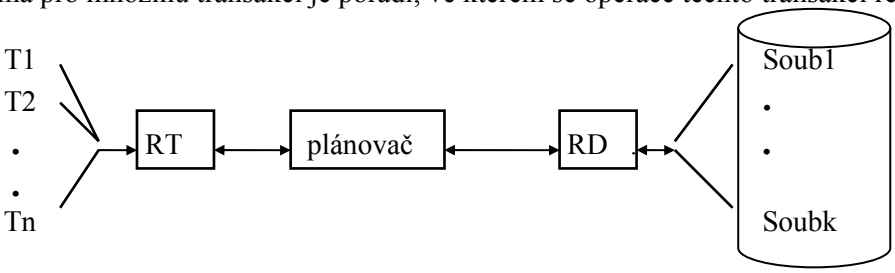
- využití přístupu k položkám je nízké, protože jsou dlouhou dobu zbytečně zamčené,
- transakce musí čekat až budou volné současně všechny údaje, které chce na začátku zamknout, a to může trvat velmi dlouho,
- z obou důvodů je tak průchodnost transakcí velmi nízká, odpovídá téměř sériovému zpracování transakcí a to jsme již výše zamítli pro nízké využití procesoru.

2. Jiná metoda prevence uváznutí využívá faktu, že k uváznutí nedojde, jestliže transakce zamykají objekty v pořadí respektujícím nějaké lineární uspořádání, definované nad těmito objekty (např. abecední ap.). Z hlediska uživatelského však takový požadavek je příliš omezující.

3. **Plánovače**

Některé systémy řeší problém uváznutí synchronizací paralelních transakcí pomocí speciálních modulů zabudovaných v SŘBD a nazvaných **plánovače**. Jsou to tyto programové moduly:

- Modul řízení transakcí (RT); je to fronta, do které transakce zapisují své žádosti o vykonání operací read(X) a write(X). Každá transakce je doplněna příkazy BEGIN TRANSACTION a END TRANSACTION, označujících jejich začátek a konec v rámci kódu programu (viz transakční analýza níže).
- Modul řízení dat (RD) realizuje čtení a zápis objektů dle požadavků plánovače a dává plánovači zprávu o výsledku a ukončení.
- Plánovač zabezpečuje synchronizaci požadavků z fronty dle realizované strategie a řadí požadavky do schémat.
- Schéma pro množinu transakcí je pořadí, ve kterém se operace těchto transakcí realizují.



Víme, že nejjednodušší schéma je sériové (vždy proběhne celá transakce, pak další), ovšem je málo průchodné. Cílem celé strategie je větší průchodnost systému.

Plánovač při dvoufázovém zamykání vykonává tyto operace

- řídí zamykání objektů;
- operace čtení a modifikace objektů povoluje jen těm transakcím, které mají příslušné objekty zamknuté;
- sleduje, jestli transakce dodržují některý protokol o zámcích; pokud zjistí jeho porušení, transakci zruší;
- předcházejí uváznutí nebo ho detekují a řeší zrušením transakce.

❑ Plánovač pomocí časových razítek

Jiný způsob řízení paralelních transakcí je pomocí časových razítek (ČR). Časové razítko je číslo přidělené transakci nebo objektu (tabulce, záznamu) databáze.

Čísla přidělovaná transakcím tvoří rostoucí posloupnost (například jsou funkcí času). Jsou jednoznačná pro všechny transakce a jsou stejná pro všechny operace read() a write() jedné transakce. Čísla používá plánovač pro řízení konfliktních operací read(A) a write(A). Konfliktními operacemi

rozumíme dvě operace týkající se téhož objektu databáze a alespoň jedna z nich je WRITE. Všechny páry konfliktních operací se provádějí v pořadí jejich ČR, pak vytvářejí sériová schémata.

Princip základního plánovače založeného na ČR:

Tento typ plánovače eviduje pro každý objekt A databáze dvě čísla

- největší ČR, které měla operace read(A), již provedená nad objektem A, označíme jej R/ČR(A)
- největší ČR, které měla operace write(A) provedená nad A, označíme jej W/ČR(A).

Algoritmus plánovače s časovým razítkem:

Když plánovač obdrží požadavek s nějakým časovým razítkem ČR na čtení hodnoty objektu A, provede příkaz:

je-li ČR < W/ČR(A)
pak odmítne požadavek a zruší transakci, která požadavek zaslala,
jinak vyhoví požadavku a aktualizuje hodnotu
R/ČR(A) = max(ČR, R/ČR(A))

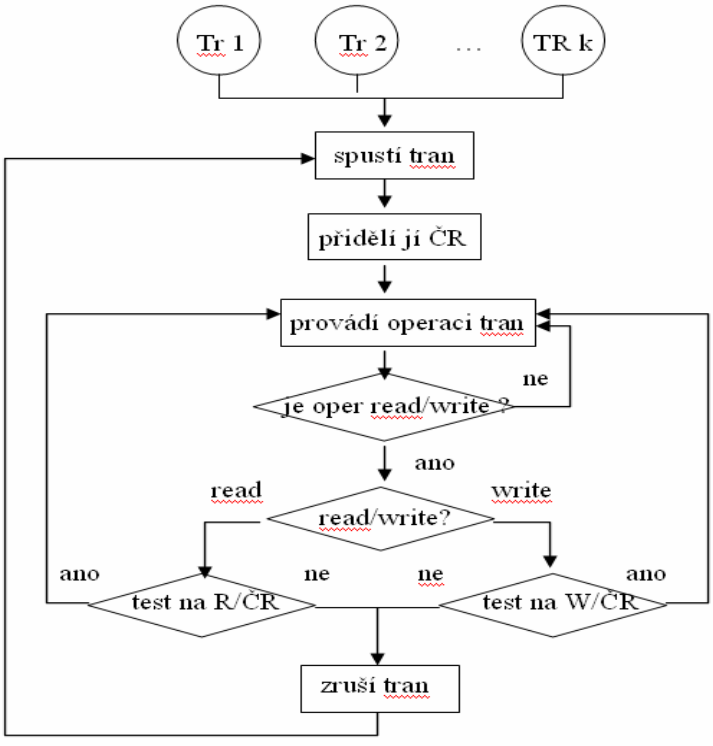
Když plánovač obdrží požadavek s nějakým časovým razítkem ČR na zápis hodnoty objektu A, provede příkaz:

je-li ČR < W/ČR(A) or ČR < R/ČR(A)
pak odmítne požadavek a zruší transakci, která požadavek zaslala,
jinak vyhoví požadavku a aktualizuje hodnotu
W/ČR(A) = ČR

Zrušené transakce se znovu spustí s novou (vyšší) hodnotou ČR.

Tento základní plánovač může způsobovat časté rušení transakcí. Existují jeho modifikace nebo jiné strategie plánovačů, které snižují počet zrušení transakcí.

Na následujícím vývojovém diagramu je postup, jak řeší SŘBD transakce pomocí plánovače při běhu aplikace.



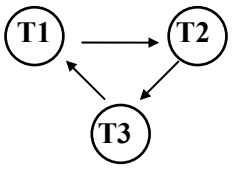
❑ Řešení problému uváznutí

Jestliže SŘBD nepoužívá prevenci uváznutí, musí mít prostředky pro **detekci (rozpoznání) uváznutí** a obnovu činnosti umrtvených transakcí.

Detekce se provádí obvykle použitím grafu vztahů "kdo na koho čeká". Je to graf, jehož uzly jsou transakce a orientované hrany představují uvedenou závislost. Záznamem a analýzou grafu čekání se rozpoznává uváznutí. Je-li v grafu cyklus, systém uvázl v mrtvém bodě.

Příklad:

Běží 3 transakce T1, T2, T3. V jisté situaci se vytvořil graf tvaru



T1 čeká na T2
T2 čeká na T3
T3 čeká na T1

Protože je v grafu cyklus, je rozpoznáno uváznutí.

♦

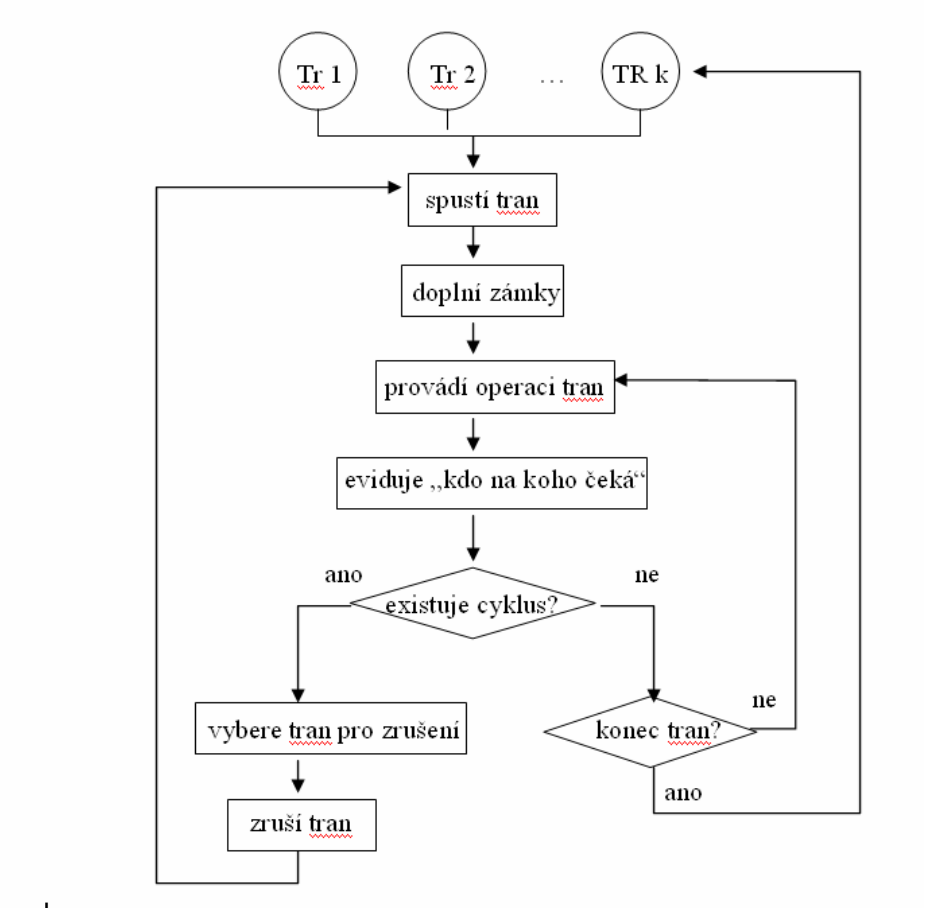
Jestliže taková situace nastane, systém musí jednu nebo více transakcí vrátit zpět (pomocí souboru log), čímž se zablokovaný přístup k datům (pro tuto transakci) odblokuje a umožní provést ostatní transakce. Připomíná to situaci, kdy se dva automobily potkají na úzké cestě a jeden musí vycouvat.

Obnovení činnosti se provádí pomocí souboru log, popsaného v předchozí kapitole. V případě potřeby je možno kteroukoliv transakci vrátit. Jde jen o to, kdy a které transakce se mají provést znovu. Systém vybírá takové transakce, aby s celým postupem byly spojeny co nejmenší náklady, k tomu bere v úvahu:

- jaká část transakce již byla provedena,
- kolik dat transakce použila a kolik jich ještě potřebuje pro dokončení,
- kolik transakcí bude třeba celkem vrátit.

Podle těchto kriterií by se mohlo dále stát, že bude vracena stále tatáž transakce a její dokončení by bylo stále odkládáno. Je vhodné, aby systém měl také evidenci o vracených transakcích a při výběru bral v úvahu i tuto skutečnost.

Závěrem si zobrazíme celý postup, jak SŘBD řeší transakce pomocí 2-fázového protokolu při běhu aplikace:



Transakční analýza nad minispecifikacemi

Když jsme se seznámili s teorií transakcí a metodami jejich využití nejen v případě havárie SW nebo HW, můžeme provádět transakční analýzu pro minispecifikace, vytvořené při funkční analýze.

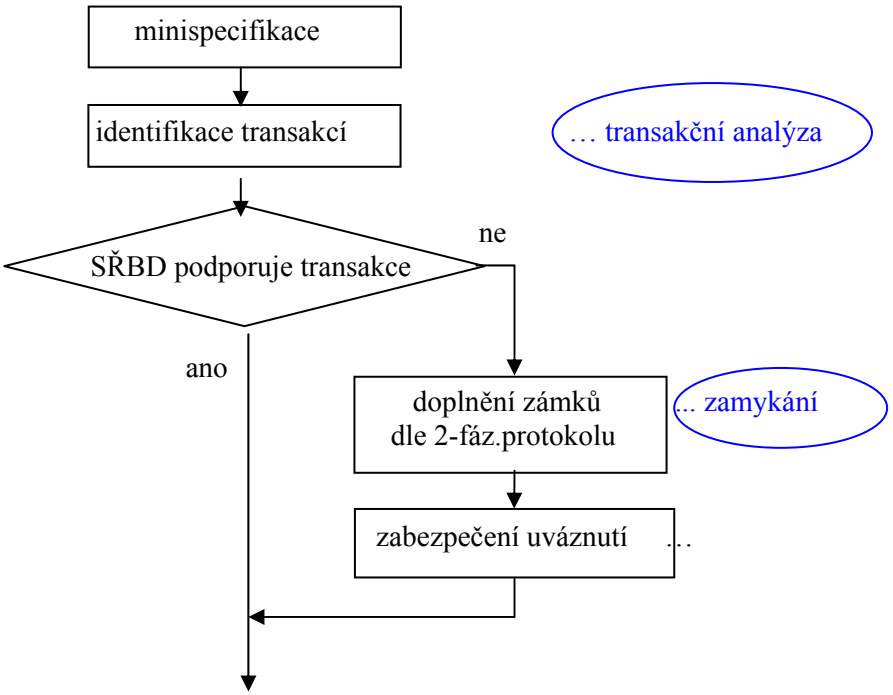
Transakční analýza znamená doplnění algoritmů v minispecifikacích o příkazy označující buď **začátek a konec každé transakce** (pokud použitý SŘBD podporuje transakce), nebo o příkazy **zamykání a odemykání objektů** databáze (pokud nepodporuje transakce).

Princip transakční analýzy spočívá ve **vyhledávání částí algoritmů v minispecifikacích**, které reprezentují samostatné transakce. Přitom se mají dodržovat následující pravidla:

- Obvykle k **jedné minispecifikaci je definována jedna transakce**, protože minispecifikace již odpovídá elementární funkci, která se provádí celá najednou; přitom dále platí
  - transakce je zhruba ohraničena prvním a posledním příkazem, manipulujícím s daty v databázi, nepatří tedy do ní úvodní nebo závěrečné příkazy prováděné v paměti počítače,
  - do transakce nesmí být zahrnuty vstupy uživatele, aby objekty databáze nebyly uzamčeny příliš dlouho, když uživatel vyplňuje data – nebo dokonce si třeba odskočí na svačinu a transakci nechá nedokončenou,
  - příkazy, které jen čtou z databáze (nemění její obsah) buď tvoří samostatnou transakci, nebo nejsou součástí transakce,
  - příkazy modifikující obsah databáze jsou seskupeny a tvoří transakci; někdy bude vhodné dokonce algoritmus minispecifikace upravit tak, aby se modifikující databázové příkazy prováděly v posloupnosti nepřerušené jinými příkazy.

- Podle použitého SŘBD se toto vymezení transakce
  - jen označí (begin transaction ... end transaction, commit, rollback) u SŘBD s podporou transakcí,
  - nebo se pro implementaci doplní o příkazy zamykání a odemykání tabulek nebo záznamů, přičemž se dodržuje dvoufázový protokol, zabezpečí proti uváznutí, řeší se konflikty uživatelů.

Schématicky můžeme tento proces transakční analýzy, prováděný v etapě návrhu implementace, zakreslit následovně:



Příklad:

Zopakujme si jednoduchý příklad algoritmu bankéře pro SŘBD s podporou transakcí. Do minispecifikace (modré příkazy tvořící tělo transakce) jsou přidány červené řádky:

```
begin transaction
    read(A,a);
    a:=a-100;
    write(A,a);
    read(B,b);
    b:=b+100;
    write(B,b);
    if ERROR then rollback
             else commit;
end transaction;
```

tělo transakce

Tentýž příklad, doplněný o zámky pro SŘBD nepodporující transakce. Zámky dodržují dvoufázový protokol. Opět jsou do minispecifikace doplněny červené příkazy.

LX(A)
read(A,a)
a:=a-100
write(A,a)
LX(B)
UN(A)
read(B,b)
b:=b+100
write(B,b)
UN(B)

♦

Algoritmus z minulého příkladu je velmi jednoduchý. Skutečné transakce však mnohdy mají mnoho příkazů a manipulují ne se dvěma, ale s mnoha záznamy databáze. Někdy je nutné při transakční analýze poněkud přeorganizovat vytvořený algoritmus minispecifikace, aby se příkazy manipulující s databází vykonaly najednou těsně po sobě a tak „zdržely“ ostatní transakce na co nejkratší dobu.

Příklad:

Z funkční analýzy máme minispecifikaci 5.3. Seznam chybějícího skladu materiálu.

Nebudeme zde vypisovat celou minispecifikaci, jen naznačíme algoritmus: z tabulky Sklad se přečtou všechny záznamy s množstvím = 0, setřídí se podle typu materiálu a abecedy a vytisknou s firemní hlavičkou, datem a nadpisem „Chybějící materiál skladu“.

Algoritmus obsahuje databázovou operaci přečtení tabulky Sklad, ale jde jen o čtení, které nemůže změnit obsah databáze. Proto se tady transakce označovat nebude. Samotnou operaci čtení z tabulky (zřejmě to bude SQL příkaz SELECT) zabezpečí SŘBD sám – pokud by při ní došlo k chybě, bude ji opakovat.

Ovšem pokud použijeme SŘBD bez podpory transakcí a budeme používat zámky, musíme před příkazem čtení uzamknout tabulku nebo záznamy sdíleně (LS), aby ostatní transakce, které potřebovaly stejnou část databáze také uzamknout, mohly být informovány o tomto zámku. Pak by totiž nemohly uzamknout tutéž část databáze exkluzivně, ale jen také sdíleně. Jinak by musely počkat na ukončení zámku naší transakcí.

♦

Příklad:

Informační systém ABC soukromého zdravotnického střediska s několika lékaři eviduje lékaře, pacienty, objednané pacienty a uskutečněné návštěvy u lékaře i lékařů u pacientů (datum a čas objednaný i realizovaný, diagnóza, výkony, cena pro pojišťovnu).

Lekar (RC_L, jmeno_L, spec)
Pacient (RC_P, jmeno_P, pojistovna)
Navsteva (id_navst, RC_L, RC_P, datum, hodina, diagnoza, id_vykon)
Cisel_vykonu (id_vykon, cena)

Minispecifikace pro funkci 2.3. *Záznam* o návštěvě pacienta = *záznam diagnózy a výkonu je následující, červeně je do ní vepsána transakce.Tučně jsou označeny databázové příkazy, červeně transakce, modře komentář.*

1. **Zobraz seznam objednaných Návštěv pro aktuální den**

Objednávky ze dne …		
jmeno	datum	hodina
…	…	…

2. Lékař vybere objednanou návštěvu aktuálního pacienta.

3. Ulož vybraný záznam do proměnných Pid_navst, Pjmeno, ….
4. Zobraz formulář pro doplnění údajů

Jméno: xxxxxx	... automaticky vyplní vybrané
Datum: xxxxxx	... automaticky systémové datum
Diagnóza:	… zpaměti bez kontroly
Výkon:	… **pomocí nabídky z číselníku**

5. Lékař doplní diagnózu do Pdiag a doplní výkon do Pvykon dle seznamu z číselníku.

begin transaction

6. **Modifikuj záznam** v **Navsteva** (**s Pid_navst)** hodnotami **Pdiag a Pvykon**

end transaction

Příkazy 1. a 4. (čtení výkonů) jsou sice databázové, ale nemodifikují ji a proto nemusí být součástí transakce. Jediný databázový příkaz modifikující je příkaz 6, proto jako jediný tvoří transakci. A pokud ho bude tvořit jediný UPGRADE, ani ten nemusí být nutně označen za transakci.

Označte dále do minispecifikace uzamykání a odemykání objektů databáze s dodržením požadavku sériovosti transakcí.

LS(Navsteva)

1. **Zobraz seznam objednaných Návštěv pro aktuální den**

UN(Navsteva)

Objednávky ze dne …		
jmeno	datum	hodina
…	…	…

2. Lékař vybere návštěvu aktuálního pacienta.
3. Ulož vybraný záznam do proměnných Pid_navst, Pjmeno, ….

LS(Cisel_vykonu)

4. Zobraz formulář pro doplnění údajů

Jméno: xxxxxx	
Datum: xxxxxx	
Diagnóza:	… zpaměti bez kontroly
Výkon:	… **pomocí nabídky z číselníku**

UN(Cisel_vykonu)

5. Lékař doplní diagnózu do Pdiag a výkon do Pvykon dle seznamu z číselníku.

LX(záznam s Pid_navst)

6. Modifikuj aktuální záznam v Navsteva (s Pid_navst) hodnotami Pdiag a Pvykon

UN(záznam s Pid_navst)

U sdílených zámků nemusí být dodržen dvoufázový protokol, protože se data jen čtou a nemodifikují, jediný exklusivní zámek je na jedinou operaci 6.

♦

Příklad:

Databáze IS Sklad obsahuje mimo jiné tabulky

Sklad (karta, nazev, cena_jedn, mnozstvi)

Pohyb (karta, typ_zmeny, datum, mnoz_zmen, cena_prij, id_zakazka, cis_faktura)

Z funkční analýzy máme minispecifikaci 1.1. Příjem materiálu do skladu. *Jde o funkci, při které uživatel postupně zapíše všechen přijatý materiál na sklad, který je na jedné faktuře.*

Vyhledáme v ní transakci a postupně původní algoritmus upravíme.

Nejprve v algoritmu najdeme první a poslední databázový příkaz. Před prvním označíme začátek, za posledním konec transakce (červeně), vpravo jsou komentáře (modře). Tak dostaneme hrubé ohraničení transakce:

1. Zapiš do sestavy hlavičku:

Firma				Strana 1
	Příjemka materiálu			
Zpracováno dne dd.mm.rrrr				.
karta	název	cena jedn.	množství	zakázka .

2. Suma = 0
3. Pro všechen přijatý materiál na faktuře dodavatele proveď
   begin transaction **{před prvním databázovým příkazem}**
   4. **zobraz seznam karet ze Sklad**
   5. uživatel vybere kartu
   6. zapamatuj Sklad.karta, Sklad.cenj a Sklad.mnoz
   7. zobraz formulář příjmu pro vybranou kartu, uživatel vyplní

karta :		vybraná, opsáno ze Sklad, jen pro čtení
název :		opsáno z tabulky Sklad, jen pro čtení
změna :		=1 (příjem), bez editace
datum :		dnešní, možnost přepsat, kontrola na měsíc…
mnoz_zmen:	>0	
cena_prij :		>0 nebo NULL
id_zakaz :		**kontrola na existenci v Zakázka nebo NULL**
cis_fakt :		**kontrola na existenci ve Faktury nebo NULL**

   8. **zapiš vyplněný formulář jako nový záznam do Pohyb**
   9. vypočti nové množství ve skladu, cenu jedn - zprůměrovanou
   10. **modifikuj v záznamu Sklad.karta** hodnoty Sklad.mnoz a Sklad.cenj
   end transaction **{po posledním databázovém příkazu}**
   11. zapiš do sestavy řádek: karta, nazev, cena_prij, mnoz_zmen, id_zakaz
   12. Suma = suma + mnoz_zmen * cena_prij
12. Konec cyklu pro jeden materiál
13. Zapiš na konec sestavy

.		.
. **Celkem**		**suma** .

Tato transakce však nesplňuje všechna výše uvedená pravidla: uvnitř transakce je opakovaně vstup uživatele, příkaz 4 jen čte tabulku Sklad beze změny, v příkazu 10 se modifikuje jen jeden záznam. Proto označení transakce upravíme. První úprava bude taková, že ***každý přijatý materiál bude tvořit samostatnou transakci.***

Pro stručnost další varianty budeme psát bez zobrazovaných rámečků vstupních a výstupních. Ponecháme jen databázové operace.

1. Zapiš do sestavy hlavičku:
2. Suma = 0
3. Pro všechen přijatý materiál na faktuře dodavatele proveď
   begin transaction {není nutné, jen čtení, jediná operace}
   4. **zobraz seznam karet ze Sklad**
   end transaction

5. uživatel vybere kartu
6. zapamatuj Sklad.karta, Sklad.cenj a Sklad.mnoz do pole PSklad
7. zobraz formulář příjmu pro vybranou kartu
8. uživatel vyplní

...	
id_zakaz :	{kontrola na existenci v Zakázka, čtení}
cis_fakt :	{kontrola na existenci ve Faktury, čtení}

9. zapiš vyplněný formulář jako nový záznam do pole PPohyb

begin transaction

10. **přečti aktuální záznam ze Sklad**
11. **zapiš nový záznam z PPohyb do Pohyb**
12. vypočti nové množství ve skladu a cenu jedn - zprůměrovanou
13. **modifikuj aktuální záznam ve hodnotami Sklad.mnoz a Sklad.cenj**

end transaction **{po posledním databázovém příkazu}**

14. zapiš do sestavy řádek: karta, nazev, cena_prij, mnoz_zmen, id_zakaz
15. Suma = suma + mnoz_zmen * cena_prij

16. Konec cyklu pro jeden materiál
17. Zapiš na konec sestavy Suma
18. Zobraz uloženou výstupní sestavu

Jak je vidět, vymezili jsme modifikující záznamy do jednoho bloku neobsahujícího vstup uživatele. Hodnoty načtené od uživatele nebo vypočtené ukládáme zatím v paměti.

Takto je ošetřen jako transakce příjem každého materiálu samostatně (2 změny v databázi – jeden nový záznam a jedna modifikace) tvoří transakci, mezi nimi nesmí dojít k přerušení, protože by nesouhlasilo množství na skladě se záznamem o přijatém množství).

Ovšem skutečná funkce příjmu veškerého zboží z faktury na sklad vyžaduje více: kontrolu, že nedošlo k nějaké chybě uživatele při zadávání karty, množství, ceny. Proto bude vhodné definovat příjem všeho materiálu jako jednu transakci. Na konci vstupního cyklu uživatel vidí na výstupní sestavě, jestli spočítaná suma odpovídá skutečné sumě na faktuře. Pokud ano, teprve je transakce připravena. Pokud ne, vrátí se uživatel k chybným zápisům a opraví je. To vše vede k novému řešení, že dokud není potvrzena suma, nebude se zapisovat do databáze. Průběžná data se budou ukládat v paměti a teprve na závěr se zapíší do databáze.

Poslední řešení tedy znovu vyžaduje úpravu algoritmu minispecifikace s použitím paměťových polí PSklad a PPohyb. Opět pro stručnost vynecháváme zobrazení vstupů a výstupů.

1. Zapiš do sestavy hlavičku.
2. Suma = 0
3. Pro všechen přijatý materiál na faktuře dodavatele proveď
4. **načti seznam karet ze Sklad** do pole NSklad (karta, nazev, …)
5. zobraz seznam karet z PSklad (karta, nazev, …)
6. uživatel vybere kartu
7. zobraz formulář příjmu pro vybranou kartu,
8. uživatel vyplní … mnozpri, cenpri
9. zapiš vyplněný formulář jako nový záznam do **pole PPohyb**
10. Konec cyklu pro jeden materiál
11. Zobraz kontrolní výpis celé příjemky z PPohyb

begin transaction

12. Pro všechny hodnoty **karta** z PPohyb proveď
    13. **záznam z PPohyb zapiš jako nový do Pohyb**
    14. **přečti záznam s hodnotou karta ze Sklad do PSklad**
    15. vypočti nové množství PSklad.mnoz=PSklad.mnoz + PPohyb.mnozpri
    16. vypočti zprůměrovanou cenu jednotkovou PSklad.cenj = …..
    17. **modifikuj aktuální záznam v Sklad - hodnoty PSklad.mnoz,PSklad.cenj**
18. konec cyklu pro záznamy z PPohyb

end transaction

Toto řešení splňuje všechna pravidla: vstup uživatele je mimo transakci, jeho data jsou v paměti, všechny změny v databázi jsou v bloku (cyklu) příkazů prováděném bez zdržování.

Navíc je vidět, jak se liší elementární funkce (celá minispecifikace) od vlastní transakce (části, která modifikuje databázi).

♦

Příklad:

V IS Banka je definována databáze účtů a nad ní se provádějí tyto transakce:

Převod z účtu na jiný účet.
Vklad na účet.
Výběr z účtu.
Platby inkasa pro některé klienty.
Platby bance (na účet v téže bance) za vedení účtů pro všechny klienty.
Připisování úroků všem klientům

Každá z nich je definována jako 1 transakce, tedy hromadné transakce realizují modifikaci všech účtů. Předpokládáme dodržení dvoufázového protokolu.

.

účet	**suma**
...	
účet pana A	1000
...	
účet pana B	3000
...	
účet pana C	2000

Určete, u které z následujících dvojic transakcí může dojít k uváznutí, pokud jsou prováděny současně.

1. Pan A platí 100.- panu B, pan B vybírá 200.-

Nemůže dojít k uváznutí, protože 1. transakce zamyká 2 záznamy, ale druhá jen jeden. Která zamkne jako první záznam pana B, ta první skončí. Zatím druhá počká a po skončení první dokončí své operace.

2. Pan B vrací panu A, pan A platí panu B.

Může dojít k uváznutí, když jedna transakce zamkne záznam B, potom druhá záznam A. První nemůže zamknout A, čeká, druhá ale nemůže zamknout B a také čeká. Čekají navzájem.

3. Pan A platí panu B, pan B platí panu C.

Nemůže dojít k uváznutí, protože používají jediný společný záznam. Která jej zamkne prvně, ta se první dokončí. Po jejím skončení dokončí druhá.

4. Všem jsou připisovány úroky, pan A platí panu B.

Může dojít k uváznutí, protože první transakce postupně zamyká všechny záznamy, až po posledním zámky všechny zpracované odemyká. Mezitím druhá transakce zamkne dosud volný záznam A. Pokud záznam B je již zamčený 1. transakcí, druhá čeká na jeho odemknutí. Prvá transakce však nemůže skončit, protože až dojde k záznamu A, také čeká na jeho odemknutí.

♦

3.6. Shrnutí transakční analýzy a realizace transakcí

Závěrem výkladu o transakcích shrneme, kdy a kdo během životního cyklu vývoje IS pracuje s transakcemi.

Během zadání a analýzy se transakce ještě neřeší. Jsou to etapy upřesnění věcných a organizačních požadavků a modelování budoucího systému bez ohledu na jeho budoucí implementaci.

V etapě návrhu implementace provádí řešitel IS, informatik-návrhář, který zná dobře teorii transakcí, transakční analýzu:

- pro každou minispecifikaci, ve které se modifikuje obsah databáze, vyznačí začátek a konec transakce, případně algoritmus nejprve upraví tak, aby všechny databázové operace uvnitř transakce nebyly prokládány vstupy uživatele nebo složitými výpočty;
- pokud použitý SŘBD podporuje transakce, je analýza hotova;
- pokud SŘBD nepodporuje transakce, uvnitř označené transakce návrhář
  - doplní zámky (tabulek nebo záznamů) pomocí 2-fázového protokolu
  - zabezpečí transakce proti uváznutí některou z metod, například
    - časovým omezením pokusů o zámek,
    - lineárním uspořádáním zámků
    - konstrukcí funkcí znovuspustitelných a upozorněním uživatele, ať po chybě pustí funkci znovu.

V etapě implementace se realizují příkazy podle návrhu v konkrétním jazyce a podle syntaxe odpovídajícího SŘBD.

Při provozu s paralelním spouštěním aplikací

- pokud použitý SŘBD podporuje transakce, provádí podle svého způsobu zabezpečení transakcí
  buď hlídání a rozpoznání uváznutí s vycouváním
  nebo plánování spouštění transakcí pomocí některého typu plánovače.
- pokud SŘBD nepodporuje transakce, realizuje implementovaný program s jeho příkazy zámků a dalšího naprogramovaného řízení funkcí; pokud dojde k chybě, řeší ji uživatel například novým spuštěním funkce.

Shrnutí pojmů 3.3. – 3.6.

Transakce, atomická funkce.
Metody zabezpečení transakcí, přímá a zpožděná aktualizace, log-soubor.
Obnova databáze po havárii.
Paralelní provádění transakcí, sériovost transakcí, dvoufázový protokol.
Zamykání objektů databáze, uváznutí, řešení uváznutí. Plánovače transakcí.

Otázky 3.3. – 3.6.

1. Jak může dojít při běhu aplikační úlohy ke ztrátě konzistence databáze?
2. Co je transakce a jak se zabezpečuje její atomická vlastnost?
3. Jakými metodami jsou v SŘBD podporovány transakce?
4. Jak SŘBD zabezpečuje databázi proti HW chybám?
5. Co je sériovost transakcí při víceuživatelském využívání databáze?
6. Proč se transakce při víceuživatelském provozu nespouštějí sériově za sebou?
7. Jakými metodami se zabezpečí sériovost transakcí?
8. Co je zamykání objektů databáze a k čemu se používá?
9. Jaké typy zámků objektů databáze existují?
10. Jak se zajišťuje sériovost transakcí zamykáním?
11. Co je uváznutí?
12. Jak se SŘBD s podporou transakcí brání proti uváznutí?
13. Jak se rozpozná uváznutí při běhu aplikací?
14. Co jsou plánovače transakcí a jaký je jejich princip?

3.7. Návrh modulů a modulové schéma

Probrali jsme řadu úkolů, které musí provést návrhář implementace, aby výsledný IS byl nejen správný, ale i efektivní a schopný správně fungovat i v provozu, kdy databázi současně využívá mnoho uživatelů.

Ale ještě není vše hotovo. Jsou upraveny a doplněny původní minispecifikace (spojeny podobné funkce do jedné, doplněny indexy, transakce, provedena optimalizace přístupu k datům atd.), je doplněna řada systémových funkcí (o kontroly stavů, zálohování, archivace, hlídání uživatelů a jejich přístupových práv atd.) a jsou doplněny některé systémové tabulky nebo atributy.

Závěrem návrhu se provádí návrh modulů.

Definice:

Modulem na úrovni implementace rozumíme samostatnou programovou jednotku, volatelnou část programového systému, rozlišitelnou při překladu. Může to být procedura nebo funkce v nějakém programovacím jazyce, řada procedur či celý program. Obecně má tyto aspekty:

- **název**
- **definované vstupy a výstupy** (údaje od volajícího modulu a údaje volajícímu modulu vracející)
- **definované funkce** (co modul dělá při transformaci vstupů na výstupy)
- **způsob práce** (vnitřní logika modulu, algoritmy, kód procedury)
- **interní data** (lokální data, vlastní pracovní oblast modulu)
- **volané funkce a procedury uvnitř**
- příp. další atributy.

Jde tedy o rozhodnutí, zda všechny funkce a procedury budoucího programu budou realizovány jediným zdrojovým souborem = modulem, nebo každá funkce a procedura bude samostatným

souborem (obě možnosti jsou krajní a obvykle nevhodné), nebo se funkce rozdělí do několika zdrojových souborů = modulů a jak. Mnoho modulů (u velkého IS to jsou stovky i tisíce) je nepřehledných, když je málo modulů, obsahuje každý mnoho funkcí a hůře se ladí. Úkolem je najít nejlepší rozdělení, aby jich na jedné straně nebylo příliš mnoho a na druhé straně aby v jednom moduly byly funkce nějak „příbuzné“ a dobře se s nimi pracovalo.

Důležitým dalším důvodem, proč se navrhují předem moduly, je možnost rozdělení programátorské práce na více programátorů. Každý z nich dostane své moduly pro implementaci. Když je modulové schéma dobře navrženo, může pracovat každý samostatně a vzájemné konzultace jsou omezeny na minimum.

Definuje se tedy pojem modularita programu:

Modularita vyjadřuje míru rozkladu programu na moduly takové, že změna jednoho modulu má minimální vliv na ostatní moduly.

Modulární návrh může být vytvářen teoreticky

- **dle datových struktur -** všechny funkce ke stejné entitě nebo skupině entit tvoří modul,
- **dle typu funkcí -** všechny vstupní formuláře v jednom modulu, všechny reporty v dalším, všechny výpočty v dalším, systémové funkce v dalším, …
- **dle datových toků** – funkce se stejnými přenosy dat tvoří modul.

Příklad:

Buduje se rozsáhlý IS výrobní firmy XYZ, obsahující subsystémy Zakázky, Sklad, Výroba, Účetnictví,Zaměstnanci, Majetek. Každý subsystém používá mnoho tabulek, některé společně. Každý subsystém obsahuje několik desítek funkcí (z funkční analýzy) a celý IS několik desítek dalších funkcí systémových, uživateli skrytých.

Návrh modulů podle dat znamená spojit všechny funkce zakázky, všechny skladu, všechny výroby atd. vždy do jednoho modulu. Výhodu to má v tom, že tyto funkce pracují se stejnými daty a chyba nebo změna funkcí skladu se snadno hledá, případné změny se provádějí v rámci jednoho modulu. Nevýhodu, že modul obsahuje jak vstupy, tak výpočty a výstupní sestavy. Nejčastěji se totiž přidávají později právě další reporty a „specialisté“ na reporty pak pracují s velkým modulem. Varianta je výhodná pro velkou zapouzdřenost dat, modul minimálně spolupracuje s jinými moduly.

Návrh modulů podle funkcí znamená spojit všechny vstupní formuláře všech subsystémů do jednoho modulu, všechny výpočty, všechny reporty apod. „Specialisté“ na formuláře, na reporty, na výpočty tak mají pohromadě stejný typ funkcí a snadněji, jednotným stylem je vytvářejí.Tak jsou pohromadě reporty ze zakázek, skladu, výroby atd. Ovšem tak změna či oprava v jednom subsystému musí někdy zasáhnout do několika modulů. Tato varianta je pohodlná při tvorbě, nevýhodná při realizaci dalších oprav a změn.

Zdá se, že návrh podle datových toků není příliš rozdílný od dělení podle dat, že stejné toky vedou k nebo od stejných dat Ale přece jen stejný datový tok může vést k různým funkcím různých subsystémů. Například data o zaměstnanci tečou do zakázek, do účetnictví, do zaměstnanců, prakticky do všech subsystémů. Tak se dostanou do modulu funkce téměř navzájem nesouvisející, zapouzdřenost dat je minimální.

♦

Nejvýhodnější se jeví dělení modulů podle dat. Přesto nastávají případy, kdy je vhodné vytvořit modul podle funkcí. Jde například o systémové funkce, které se používají u mnoha funkcí elementárních z funkční analýzy. Ty pak je vhodné zařadit do jednoho nebo několika „systémových“ modulů. Příkladem jsou mnohé kontroly uživatelů a jejich práv, různá chybová a informační hlášení, kontroly stavů systémů (ne stavů entit, ty bývají uvnitř elementárních funkcí) apod.

Další výhodou takových systémových modulů je to, že u prvního vytvářeného IS se realizují, u dalších IS se mnohé z nich mohou opakovaně využívat. Když tvoří samostatné moduly, jen se k novému IS přiřadí. Pokud by tyto systémové funkce byly součástí modulů jiných, musely by se pro opakované použití „vytahovat“ a přenášet do nového IS.

Výsledkem celé etapy návrhu implementace je **modulové schéma**, obsahující rozdělení všech algoritmů upravených a doplněných algoritmů elementárních a funkcí systémových do modulů.

Funkce a moduly jsou úplným a podrobným zadáním programátorům pro implementaci.

Shrnutí pojmů 3.

Etapa návrhu implementace.
Systémový návrh, architektury informačních systémů.
Vlastní návrh a jeho úkoly.
Transakce, transakční analýza.
Modulové schéma funkcí, modularita.

Otázky 3.

1. Co je úkolem etapy návrhu implementace?
2. Jaké základní části návrhu rozeznáváme?
3. Jaké jsou základní typy architektury IS?
4. Co všechno patří k úkolům vlastního návrhu implementace?
5. Popište každý úkol návrhu implementace.
6. Co je modul a co je modulové schéma IS?
7. Podle jakých pravidel se seskupují funkce do modulů?
8. Proč se dělí funkce systému do modulů?

Příprava na tutoriál - Zadání semestrálního projektu

Pro vlastní úlohu informačního systému proveďte nejdůležitější potřebné kroky z návrhu implementace: doplnění a optimalizaci minispecifikací, indexovou a transakční analýzu, potřebné další systémové funkce a návrh modulů.

4. IMPLEMENTACE INFORMAČNÍHO SYSTÉMU

Čas ke studiu kapitoly: 2 hodiny studium + 2 hodiny řešení úloh

Cíl Po prostudování této kapitoly budete

- vědět, jaké dokumenty patří k úplnému popisu informačního systému a co je jejich obsahem,
- vědět, jaké typy testování informačního systému rozeznáváme,
- umět napsat úplnou dokumentaci informačního systému.

Výklad

4.4. Etapa implementace IS

Nejpozději v návrhu implementace bylo rozhodnuto o použitém prostředí pro implementaci. Z etapy návrhu jsou podrobně zpracovány podklady pro implementaci, tedy vlastní implementování je převážně rutinní programátorskou prací.

Nejprve se definuje databáze a všechny její relace = tabulky včetně všech deklarovatelných integritních omezení. Potom se přistoupí k vlastnímu kódování, u velkého IS většinou paralelně několika programátory. Každý má ke zpracování své moduly.

Vlastní programování v této učebnici probírat nebudeme. Předpokládáme u čtenáře znalost základů programování i znalost již zde probrané teorie. Programovací jazyk a další nástroje SŘBD jsou uvedeny v manuálech.

Součástí etapy implementace je také vytvoření dokumentace a testování výsledných programů. Proto probereme ještě tyto dvě související práce.

4.2. Dokumentace IS

Pro IS se obvykle postupně vyskytují tyto druhy dokumentace:

❑ Dokumentace ke specifikaci zadání

Zpracovává se na začátku řešení, obsahuje globální definice problému, funkční a nefunkční požadavky. Vypracuje zadavatel, často pak upřesňuje s analytikem. Z kapitoly o zadání víme, jaké informace má obsahovat, včetně modelů vnějšího chování systému.

Obecně obsahuje vše, co musí vědět řešitel, aby mohl analyzovat, navrhnout a pak realizovat systém.

❑ Dokumentace k analýze systému

V průběhu analýzy se dokumentují všechny zpracované modely. Jsou to

Datová analýza, obsahující úplné konceptuální schéma, ERD + datový slovník + integritní omezení.

Funkční analýza, obsahující hierarchii DFD + minispecifikace.

Dynamická analýza, obsahující stavové diagramy celého systému, jeho částí až po stavové diagramy důležitých entit.

Návrh komunikace, obsahující diagram struktury komunikace – návrh vzhledu a ovládání menu, formulářů, reportů = výstupních sestav, dialogů s uživatelem, chybových a informačních hlášení apod.

❑ Dokumentace k návrhu implementace

Zdokumentovaný návrh implementace by měl obsahovat nejprve popis a zdůvodnění všech koncepčních rozhodnutí: použitý programovací a komunikační jazyk, personální zabezpečení systému, hardwarové zabezpečení systému, odhad ceny, plánovaný přínos systému (úspory nákladů, pracovních sil, větší rozhodovací schopnosti ap.). Vše jako plánovaný projekt, případně v i několika variantách.

Druhá detailní část návrhu obsahuje

- doplněný datový model (3. úroveň ERD) o doplněné atributy, doplněné dočasné a systémové tabulky, doplněné datové toky ze systémových funkcí,
- doplněné minispecifikace o mnoho detailů uvedených v kapitole 3: zpřesněných algoritmů, optimalizovaných přístupů do databáze, upravených algoritmů pro vhodnou realizaci transakcí, doplněné systémové kontroly stavů, uživatelských přístupů atd.,
- algoritmy přidaných systémových funkcí.

Druhá část dokumentace popisuje podrobně skutečnou realizaci, implementaci.

❑ Uživatelská dokumentace - manuál

Uživatelský manuál programového systému je podrobný návod pro koncového uživatele. Měl by obsahovat tyto informace:

- specifikace zadání a základní logická struktura systému,
- na jakém HW a s jakým SW je program provozovatelný,
- instalace systému,
- jak spustit program,
- jak se program obsluhuje obecně, jak se ovládá menu, jak se dělí do nižších úrovní a seznam základních funkcí programu,
- se kterými datovými soubory systém pracuje + přístupová práva k souborům, záznamům a atributům pro různé uživatele,
- seznam vstupních obrazovkových formulářů,
- seznam výstupních sestav,
- popis každé elementární funkce, její funkčnost a ovládání,
- často kladené otázky.

Uživatelská příručka by měla být zabudovaná formou nápovědy v implementaci IS.

❑ Programátorská dokumentace

Pro případ, že na údržbě nebo úpravách programu budou spolupracovat další programátoři, i pro vlastní zdokumentování složitého programu je nutná dokumentace o implementaci systému. Ta musí obsahovat (mimo velmi vhodné podrobné komentáře ve zdrojových kódech programu) všechny důležité informace o použitém HW a SW, OS a SŘBD a hlavně o vlastní aplikaci.

Důležité je udržovat dokumentaci aktuální při všech změnách a doplňcích. Šetří to mnoho času při údržbě, opravách i vývoji nových verzí.

4.3. Testování, validace, verifikace IS

Současně s implementací se začíná provádět kontrola správnosti výsledného produktu - programu. Kontrolu správnosti chápeme v několika významech:

Validace je ověření, že produkt odpovídá představám uživatele ve všech možných případech.

Verifikace je ověření, že produkt odpovídá specifikaci ve všech možných případech.

Testování je ověřování programu pomocí konečné sady příkladů. Testováním není obecně možné dokázat správnost programu, protože vždy může existovat nějaký neotestovaný případ. Testováním tedy můžeme odhalit přítomnost chyb, ale nemůžeme dokázat jejich nepřítomnost. Za nejúspěšnější považujeme ty testy, které odhalí chyby. Návrh testů bývá proto vytvářen „proti“ jejich tvůrcům a je vhodné, aby testér byl osobou, která není na tvorbě programu přímo zúčastněna.

Spolehlivost programu znamená

- že při činnosti programu se nevyskytne žádná chyba během určité dostatečně dlouhé doby;
- pravděpodobnost toho, že během určitého časového intervalu nepřevýší náklady vzniklé uživateli chybou systému určitou výši.

Úkolem testování je odhalovat chyby. Informační systém obsahuje chybu, jestliže
- jeho chování neodpovídá zadání;
- při zadání vstupů z předem určené množiny hodnot neodpovídá požadovaným výsledkům;
- neodpovídá dokumentaci a uživateli poskytovaným informacím (helpům ap.);
- nepracuje tak, jak od něj uživatel rozumně očekává.

Shrnutí kapitoly 4.

Po provedené a odsouhlasené analýze IS se přistupuje k jeho realizaci. Před vlastní implementací je nutné, aby specialista-návrhář provedl návrh implementace (design), ve kterém popisuje technické provedení všech funkcí, které byly dosud popsány jen na logické úrovni. U elementárních funkcí z minispecifikací se zpřesňují a optimalizují algoritmy. Přibývají další systémové funkce zabezpečující mnohé kontroly z dynamické analýzy, funkce řešící přístupová práva uživatelů a další funkce, o kterých ještě nevíme. Součástí návrhu je také rozdělení budoucího kódu do modulů tak, aby se při implementaci neopakovaly části kódů a aby se i více programátorům dobře spolupracovalo.

Otázky 4.

1. Jaké druhy dokumentace patří k jednotlivým etapám vývoje projektu?

2. Co všechno patří do dokumentace k analýze?
3. Co všechno patří do programátorské dokumentace?
4. Co všechno patří do uživatelské dokumentace?
5. Jaké typy testování IS existují a jaký je mezi nimi rozdíl?

Příprava na tutoriál - Zadání semestrálního projektu

Implementujte vlastní úlohu informačního systému podle provedené analýzy a návrhu implementace.

Napište uživatelskou a programátorskou dokumentaci svého IS.

5. PŘEDÁNÍ DO PROVOZU A PROVOZ IS

Čas ke studiu kapitoly: 2 hodiny studium + 1 hodina řešení úloh

Cíl Po prostudování této kapitoly budete

- vědět, co všechno je součástí etapy předání informačního systému do provozu,
- vědět, jaké jsou druhy údržby informačního systému,
- umět připravit prezentaci informačního systému jako součást jeho předávání.

Výklad

5.1. Předávání do provozu

❑ Příprava pro předání do provozu

Je-li IS hotov, je připraven k předání uživatelům. Ovšem tato etapa neznamená jen nainstalování systému. Jde o relativně dlouhý proces, který je nutné dlouho předem připravit. Jinak se může stát, že uživatelé nebudou pozitivně spolupracovat, budou hledat chyby a zdůvodňovat, proč je systém špatný atd. V nejhorším případě zadavatel systém nezaplatí nebo zdržuje platbu.

Proto příprava předání po stránce technické, pedagogické i psychologické se velmi vyplatí. Co je tedy třeba připravit předem:

- Již v průběhu specifikace a analýzy je vhodné **spolupracovat se všemi** budoucími **uživateli**, ne jen s oficiálním zadavatelem. Požádáme všechny, aby nám upřesnili zadání, odsouhlasili analýzu, odsouhlasili návrh komunikace. Průběžně zdůrazňujeme, že bez jejich spolupráce by úplné a správné řešení nebylo možné (i kdyby tomu tak zcela nebylo). Tak z nich uděláme spoluřešitele spoluzodpovědné alespoň za věcnou stránku řešení. Pak se dost dobře nemůže stát, že nakonec uživatel vše neguje, odsoudil by i sám sebe.
- Již při zadání zjistíme, v jakém tvaru či formátu jsou uložena **data dosud**. Pokud jsou pouze v papírové podobě, využijeme jich při zaškolování uživatelů (viz níže). Pokud jsou uložena v nějakém jiném dosavadním IS, zjistíme jejich obsah i formát a průběžně s implementací vytvoříme a otestujeme **konverzní programy** z tohoto starého IS do nového.
- Pokud s IS spolupracují **jiné IS** jako aktéři – našemu IS posílají informace nebo náš IS si je sám načítá, případně náš IS posílá jim informace – pak předem nejen otestujeme oboustranně propojení, ale dohodneme se správci těchto systémů podmínky provozu, vzájemná přístupová práva a termín spuštění ostrého provozu vzájemného propojení.
- Před zaškolováním uživatelů se jim předá **uživatelský manuál** v písemné podobě, aby si mohli předem prohlédnout jednak jeho strukturu, seznámit se se stylem manuálu a pročíst si funkce systému, se kterými budou pracovat.
- Na konci implementace postupně organizujeme **zaškolování** jednotlivých typů uživatelů (rolí) na kopii systému, kde mohou zkoušet práci s jakýmikoliv daty. Do školícího systému předem

přeneseme část jejich dosavadních dat, aby fungovaly všechny funkce (uživatelé si nemuseli nejprve mnohá data sami ukládat) a tak je i testovali. Zaškolování se provádí po časových etapách. Až uživatelé zvládnou jednu část, přidá se jim další. Pokud se najednou seznámí se všemi novými možnostmi, mohou se cítit zahlceni mnoha možnostmi systému a mít pocit, že to vše nezvládnou.

- Pokud dosavadní data jsou pouze v papírové podobě, je možné požádat uživatele, aby si v rámci zaškolování naplnili alespoň některé tabulky předem.

❑ Vlastní předání IS do provozu

Je-li IS odladěn, uživatelé zaškoleni, konverzní programy připraveny a otestovány, může dojít k vlastnímu předání IS do provozu.

Pokud se spouští nový IS poprvé, přechází se z papírové evidence na databázový systém, pak záleží jen na harmonogramu domluveném se zadavatelem. Obvykle nějakou přechodnou dobu trvá, než se data z papírové podoby přenesou do databáze a než se spustí plný provoz IS. Je vhodné tyto přechodné operace předem naplánovat s jednotlivými uživateli a předem jim vysvětlit nutnost dočasné práce navíc proti jejich běžným povinnostem.

Pokud se přechází na nový IS z dřívějšího již nevyhovujícího systému (často z několika různých evidencí, pokrývajících jednotlivé části nového IS), pak se obvykle provede přechod od předem definovaného data, například od 1. dne v měsíci.

Je nutné v tomto případě provést pomocí připravených konverzních programů „překlopení“ starých dat do nové databáze těsně před spuštěním provozu. Nikdy se nesmí uživatelé nutit k novému ručnímu vkládání dat.

I po dokonalé spolupráci s uživateli, po jejich zaškolení, po předání IS do provozu s aktuálními daty z dřívějších evidencí je vhodné, aby po nějakou dobu byl přítomen zástupce řešitele pro řešení jakýchkoliv počátečních problémů – od rad a povzbuzení uživatelům až po evidenci a případné odstraňování chyb.

Je-li nový IS velmi rozsáhlý nebo zadavatel má řadu odloučených pracovišť, spouští se nový provoz postupně. Podle harmonogramu se spouští například postupně jednotlivé subsystémy nebo jednotlivé provozy. Organizace a propojení takových částí je obvykle mnohem náročnější.

Příklad:

Po skončení pracovní doby předcházejícího dne se dostaví na pracoviště zadavatele řešitelé nového IS, spustí všechny konverze a všechna data zkopírují do nové databáze.

Uživatelé přijdou druhý den do práce, jsou poučeni a zaškoleni, mají všechna aktuální data v novém IS, takže přirozeně pokračují ve své běžné práci, ale již na novém IS.

♦

5.2. Provoz a údržba

Závěrečnou částí životního cyklu programového díla je údržba produktu. Bývá často podceňována, přesto že zabírá asi 80% života programu. Pro údržbu by měli být určeni velmi zkušení informatici, kteří znají dokonale systém a jeho funkce včetně všech detailů. Bohužel se často stává, že se údržba podceňuje, svěří se nepříliš zkušeným pracovníkům a ti mohou z neznalosti souvislostí nadělat mnoho škod dílčími „opravami“ chyb nebo jinými zásahy do systému.

Definice:

Softwarová údržba je **modifikace softwarového produktu po jeho předání** za účelem opravy chyb, zlepšení výkonnosti nebo dalších atributů nebo přizpůsobení produktu změněnému prostředí.

Druhy údržby SW

Rozlišujeme 4 kategorie údržbových prací:

1. **Opravárenská údržba** odstraňuje nalezené chyby. Provádí se v rámci reklamací systému.
2. **Adaptivní údržba** přizpůsobuje SW změnám prostředí, jako je nový HW nebo nová verze OS. Nemění funkce systému. Provádí se na objednávku provozovatele IS.
3. **Zdokonalovací údržba** zahrnuje do systému nové nebo změněné požadavky uživatele a vede tak ke změně funkcí (zlepšení?) systému. Někdy se tento typ údržby používá i ke zvýšení výkonnosti systému nebo zlepšení uživatelského rozhraní. Provádí ji řešitel na základě vlastních analýz provozu, zkušeností uživatelů nebo požadavků uživatelů.
4. **Preventivní údržba** zahrnuje aktivity zaměřené na zlepšení udržovatelnosti systému, jako aktualizace dokumentace, doplnění komentářů, zlepšení modularity ap.

Shrnutí pojmů 5.

Předání informačního systému uživateli, zaškolení, dokumentace.
Konverze starých dat do nového IS.
Druhy údržby IS.

Otázky 5.

1. Co je úkolem implementace IS, co je pro něj zadáním a co výsledkem?
2. Kdy se dělá dokumentace k IS a které druhy dokumentace rozlišujeme?
3. Co je úkolem testování hotového IS a které typy testování rozlišujeme?
4. Co všech zahrnuje údržba IS a jaké typy údržby rozlišujeme?

Úlohy k řešení 5.

Příprava na tutoriál - Zadání semestrálního projektu

Připravte prezentaci svého informačního systému.

Implementovanou vlastní úlohu informačního systému předejte podle pokynů svému cvičícímu nebo tutorovi současně s úplnou dokumentací systému.

6. DISTRIBUOVANÉ INFORMAČNÍ SYSYTÉMY

Čas ke studiu kapitoly: 3 hodiny studium + 2 hodiny řešení úloh

Cíl Po prostudování této kapitoly budete

- znát definice distribuované databáze a distribuovaného informačního systému,
- vědět, jaká nové problémy je třeba vyřešit při budování distribuovaného IS,
- znát výhody a nevýhody centrálního a decentralizovaného řízení distribuovaného IS,
- umět navrhnout rozmístění dat pro distribuovanou databázi,
- umět navrhnout vhodnou frekvenci aktualizací replik dat,
- umět rozpoznat správné zpracování dat v centrálním a decentralizovaném řízení.

Výklad

6.1. Distribuovaná databáze

❑ Rozsáhlé územně vzdálené databáze

Vývoj systémů se vzájemným propojením výpočetní a komunikační techniky začal asi ve druhé polovině sedmdesátých let. V současnosti jsou počítačové sítě a internet samozřejmostí.

Probrali jsme podrobně IS s víceuživatelským provozem a jejich problémy. Předpokládali jsme zatím mlčky provoz na lokálních sítích se společnou databází, k níž přistupuje současně řada uživatelů.

Ovšem vzniká stále častěji potřeba sdílet společná data subjekty vzájemně vzdálenými. Mnohé firmy mají řadu poboček územně vzdálených, podobně banky, pojišťovny, knihovny a mnohé další. V době internetu by se zdálo přirozené sdílet jednu společnou databázi na nějakém centrálním serveru a spojení provozovat právě prostřednictvím internetového IS. Pak by se celý IS choval podobně jako lokální síť, jen data by „cestovala“ na velké vzdálenosti.

Jde-li o informační systém, určený skutečně jen k podávání informací uživatelům *(jízdní a letové řády, mapy a podobné)*, z nichž se jen čte, je tato forma přístupu k datům vhodná. Ale velké firemní systémy, kde se průběžně denně manipuluje s obrovským množstvím dat, kde se provádí množství automaticky spouštěných transakcí, kde každá pobočka má svá data, která by však měla být dostupná i ze všech ostatních poboček, tam by jedna centrální databáze a přenosy dat po síti prostě nestačily kapacitou ani rychlostí.

Technická realizace takového systému se musí podřídit potřebám aplikační úlohy. Technické možnosti komunikačních systémů a vzájemného propojování lokálních počítačových sítí umožňují spojovat i původně samostatně pracující počítače do **distribuovaných systémů (DBS)**. To má řadu výhod, jako možnost zálohování počítačů, možnost sdílení společných informačních fondů, racionalizace provozu. U databázových systémů pak nazýváme územně rozložené lokální báze dat podřízené jednotnému řízení distribuovanými bázemi dat. Jednotné řízení distribuované báze může být realizováno různými formami na různém stupni centralizace řízení.

Proti klasickým databázovým systémům se u distribuovaných databází vyskytují problémy nového typu. Jde o problémy s územní distribucí dat, komunikací v počítačových sítích, kooperací systémů založených na různých datových modelech nebo realizovaných různými jazyky apod. Jsou to komplikované a vzájemně propojené problémy.

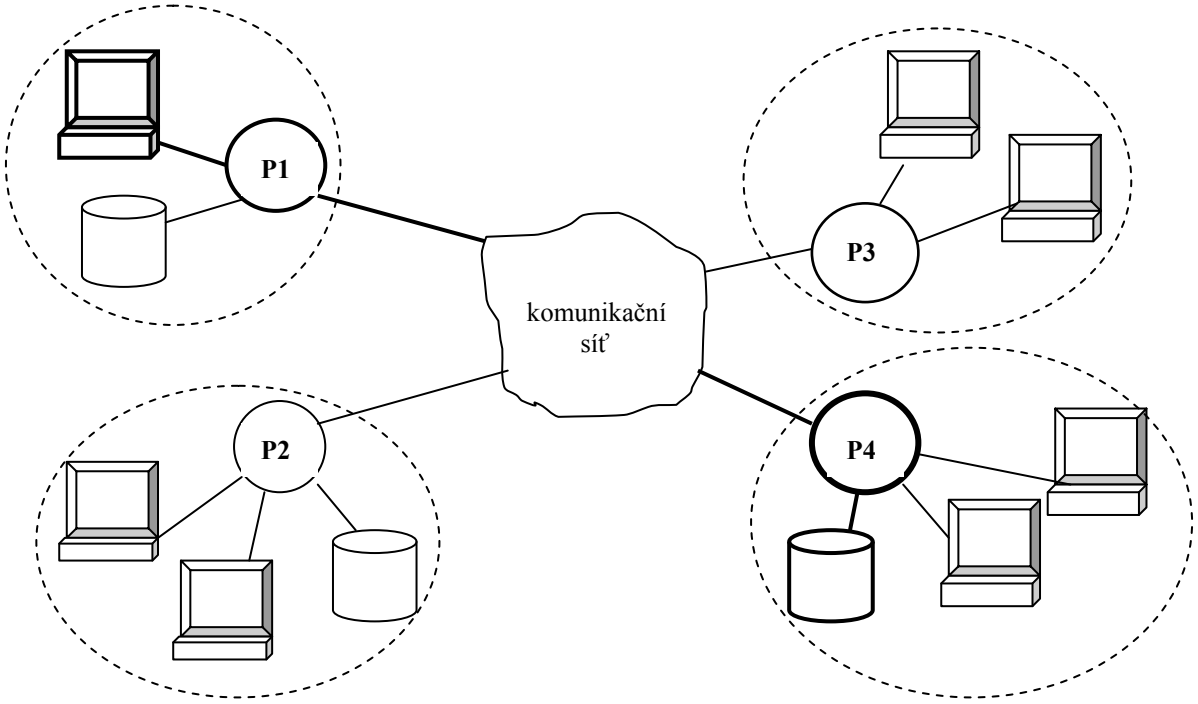
V následujícím výkladu si uvedeme hlavní hlediska, která se mohou použít při klasifikaci a návrhu distribuovaných databázových systémů. Jsou to

- modely dat lokálních databází
- rozmístění dat v distribuované databázi
- těsnost spojení lokálních databází
- stupeň centralizace řízení distribuovaných transakcí

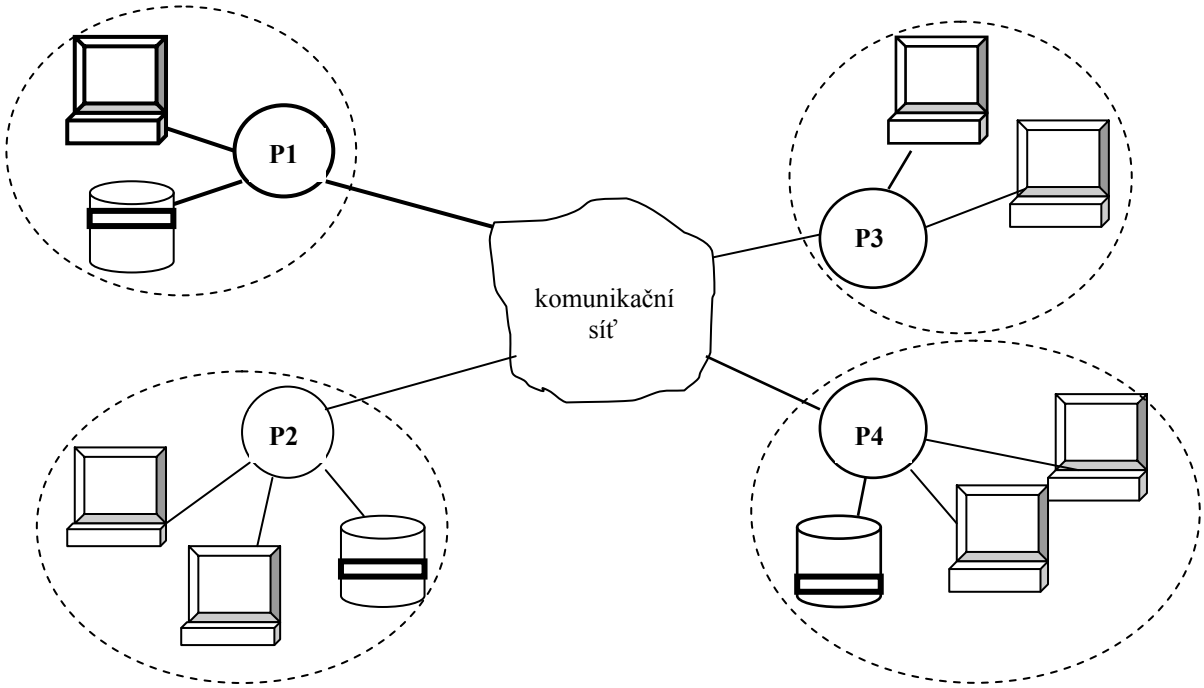
Co není distribuovaná databáze

Z dalších úvah přitom vyloučíme dva triviální případy práce s DBS, které se sice běžně používají, ale nepředstavují žádné podstatné změny proti dosud uváděným lokálním systémům.

Prvním případem je počítačová síť několika počítačů vzájemně propojených, každý může mít lokální síť pracovních stanic. Na každém může být provozována lokální databáze. Z hlediska provozu báze dat není rozdílu v tom, pracuje-li uživatel u terminálu jednoho počítače s lokální bází tohoto počítače, nebo prostřednictvím sítě s lokální bází jiného počítače. Počítačová síť je zde jen prostředkem k připojení vzdáleného terminálu.



Druhým případem je opět počítačová síť několika počítačů vzájemně propojených. Na jednom z nich je provozován SŘBD, jeho data jsou z zčásti uložena na paměťových médiích tohoto počítače, zčásti pak na paměťových médiích jiných počítačů sítě. Z hlediska konceptuálního schématu báze není toto rozdělení báze dat v síti podstatné. Vnitřní organizace báze (interní schéma) obsahuje přístupové cesty k jednotlivým částem báze, takže se jeví jako by byly všechny uloženy jen v paměti uživatelského počítače. Další lokální báze mohou být provozovány na všech dalších počítačích sítě. Počítačová síť je pouze součástí přístupové cesty SŘBD k datům



V obou uvedených případech se nic nemění na práci SŘBD, nedochází k žádné koordinaci práce více SŘBD v síti ani ke sdílení dat. V těchto případech tedy ještě nemluvíme o distribuovaných databázových systémech.

❑ Co je distribuovaná databáze

Pro další výklad si zavedeme následující terminologii:

Definice:

Lokální báze dat je část celé (distribuované) báze, která je umístěna v jednom uzlu (počítači) počítačové sítě; je řízená lokálním SŘBD, který pracuje ve spolupráci s ostatními složkami distribuovaného databázového systému,

Definice:

Distribuovaná báze dat je sjednocení všech lokálních bází dat distribuovaného databázového systému,

Definice:

Distribuovaný databázový systém (DDBS) je tvořen

distribuovanou bází dat a
programovým vybavením, skládajícím se
z lokálních IS provozovaných pod lokálními SŘBD a
dalších programů potřebných k jejich koordinaci a řízení, k zabezpečení celosystémových úloh spojených s přístupem uživatelů k DDBS, s udržováním jeho integrity a provozuschopnosti v prostředí počítačové sítě.

Důležitou vlastností DDBS je to, že přes rozložení dat do jednotlivých uzlů sítě **se jeví celá distribuovaná databáze uživateli tak, jako by byla lokální**.

Distribuce je uživateli skryta a z hlediska jeho potřeb není podstatná. Celá distribuovaná báze dat je popsána v **globálním schématu DDBS**. To je popis báze nezávislý na distribuci dat a stejný pro všechny uživatele ve všech uzlech sítě. Nejčastěji je globální schéma popsáno pomocí relačního modelu dat v tzv. **centrálním datovém slovníku**.

Distribuovaná databáze je skutečně sjednocením lokálních databází v matematickém smyslu slova. V následujících kapitolkách se dozvíme, že z různých důvodů se budou některá data kopírovat do jiných uzlů. Sjednocení takových databází pak zahrnuje každou tabulku či záznam jen jednou, byť jsou uloženy několikrát na různých místech distribuované databáze.

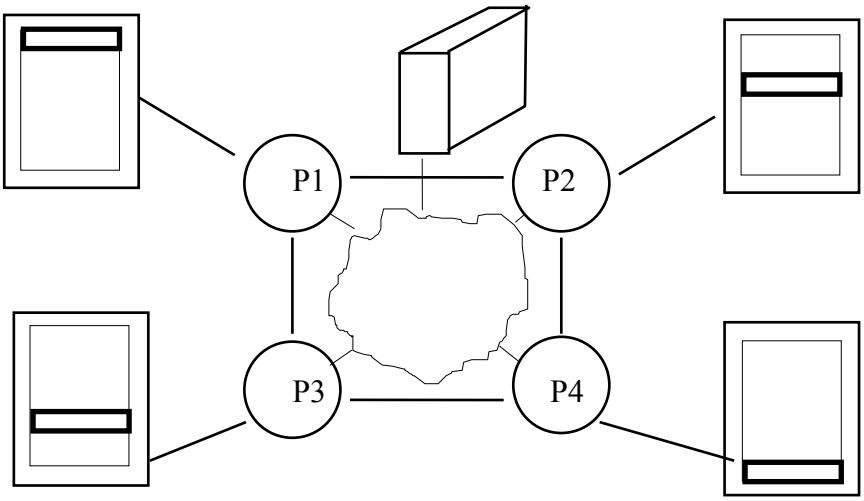
Příklad:

Představme si čtyři pobočky spořitelny v různých městech. Každá má svůj počítač, na něm instalovány relační databáze o zákaznících:

Konto = (pobocka, číslo_konta, zakaznik, castka).

Kromě toho existuje centrální počítač s relační databází a tabulkou

Pobocka = (pobocka, vklady_celkem, mesto).



Další relace, které jsou v databázi, nyní neuvažujeme.

*Dokud bude zákazník ukládat nebo vybírat pouze u pobočky, kde má konto, stačí na to **lokální operace**. Aby však mohl ukládat a vybírat i u jiných poboček, než má konto, musí existovat nějaké propojení všech počítačů. Jedna možnost by byla, že každá pobočka by obsahovala kompletní databázi všech zákazníků všech poboček. To by ovšem mělo tyto nevýhody: souhrnné databáze zabírají zbytečně mnoho místa, redundance by vedla ke složité aktualizaci a k nebezpečí nekonzistence, operace v kterémkoliv uzlu by se musela promítnout do databází ve všech ostatních uzlech. Zřejmě tedy výhodnější možností je distribuovaná databáze, kdy všechny počítače jsou vzájemně propojeny a jsou realizovány **globální operace**.*

♦

6.2. Modely dat lokálních databází

Jednotlivé lokální báze dat mohou být realizovány s použitím jednoho nebo různých datových modelů. Použití jednoho datového modelu nebo jediného SŘBD samozřejmě snižuje složitost architektury

DDBS. Tento případ se užívá u nově vytvářených systémů. Pokud se však dodatečně vytváří distribuovaný systém z řady již existujících IS, které se až dodatečně propojují distribuovaným systémem, může nastat nutnost propojení aplikací v různých SŠBD nebo dokonce různých datových modelů.

Pokud z nějakých důvodů architektura DDBS připouští různé modely dat v jednotlivých lokálních databázových systémech, používá se obvykle jeden společný datový model pro komunikaci a popis hlavních struktur dat v rámci celého distribuovaného systému. Tímto společným datovým modelem je nejčastěji relační model. Pokud lokální SŘBD podporují jiné modely dat, musí být doplněny programovým vybavením, které umí požadované funkce relačního modelu alespoň emulovat. Standardem pro komunikaci lokálních SŘBD bývá většinou jazyk SQL.

6.3. Rozmístění dat v DDBS

U klasických SŘBD jsme za nejslabší místo zpracování (z hlediska času) považovali diskové operace, případně přesuny bloků mezi operační pamětí a diskem. V distribuovaných databázích mají největší časovou náročnost operace přesunu dat po komunikační lince mezi uzly sítě. Tím vznikají některé problémy, které jsme u klasických systémů nepoznali. Pro řešení distribuce dat v DDBS existuje několik optimalizačních technik.

1. Obecně platí, že je **data nutno umisťovat co nejblíže místům jejich vzniku nebo využívání**, pokud tomu nebrání např. parametry počítačů v síti apod.

2. Vzhledem k vyšší možnosti poruch komunikačních linek a uzlových počítačů musí být relace uloženy tak, aby byla zajištěna jejich **dostupnost i při výpadku některých částí sítě**. Relací zde rozumíme logický celek, ale fyzicky může být implementována nejen jako jeden datový soubor.

Používá se dvou hlavních způsobů jejich fyzické implementace:

- **replikace**
- **fragmentace**

❑ Replikace

Replikace spočívá v uchování kopií relací v různých uzlech. Důvodem je to, aby porucha v jednom uzlu neznemožnila přístup k jeho lokálním relacím. Navíc jsou data v lokálních bázích připravena k použití okamžitě, bez nutnosti přenosu.

Příklad:

Firma CDE vlastní zásilkovou síť prodejních skladů o 10 pobočkách, každý sklad se svým IS prodává poněkud jiný sortiment zboží.Zájmem firmy je nabízet nejen zboží své pobočky, ale i kterékoliv sesterské pobočky. Proto každý uzel distribuované sítě má repliky = kopie všech skladů i s cenami a množstvím zboží na skladě. Objednávky na zboží jiné pobočky si denně vzájemně předávají, také aktuální množství nebo změny v sortimentu denně aktualizují.

♦

Urychlení výběru dat však má za následek ztížení aktualizace, protože všechny kopie musí být aktualizovány současně a v průběhu aktualizace je nutno uzamknout aktualizovaná data ve všech uzlech sítě. Proto se v DDBS ukládají v kopiích především ta data, ke kterým je potřebný rychlý přístup a která nejsou často aktualizována, příp. která jsou pro systém mimořádně důležitá. Jsou to např. různé číselníky, registry ap. Obecně tedy stupeň replikace záleží na četnosti změn v databázi. Pro menší četnost změn je lepší použít většího počtu kopií. Replikace zvyšuje výkon při operaci SELECT, ale snižuje výkon pro operace UPDATE a INSERT.

❑ Fragmentace

Fragmentace znamená rozložení relace na části (fragmenty), které jsou umístěny v různých uzlech sítě. Může jít o horizontální fragmentaci, kdy se v různých uzlech ukládají části relace rozložené do skupin řádků, nebo o vertikální fragmentaci, kdy se v uzlech ukládají různé projekce relace. Fragmentace se provádí tak, aby bylo možno z fragmentů získat původní relaci standardními operacemi nad relační databází, např. sjednocením nebo spojením.

Příklad:

Vysoká škola má 10 fakult, většinou umístěných v jiných areálech města. Eviduje mimo jiné své studenty a zaměstnance. Není nutné, aby každá fakulta měla evidovány všechny studenty a zaměstnance školy, celé tabulky Student z Zam jsou horizontálně rozděleny na fragmenty podle fakult, na rektorátě je navíc replika obou celých tabulek.

♦

❑ Repliky fragmentů

Obě metody se často kombinují tak, že se v distribuované bázi uchovávají kopie fragmentů v různých uzlech. V distribuované databázi musí být unikátní jména všech položek, tedy ani dvě lokální relace nesmí používat stejná jména pro různé položky. Jednou z možností, jak tento problém řešit, je použití **centrálního slovníku**. To má však nevýhody v tom, že slovník se stává nejužším článkem systému, protože požadavky na paralelní činnost slovníku mohou být vysoké. Má-li centrální slovník poruchu, havaruje celý systém. Samostatná práce nad lokálními databázemi je velmi omezena.

Jiná možnost spočívá v používání prefixů (předzdívek, předpon, alias) odvozených ze jména uzlového počítače. Tím se zase snižuje nezávislost označení dat na implementaci. Kromě unikátních jmen položek musí mít také každá kopie (replika) a každý fragment své unikátní jméno: UZEL.RELACE.FRAG.REPL

Není možno požadovat od uživatele distribuované databáze, aby sám rozhodoval, kterou kopii relace bude používat. To je úlohou systému a systém se musí také postarat o modifikaci všech kopií, byla-li provedena modifikace relace. Stejná zásada platí pro použití fragmentace. Existence, druh a jména fragmentů jsou vnitřní záležitostí systému. Stejně tak rozhodnutí o tom, zda pro požadovanou operaci je nutné sestavení celé relace, nebo zda lze operaci provést efektivněji nad fragmenty.

K tomu existuje tabulka prefixů, kde jsou uvedeny uživatelské názvy objektů i názvy objektů ve vnitřní reprezentaci systému. K tabulce existuje algoritmus, který určuje fragmenty a kopie. Pro zjištění nákladů na zpracování akce se musí brát v úvahu rozmístění kopií a fragmentů, možnost paralelního zpracování částí akce, délky komunikačních linek mezi uzly, které se na zpracování podílejí a množství informace, kterou je nutno přesouvat.

Příklad:

*Banka s evidencí klientů a kont má horizontálně fragmentována konta podle poboček, kde byly účty založeny. Z bezpečnostních důvodů jsou **fragmenty replikovány**. Protože však všechny repliky musí mít stále aktuální hodnoty, nebudou replikovány do všech ostatních poboček, ale jen do 2 sousedních – aby aktualizace replik necestovala daleko a aby se nemuselo aktualizovat mnoho replik.*

Klient banky má své konto u pobočky P1 a uzlový počítač této pobočky má poruchu. V distribuované bázi může zákazník používat své konto prostřednictvím uzlového počítače jiné pobočky, kde je umístěna replika relací pobočky P1.

♦

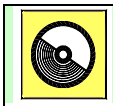
Příklad:

Vypište úplnou tabulku Konta.

Dotaz je sice jednoduchý, ale jeho zpracování v distribuované databázi nemusí být snadné, protože relace Konta je replikována a fragmentována.

Především se musí zjistit, zda jsou kopie fragmentovány. Pokud ne, zvolí se kopie, pro kterou budou nejnižší přenosové náklady. Pokud ano, existuje mnoho strategií, které fragmenty a odkud použít k sestavení kompletní kopie.

♦



Animace

Na CD-ROMu jsou animované příklady na repliky a fragmenty.

- Repliky frag\R01 replikace.exe
- Repliky frag\R02 horizon fragmen.exe
- Repliky frag\R03 vertikal fragment.exe
- Repliky frag\R04 reliky fragmentu.exe
- Repliky frag\R05 navrh rozmisteni.exe
- Repliky frag\R06 dotaz 1.exe
- Repliky frag\R07 aktualizace replik.exe
- Repliky frag\R08 dotaz 2.exe

❑ Optimalizace distribuovaných operací

Ukažme si dále strategie, které můžeme v distribuované databázi použít pro zpracování dotazu obsahujícího operaci spojení.

Příklad:

Máme tři relace R1, R2, R3, které nejsou ani replikovány, ani fragmentovány. Každá z relací je umístěna v jiném uzlu U1, U2, U3 a v dalším uzlu U4 položíme dotaz, který vyžaduje provést operaci spojení R1 + R2 + R3.

Můžeme použít např. následující strategie:

1. *Kopie všech tří relací přemístíme do uzlu U4 a zpracujeme dotaz v U4.*
2. *Kopii R1 pošleme do R2 a v U2 zpracujeme spojení R1 + R2. Výsledek pošleme do R3 a v U3 zpracujeme (R1 + R2) + R3. Výsledek potom pošleme do U4.*
3. *Kopie zpracováváme podobně jako při strategii 2, ale použijeme jiné pořadí.*

Nebo pro 4 fragmenty je možno spustit paralelně spojení (R1 + R2) v uzlu U1, (R3 + R4) v uzlu U3 a pak výsledek spojit do uzlu U5.

♦

Obecně není možno rozhodnout, která strategie je lepší. K tomu je třeba vzít v úvahu

- množství dat, které je třeba přesunout,
- cenu za přenos bloku dat mezi jednotlivými uzly,
- rychlost zpracování v jednotlivých uzlech.

a vhodným optimalizačním algoritmem zvolit nejlepší strategii.

6.4. Těsnost propojení replik lokálních databází

V klasických SŘBD se přirozeně požaduje, aby se aktualizační akce provedly okamžitě a aktualizované hodnoty byly ihned nato přístupné všem uživatelům databáze. V distribuovaných systémech se tento přirozený požadavek zabezpečuje podstatně hůř a je příčinou složitosti programového řešení DDBS a tím i vyšších nákladů na jeho provoz. Podle konkrétních aplikačních úloh však není nutné přísně dodržovat tyto požadavky vždy a tak je možno někdy celý systém zjednodušit a zlevnit.

Samozřejmě se se zvoleným řešením aktualizace musí počítat již při celkovém návrhu informačního systému, v etapě návrhu implementace. Aby byla zajištěna aktuálnost všech dat, nebo aby z podstaty úlohy bylo známo, že data nejsou aktuální, nebo aby dovoloval počítat s dočasně nepřesnými údaji. Poslední DDBS nazýváme také volně spojené systémy.

Návrh rozmístění dat v distribuované databázi je tedy dalším úkolem etapy návrhu implementace, pokud budovaný IS bude distribuovaným.

Příklad:

Prodejní a skladová síť velkého výrobního podniku je ve svých pobočkách vybavena lokální počítačovou sítí, každá pobočka vede evidenci o stavu a pohybu výrobků na svém skladě. Tyto lokální databáze jsou součástí celopodnikové distribuované databáze, ke které je přístup ze všech poboček.

Často používaná data o stavu výrobků na skladě jsou v ní rozmístěna v kopiích v každé lokální databázi. Aby se zjednodušilo řízení a provoz tohoto DDBS, během dne se aktualizační operace jen zaznamenávají a provedou se najednou ***denně*** *ve večerních hodinách, kdy počítače nejsou vytížené a volnější jsou i komunikační linky. Tak má každá pobočka denně čerstvé informace o stavu výrobků v celém podniku, není nutné znát průběžný okamžitý stav.*

Tento podnik má společný ceník prací a výrobků, který je občas, nepravidelně měněn v centrálním uzlu podniku. Ceník je v kopiích umístěn na všech pobočkách, nový je přenášen vždy ***po změně*** *do všech ostatních poboček.*

♦

Příklad:

Velká zásilková firma má pobočky po Evropě, každá pobočka má svůj informační systém s databází. Společně používají pouze SW pro výpočet mezd a prémií za výkony. Pracovníci mohou vykazovat výkony nebo pracovat nejen pro svou pobočku, ale i pro ostatní pobočky, tam jsou jejich výkony evidovány. Ale na své „mateřské“ pobočce o těchto aktivitách musí být také evidence. Mzdy se vyplácejí jednou měsíčně.

Každá pobočka eviduje pracovní docházku, dovolené, nemocenské i pracovní výkony apod. pro své zaměstnance v průběhu celého měsíce. ***Měsíčně****, vždy k 1. následujícího měsíce, se přenášejí údaje o pracovní době a výkonech zaměstnanců mezi všemi pobočkami, v jednom centrálním uzlu se pak počítá mzda a vyplácí centrálně na účty zaměstnanců. Pro mzdy není nutné, aby informaci o docházce byla známa ostatními pobočkami dříve.*

♦

Příklad:

Banka má mnoho poboček po republice, každá pobočka má své klienty a ti mají svá konta. Denně se provádí velmi mnoho transakcí na každé pobočce, ale klienti mohou své požadavky vyřizovat na kterékoliv pobočce.

*Zde je zřejmě nutné znát **okamžitý stav** každého účtu v kterémkoliv okamžiku, aby podnikavý klient nemohl „současně“ vybrat na řadě poboček svůj jediný vklad mnohokrát (jak tomu bylo v době počátků bankomatů, které nebyly propojeny s databází a údaje se přenášely denně).*

Jednou možností řešení je jediná centrální databáze se všemi účty, ale ta by byla zřejmě přetížena. Kopie všech účtů na všech pobočkách by zase musely být neustále všechny pořád aktualizovány, což by vedlo k přetížení přenosů dat.

*Vhodné tedy bude například řešení, kdy každá pobočka eviduje své klienty i jejich účty – tedy fragmenty celých relací – protože tam se předpokládají nejčastější změny. V nejbližších 2 uzlech budou bezpečnostní repliky těchto fragmentů pro případ výpadku nebo přetížení mateřského uzlu. Aktualizace účtů se musí provádět **okamžitě,** aktualizace klientů případně stačí **denně** nebo 2 x denně.*

♦

Příklad:

*Informační systém **ABC** zdravotnického střediska s několika lékaři eviduje lékaře, pacienty, objednané pacienty a uskutečněné návštěvy u lékaře i lékařů u pacientů (datum a čas objednané nebo realizované návštěvy, diagnózu, předepsané léky, provedené výkony pacientovi, cena výkonu pro pojišťovnu). Při jedné návštěvě se eviduje jediná hlavní diagnóza. Na konci měsíce se výkony provedené pacientům vyúčtují pojišťovně, to se vyznačí v vyuct = ano. Po zaplacení pojišťovnou se zaplacená návštěva vyznačí v zaplac = ano. Ve stejnou dobu je vždy objednán nebo ošetřen jediný pacient, jeden pacient však může být objednán na stejnou dobu k více lékařům. Databáze ABC má strukturu:*

Lekar (RC_L, jmeno_L, specializace)
Pacient (RC_P, jmeno_P, pojistovna)
Navsteva (id_navst, RC_L, RC_P, datum, hodina, diagnoza, vyuct, zaplac)
Leky_pac (id_navst, id_lek, mnozstvi)
Vykony_pac (id_navst, id_vykon)
Cisel_vykonu (id_vykon, nazev_vyk, cena_vyk)
Cisel_leku (id_lek, nazev_lek, cena_lek)

Najděte vhodná rozmístění zadaných relací a frekvence aktualizací replik v distribuované databázi IS ABC, má-li zdravotní středisko 10 poboček ve městě. Každé středisko má vlastní IS. Protože každá pobočka nemá stejné vybavení, pacienti jsou občas posíláni k lékařům na specializovaná vyšetření do jiných poboček – pak jde o samostatnou návštěvu.

Řešení:

Cisel_leku *se nemění po dlouhou dobu, bude proto replikován do všech poboček. Aktualizace bude provedena do všech uzlů při každé změně.*

Cisel_vykonu - *důvody i řešení stejné jako u číselníku léků.*

Pacient *se sice mění, ale poměrně málo, navíc změnu nemusí nutně znát okamžitě ostatní uzly. Bude buď replikován do všech poboček s denní nebo řidší aktualizací, nebo bude fragmentován podle mateřských poboček, fragmenty replikovány do nejbližších uzlů z bezpečnostních důvodů jako zálohy, aktualizace denně.*

Lekar *se mění ještě méně, než pacient. Bude replikován do všech poboček, aktualizace denně nebo méně často, například začátkem měsíce po případných změnách. Vhodnější proto bude řešení, že* Lekar *bude replikován do všech poboček, aktualizace při každé (málo časté) změně.*

Navsteva *se aktualizuje i přibývá denně relativně hodně a jde o jednu ze tří nejdůležitějších evidencí. Bude fragmentována podle mateřských poboček, tam se bude denně doplňovat a měnit, fragmenty budou replikovány do 2 nejbližších uzlů z důvodů bezpečnostní zálohy. Aktualizace denně, v případě zničení databáze by chyběly jen záznamy posledního dne.*

Vykony_pac *jsou asi nejdůležitější relací pro vyúčtování zdravotní pojišťovně, podobně* Leky_pac. *Řešení bude jako u návštěv s případnou častější (např. 2 x denně v polední přestávce) aktualizací replik v sousedních uzlech.*

♦

6.5. Stupeň centralizace řízení transakcí

Toto hledisko je v architektuře DDBS jedno z nejpodstatnějších. Centralizace řízení DDBS vypovídá o tom, nakolik je řízení systému koncentrované na jedno místo. Dva krajní případy jsou plně centralizované a plně decentralizované distribuované systémy.

❑ Centralizované řízení DDBS

Popis databáze (centrální datový slovník) i řízení DDBS soustředěno na jeden centrální počítač. Toto centrum DDBS nemusí být v centru příslušné počítačové sítě. V centru DDBS jsou **soustředěny popisy všech dat** tvořících DDBS a centrálně se tu řídí

- přístup k distribuované bázi dat,
- provádění změn ve struktuře distribuované báze dat,
- provádění a synchronizace transakcí v DDBS,
- všechny další činnosti systému.

Výhodou centralizovaných DDBS je poměrná jednoduchost řízení všech činností systému. Řídicí SW má soustavný přehled o aktuálním stavu všech částí systému a má možnost v přesně a jednoznačně zasahovat.

Nevýhodou těchto DDBS jsou vysoké celkové nároky na komunikaci v systému. Každá transakce, každý přístup k datům, každá změna musí být povoleny a řízeny centrem. Tato skutečnost může značně zpomalit a prodražit provoz DDBS. Jen v lokálních počítačových sítích s vysokými přenosovými rychlostmi se tato nevýhoda nemusí projevit tak výrazně.

Dalším nebezpečím je možnost poruchy počítače s centrálním řízením databáze, protože při výpadku hrozí pracná obnova celého DDBS.

Příklad:

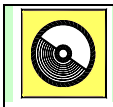
Firma má několik poboček v distribuovaném IS s centrálním řízením, centrální řídicí uzel je v Ostravě, ostatní pobočky v Brně, Zlíně a Olomouci. Eviduje mimo jiné své zaměstnance (fragmentovány dle poboček s replikou celé relace do centrálního uzlu, aktualizace denně v noci).

Ve Zlíně potřebují zvýšit plat svému zaměstnanci panu Novákovi.

Protože datový slovník a řízení distribuovaného zpracování je v Ostravě, tento požadavek (UPDATE) je zaslán do centra, tam řídicí SW spustí transakci – pošle příkaz zpět do Zlína, tam se transakce provede a zpráva o ukončení transakce se pošle centru do Ostravy.

Má-li ostravský uzel poruchu, vypadne celý distribuovaný systém, i když ostatní pobočky jsou v pořádku.

♦



Animace

Na CD-ROMu jsou animované příklady na provádění operací při centralizovaném řízení.

- Central rizeni\C01 modif nereplik tabulky.exe
- Central rizeni\C02 modif replik tabulky.exe
- Central rizeni\C03 modif vada uzlu.exe
- Central rizeni\C04 dotaz vertik fragment.exe
- Central rizeni\C05 dotaz vertik fragment 2.exe
- Central rizeni\C06 dotaz vertik fragment 3.exe
- Central rizeni\C07 dotaz vertik fragment 4.exe
- Central rizeni\C07 prace v lokal uzlu.exe
- Central rizeni\C08 prace v cizim uzlu.exe
- Central rizeni\C10 skladani fragmentu.exe

❑ Decentralizované řízení DDBS

Decentralizované DDBS jsou tvořeny počítačovou sítí, kde žádný uzel nemá privilegované postavení. Všechny počítače mají stejné informace o DDBS a stejnou zodpovědnost za dodržování pravidel vedoucích k zachování integrity systému. Je zřejmé, že algoritmy pro řízení transakcí v takovémto distribuovaném prostředí, bez centrálního řízení, budou složitější. Decentralizované systémy však proti centralizovaným vynikají svou stabilitou. Výpadek žádného počítače nemá za následek větší ztrátu, než ztrátu přístupu k vlastním datům. Tu je navíc možno duplikováním kritických dat v několika uzlech sítě podle potřeby zmírňovat.

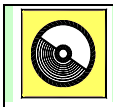
Příklad:

Firma má několik poboček v distribuovaném IS s plně decentralizovaným řízením, každý uzel má vlastní řízení, pobočky jsou v Ostravě, Brně, Zlíně a Olomouci. Eviduje mimo jiné své zaměstnance (fragmentovány dle poboček s replikou celé relace do centrálního uzlu, aktualizace denně v noci).

Ve Zlíně potřebují zvýšit plat svému zaměstnanci panu Novákovi.Transakce se spustí a provede ve Zlíně. Má-li ostravský uzel poruchu, vypadne jen Ostrava, ostatní pobočky pracují dál. Pokud potřebují pracovat s daty ostravskými, musí se počkat na opravu tohoto uzlu.

Má-li poruchu databáze zlínská, může se transakce provést na replice v Ostravě a po opravě Zlína se aktualizuje i zlínská databáze.

♦



Animace

Na CD-ROMu jsou animované příklady na provádění operací při decentralizovaném řízení.

- Decentral rizeni\DC1 dotaz 1.exe
- Decentral rizeni\DC2 modif 1.exe
- Decentral rizeni\DC3 dotaz 2 spojeni.exe
- Decentral rizeni\DC4 modif 2.exe
- Decentral rizeni\DC5 insert 1.exe
- Decentral rizeni\DC6 delete 1.exe
- Decentral rizeni\DC7 modif 3.exe
- Decentral rizeni\DC8 dotaz 3 sum.exe

❑ Kombinované řízení DDBS

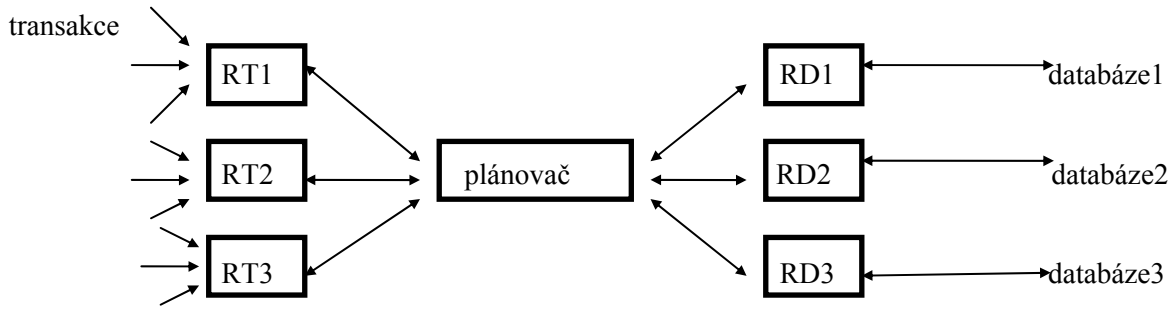
Optimální většinou je **kombinovaná architektura řízení DDBS**. Některé uzly jsou řídicí, jiné jsou řízeny nejbližšími řídicími uzly. Tato architektura je sice nejstabilnější, ale také nejnáročnější na vývoj i provoz řídicího SW.

6.6. Paralelní zpracování v distribuovaných databázích

Zajišťování atomického charakteru transakcí v distribuované bázi je řádově složitější, než u lokálních sítí, protože možnosti poruch jsou mnohem složitější.

❑ Plánovač v centralizovaném řízení DDBS

V centralizovaných DDBS řídí pořadí vykonávání operací jediný plánovač umístěný v centru DDBS. Řízení paralelních transakcí v centralizovaném systému znázorníme takto:



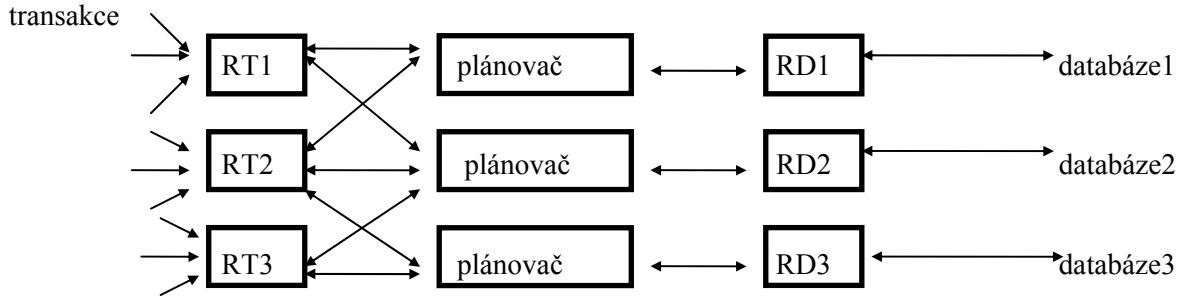
Protože plánovač má kontrolu nad všemi daty v DDBS, mohou se bez problémů použít typy plánovačů pro klasické SŘBD. Připomeňme, že každá transakce může číst a měnit hodnoty objektů v libovolných lokálních bázích dat. Proto případné zrušení transakce je spojeno s většími problémy. Zrušená transakce musí vycouvat, tj. vrátit původní hodnoty objektům, které již změnila. V distribuované bázi to může být zdlouhavá a nákladná operace, proto budou výhodnější takové plánovače, které pracují bez rušení transakcí.

U plánovačů pracujících s časovými razítky ČR je nutno sladit lokální hodiny - tak, aby časová razítka byla navzájem různá, i když pocházejí z různých uzlů.

❑ Plánovač v decentralizovaném řízení DDBS

V případě decentralizovaných DDBS jsou plánovače v každém lokálním SŘBD. Řízení paralelních transakcí v decentralizovaném systému je výrazně složitější, protože musí i na jedné transakci spolupracovat různé plánovače.

I v decentralizovaných DDBS je možno použít typy plánovačů z klasických SŘBD. Při zamykání objektů zamyká a odemyká každý lokální plánovač objekty ve své bázi. Problémem je ale kontrola korektního dodržování dvoufázového protokolu zamykání. Řešením může být pozdržení odemknutí všech objektů zamknutých pro transakci, dokud transakce neskončí a pak vyslat všem plánovačům zprávu o skončení transakce. Pák je možno objekty uvolnit.



V decentralizovaném systému se těžko zjišťuje uváznutí, to nemůže detekovat žádný lokální plánovač. Proto je výhodné uváznutím předcházet, např. použitím plánovačů pracujících s ČR, kde k uváznutí nedochází.

6.7. Shrnutí

Celkově můžeme shrnout, že distribuovanost databází zvyšuje jejich složitost, náchylnost k chybám i cenu celého systému. Výrazně se zvyšuje podíl režie systému na celkovém zpracování. Řadu problémů zpracování dat ovšem nelze řešit efektivně jiným způsobem. Základ tvoří řada pracovišť, která mohou pracovat s lokálními databázemi, tj. mohou provádět lokální transakce. K tomu dále patří možnost provádět také globální transakce s daty umístěnými v jiných lokálních databázích. Provádění globálních transakcí předpokládá existenci technické a programové podpory komunikace mezi jednotlivými lokálními pracovišti. Důležitým požadavkem je, aby se problémy týkající se programového vybavení nedotýkaly uživatele a jeho způsobu práce s distribuovanou databází. Aby byl distribuovaný systém zabezpečen i po stránce technického vybavení, musí mít zabudován systém detekce chyb a možnost rekonfigurace pro případ poruchy některého ze stanovišť nebo některého spojovacího vedení.

Shrnutí pojmů 6.

Lokální databáze, distribuovaná databáze, centrální datový slovník.
Lokální IS, distribuovaný IS.
Lokální operace, globální operace.
Datové modely lokálních bází a jejich propojení.
Metody rozmístění dat v distribuované databázi, replikace, fragmentace.
Těsnost propojení replik.
Stupně centralizace řízení distribuovaných transakcí.
Paralelní provoz v distribuovaných IS, centrální a decentralizované řízení transakcí.

Otázky 6.

1. Definujte lokální a distribuovaný IS.
2. Definujte lokální a distribuovanou databázi.
3. Které metody se používají pro rozmístění dat v distribuované databázi?

4. Podle jakých pravidel se navrhuje rozmístění dat v distribuované databázi?
5. Podle jakých pravidel se navrhuje aktualizace replik v distribuované databázi?
6. Jak jsou řízeny transakce v distribuované databázi s centrálním řízením?
7. Jak jsou řízeny transakce v distribuované databázi s decentralizovaným řízením?

Úlohy k řešení 6.

1. Veřejná knihovna KOMEN má databázi se strukturou:

   Titul (ISBN, nazev, autor, obor, vydavatel, rok_vydani, cena)
   Exempl (prir, ISBN, stav)
   Vypujcky (prir, id_cten, dat_vypuj, ter_vraceni, dat_vraceni, upomin)
   Ctenar (id_cten, jmeno, adresa, telefon, e-mail)

   kde prir je přírustkové číslo, jednoznačný klíč každého exempláře, datum výpůjčky i předepsaný termín vrácení se zapisuje hned při vypůjčení, dat_vraceni je skutečné datum vrácení, upomin je logická hodnota 0 = neupomínáno nebo 1 = upomínáno, stav má hodnoty 0 = nevypůjčeno, 1 = vypůjčeno, -1 = ztraceno, -2 = zničeno.

   Najděte vhodná rozmístění dat a frekvence aktualizací replik v distribuované databázi IS KOMEN, má-li knihovna 10 poboček v okrese, každá pobočka má vlastní IS, čtenáři si mohou půjčovat i vracet knihy kdekoliv v síti.

2. Je dána databáze LETIŠTĚ, evidující letiště, letecké společnosti, letadla, letové plány (obdoba jízdního řádu) a letenky vydané ke konkrétním letům v letovém plánu.

   Let_spol (**id_spol**, nazev_spol, adresa, telefon, email)
   Letadlo (**cis_letadla**, oznac_letadla, typ_letadla, kapacita, dolet, *id_spol*)
   Letiste (**id_letiste**, nazev_letiste, mesto, ulice, psc, zeme, telefon, fax, email, pocet_drah)
   Letovy_plan (**id_planu**, *id_letiste*, *id_letiste_cil*, *cis_letadla*, datum_odletu, cas_odletu, delka_letu, skut_pocet_osob)
   Letenka (**id_letenky**, *id_planu*, jmeno, prijmeni,telefon)

   Najděte vhodná rozmístění dat a frekvence aktualizací replik v distribuované databázi IS Letiště, má-li společnost provozující IS 30 poboček – letišť v Evropě, každá pobočka má vlastní IS.

LITERATURA

1. DATE, C. J.: *An Introduction to Datebase Systems.* Addison-Wesley Publishing Company, USA, 1990.
2. DATASEM'92 - DATASEM'99. *Sborníky seminářů*. CS-Compex, Brno, 1992-1999.
3. DATAKON'2000 - DATAKON'2006. *Sborníky seminářů*. Brno, 2000 – 2006
4. KROHA P.: *Báze dat*. FE ČVUT, Praha, 1990.
5. LUKASOVÁ, A. *Úvod do databázové technologie*. Ostrava: Ostravská Univerzita 1993
6. MODERNÍ DATABÁZE. *Sborníky seminářů*, Dům techniky, Ústí nad Labem, 1995-2006.
7. POKORNÝ, J.: *Databázové systémy a jejich použití v informačních systémech.* Academia, Praha, 1992.
8. POKORNÝ, J.: *Databázová abeceda*. Science, Praha, 1998.
9. POKORNÝ, J.: *Dotazovací jazyky*. Science, Praha, 1994.
10. POKORNÝ, J.: *Počítačové databáze*. Kancelářské stroje, Praha, 1991.
11. POKORNÝ, J.: *Učíme se SQL*. PLUS, Praha, 1993.
12. POKORNÝ, J. - HALAŠKA, I.: *Databázové systémy. Vybrané kapitoly a cvičení*. FE ČVUT, Praha, 1998.
13. RICHTA, K. - SOCHOR, J.: *Softwarové inženýrství I*. FE ČVUT, Praha, 1996.
14. ŠARMANOVÁ, J. *Teorie zpracování dat.* Ostrava: VŠB-TU 2007
15. TIETZE, P.: *Strukturální analýza, úvod do projektu řízení*. Grada, Praha, 1992.
16. TSICHRITZIS, D.C. - LOCHOVSKY, F.H.: *Databázové systémy*. SNTL, Praha, 1987.