Vysoká škola báňská – Technická univerzita Ostrava

INFORMAČNÍ SYSTÉMY A DATOVÉ SKLADY

Učební text

Jana Šarmanová

Ostrava 2007

Recenzoval: prof. RNDr. Alena Lukasová, CSc.

Název: Informační systémy a datové sklady
Autor: Jana Šarmanová
Vydání: první, 2007
Počet stran: 169

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 978-80-248-1500-8

POKYNY KE STUDIU

Informační systémy a datové sklady

Prerekvizity

Předmět je určen pro studenty oboru Informatika, Matematika nebo oboru příbuzného. Předpokládají se základní znalosti z databází a informačních systémů v rozsahu předmětů Bc. studia Úvod do SW inženýrství, Teorie zpracování dat (TZD) a Databázové a informační systémy (DAIS).

Cíl modulu

Předmět je určen databázistům a budoucím analytikům v oblasti analýz dat pomocí datových skladů. První část učebnice opakuje a prohlubuje znalosti o budování informačních systémů, zvláště rozsáhlých systémů budovaných na zakázku. Ve druhé části se seznámíte s důvody vzniku a základními principy datových skladů jako speciálních druhů databází. Třetí doplňující část podává základní informace o textových databázích a práci s nimi.

Na tento předmět navazuje předmět Metody analýzy dat, v němž jsou probírány metody získávání znalostí z dat, databází a datových skladů.

Průvodce studiem

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.

Při studiu každé kapitoly doporučujeme následující postup:

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 datových skladů ještě nikdy nesetkali a naopak takoví, kteří již v tomto oboru mají bohaté zkušenosti.

Cíl Po prostudování tohoto odstavce budete umět

- popsat …
- definovat …
- 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.

Protože tento předmět navazuje na TZD a DAIS, jsou některé odstavce zopakovány, ale ve stručnější formě. Podrobněji je najdete v příslušných učebnicích. Tyto odstavce jsou nadepsány tmavě červenou barvou, ostatní modrou.

Shrnutí pojmů 1.1.

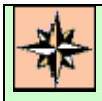
Na závěr kapitoly jsou zopakovány hlavní pojmy, které si v ní máte osvojit. Pokud některému z nich ještě nerozumíte, vraťte se k nim ještě jednou.

Otázky 1.1.

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

Úlohy k řešení 1.1.

Protože většina teoretických pojmů tohoto předmětu má bezprostřední význam a využití v databázové praxi, 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.



Průvodce studiem

Občas v tomto rámečku budou průvodní informace například o tom, co je důležité umět, co stačí jen přečíst informativně, na co látka navazuje nebo co bude navazovat na ni apod.

Úspěšné a příjemné studium s touto učebnicí Vám přeje autorka kurzu

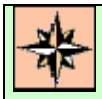
Jana Šarmanová

OBSAH

A. INFORMAČNÍ SYSTÉMY

B. DATOVÉ SKLADY

1. ROZSÁHLÉ INFORMAČNÍ SYSTÉMY



Průvodce studiem

V předmětech Teorie zpracování dat a Databázové a informační systémy jsme se naučili, jakým postupem se vytvoří menší informační systém. Ze SW inženýrství známe životní cyklus vývoje SW.

Většina současných IS se vyvíjí na zakázku cizího zadavatele a většina těchto IS jsou rozsáhlé systémy, na kterých již nemůže pracovat jediný řešitel. Proto si musíme již získané znalosti a zkušenosti zopakovat a doplnit o další informace.

Čas ke studiu kapitoly: 2 hodiny

Cíl Po prostudování této kapitoly

- si zopakujete, co je systém obecně a v této souvislosti co je informační systém a jeho životní cyklus,
- seznámíte se s tím, jak se vývoje informačního systému účastní tým spolupracovníků, jaké jsou SW profese, jaké jsou typy týmů,
- se dále seznámíte s pojmem SW fyzika a s tím, co z jejích závěrů vyplývá,
- budete umět popsat rozdíl mezi různými typy informačních systémů.

Výklad

1.1. Informační systémy

Systém a informační systém

Úvodem si zopakujeme několik základních pojmů, známých z předmětů SW inženýrství, Databázové a informační systémy, případně dalších.

Definice:

> **Systémem** nazýváme seskupení prvků ucelené oblasti reality, doplněné o vazby mezi nimi, o jejich uspořádanost, jejich strukturu a hierarchii.

Tento celek – objekt našeho zájmu je provázán s okolním světem, tedy se svým okolím vazbami - vstupy a výstupy systému. Systém je podobný pojem jako organizace nebo struktura. Systém může být sám o sobě rozložitelný na podskupiny, které mají z našeho hlediska zkoumání významnou funkci a mezi kterými existují opět podstatné a definovatelné vazby.

Každou z částí systému je dále možno označit jako další objekt zkoumání, který se všemi jeho vstupy a výstupy můžeme chápat jako samostatný systém. Takovéto objekty budeme nazývat **subsystémy** původního systému.

Zřejmě každý zkoumaný objekt, který má vlastnosti systému, je zároveň subsystémem jiného systému.

Z hlediska existence systémů v závislosti na člověku můžeme rozdělit **systémy** na **přirozené** (*přírodní objekty od buňky po vesmír, systémy fyzikální, živočišné apod.*) a **umělé**, vytvořené člověkem (*telefonní síť, systém škol, systém zákonů, dětská stavebnice atd.*)

Jeden z typů umělých systémů, který nás v tomto předmětu zajímá především, je **informační systém** (IS). I toto pojmenování se používá v mnoha významech, nejen pro počítačem podporované IS. Uvedeme si tedy 2 definice, jednu pro obecný jakýkoliv IS *(třeba na informační nástěnce)*, druhou pro počítačový = automatizovaný IS.

Definice:

> **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 rozhodování v běžné praxi.

Definice:

> **Informačním systémem automatizovaným** (realizovaným na počítači) rozumíme programový 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.

Nadále se budeme zabývat pouze IS automatizovanými, pokud nebude výslovně uvedeno jinak.

❑ Tvorba umělých systémů

Aby výsledek tvorby jakéhokoliv většího umělého systému byl nejen úspěšný, ale také hotov v optimálním čase a za efektivní cenu, není možné postupovat nesystematicky, náhodně řešit každý případ. Již před staletími se lidé naučili postupovat systematicky, optimálně rozplánovat postup prací a opakovaně takové vhodné postupy používat. U většiny postupů můžeme vysledovat velmi podobný průběh prací, podobné rozložení do časových a tvůrčích etap.

Nejprve se člověk učil plánovat a vyrábět různé hmotné systémy *(stavby, stroje, komunikační sítě, atd.)*. Podobně se v posledních desetiletích učí zpracovávat i informace. Praxe ukázala, že mezi postupem při tvorbě hmotných a nehmotných systémů není podstatný rozdíl. Souhrn metodologických prostředků používaných při zkoumání a popisu existujících či plánovaných systémů 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.

Základní výsledek systémového přístupu řešení každého většího díla je dodržování etap:

1. formulovat zadání, popsat požadavky na výsledné dílo,
2. analyzovat věcné požadavky detailně, vytvořit modely výsledného díla,
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.**

Z těchto zkušeností také pochází rozložení prací na vývoji informačních systémů, nazývaný někdy životním cyklem automatizovaného informačního systému, či obecněji **životním cyklem programového díla (ŽC).** Ovšem současná znalost vhodných metod řešení software se vyvíjela postupně, nevznikla automaticky se vznikem počítačů.

Historie tvorby programových systémů

Lidé používají algoritmy - návody k tomu, jak něco udělat – odpradávna. Se vnikem počítačů se algoritmy (zprvu výpočetní, odtud dodnes „počítač“) začaly zapisovat do vhodného počítačového jazyka, který měl zadánu množinu povolených příkazů. Přesto, že v tu dobu již lidé uměli vytvářet velká díla hmotná a znali k jejich tvorbě dobré postupy, nepřenesli automaticky tyto zásady do sestavování programů. Trvalo to téměř 20 let, než došlo k prvním formulacím pravidel, jak programovat. Jak rostla kapacita a rychlost počítačů a tím i možnost vytvářet velké programy, začali programátoři pociťovat nutnost zavést jakýsi pořádek do psaní programů. Programovací jazyky připouštějící chaotické (nestrukturované) programování přestávaly být spolehlivě odladitelné.

Poprvé byl termín „SW inženýrství“ použit koncem 60. let 20. století.

Jedny z prvních pravidel definovaných v letech 1970-73 byly principy strukturovaného programování. Protože se z nich stal samozřejmý standard, vyučovaný v základech algoritmizace a využívaný ve všech následujících programovacích jazycích, zopakujeme si jeho zásady. Poprvé byly definovány základní programové struktury a dokázána jejich úplnost (je možné pomocí nich zapsat jakýkoliv algoritmus). Navíc byly definovány další zásady, například řešení velkých úloh rozkladem metodou shora-dolů a další.

V dalších letech se rozvíjely metody zajišťující spolehlivost a kvalitu programů vhodným testováním, studují se možnosti automatické verifikace správnosti programů. Také se poprvé provádí analýzy vztahu mezi cenou a mohutností programů.

Koncem 70. let 20. století se začínají zdůrazňovat etapy předcházející vlastnímu programování – modulární dekompozici, modelování. V letech 1978-80 vznikají první nástroje pro podporu SW inženýrství. V 80. letech pak vznikají ucelené systémy CASE (Computer Aided Software Engineering) podporující několik etap ŽC. Obdobné systémy se používají běžně dodnes, ovšem už nebývají spojeny s tehdejší naivní představou, že samy automaticky vyřeší všechny úlohy projektu.

Celý životní cyklus vývoje SW, jak ho známe dnes, tedy vznikal postupně.

Zásady strukturovaného programování

1. **Princip abstrakce**: člověk je schopen jasně a přesně uvažovat jen o nepříliš složitých objektech. Je-li postaven před složitý úkol, musí nejprve zanedbat některé jeho aspekty a doufat, že je zvládne později. Často si v tomto procesu pomáhá tím, že zvolené podúkoly a podobjekty nejprve pouze pojmenuje a detailněji se jimi zabývá až v dalších etapách.
2. **Zásada programování shora-dolů**: opakovaná aplikace principu abstrakce. Zkušenosti doporučují využít i psychologické stránky, že každý segment se zapíše na jednu fyzickou stránku (popisu či programu), kde jsou proloženy příkazy programovacího jazyka názvy funkčních specifikací dosud nevyjádřených ve formě algoritmu.
3. **Zásada jednoduchosti**: pro maximální zjednodušení popisu reálných komplikovaných procesů je nutné se vyvarovat všech "chytrých" konstrukcí a speciálních triků, které nevyplývají z podstaty problému a jsou všem mimo autora (a později i jemu) nesrozumitelné. Doporučuje se používat jen základní řídicí programové struktury, které dostačují k zápisu kteréhokoli programu: sekvence, větvení a cykly.

4. **Zásada přiměřené organizace**: organizace týmové práce se doporučuje jako metoda organizace práce, mající rozhodující vliv na uspokojivý průběh práce na projektu.

   Tým sestává z nevelkého počtu (<10) kvalifikovaných programátorů a je veden hlavním programátorem. Na rozdíl od řady jiných organizací práce hlavní programátor (dnes nazývaný manažerem) celý tým nejen vede, ale skutečně se podílí na řešení, a to právě u klíčových segmentů programového systému. Zásada "shora dolů" umožňuje přesně vymezit podúkoly dalším členům týmu. Záložní ideový informatik je druhá osoba, která dokonale ovládá celou logiku systému; spolupodílí se na strategii výstavby systému, dělá oponenta idejím manažera, může kdykoliv převzít vedení. Důležitá je osoba sekretáře, kterým je však nikoliv administrativní pracovník, ale kvalifikovaný informatik plně obeznámený s problematikou projektu. Jeho úkolem je evidovat a distribuovat všechny dokumenty, ať už psané v řeči stroje nebo lidí. Shromažďuje od řešitelů veškerou dokumentaci, kontroluje přípravu dat, odebírá výsledky a archivuje je (včetně chybných pokusů). Celý tým komunikuje pomocí archivu, v němž jsou uloženy všechny programy a jejich výsledky i komentáře o stavu projektu. Archiv je kdykoliv přístupný kterémukoliv členu týmu.

❑ Životní cyklus vývoje informačního systému

Již víme, že každé větší dílo je nutné tvořit vhodným postupem. O tvorbě SW to víme také a známe několik typů nejčastějších životních cyklů pro vývoj informačního systému. Liší se obvykle jen různou mírou podrobnosti, přeléváním některých činností z jedné etapy do druhé nebo tím, zda se buduje celý systém najednou či po částech. Nejčastější typy se nazývají obvykle vodopádový a iterační (spirálový). Každý cyklus obsahuje následující etapy:

1. zadání či specifikace požadavků, funkčních a nefunkčních,
2. analýza či specifikace systému, modelování systému z několika hledisek,
3. návrh implementace, upřesnění detailů budoucí pro budoucí programování,
4. implementace, programování a psaní dokumentace,
5. testování správnosti systému,
6. zavedení systému do provozu,
7. provoz systému, jeho údržba a rozvoj.

❑ Programy a programové systémy

Mezi programováním klasických algoritmů „školních“, technických výpočtů apod. nebo i velkých SW systémů jako jsou operační systémy, SŘBD (kde zadání dělá sám řešitel) na straně jedné a programy vznikajícími na zakázku dle požadavků reálných i potencionálních uživatelů na straně druhé, je podstatný rozdíl. V prvém případě je programátor pánem sám ve svém vlastním světě, sám si úlohy formuluje, analyzuje a implementuje. Ve druhém už ne. Je zde nucen brát v úvahu požadavky zadavatele, byť někdy ne zcela ideální, musí brát v úvahu fungování systému ve vnějším světě.

Zkušenost ukazuje, že metody a postupy navržené a vyzkoušené na jednoduchých příkladech nestačí patřičně zvětšit, aby fungoval velký systém. Vývoj velkých programových systémů musí probíhat podle jiných pravidel, komplexnějších. Někdy se pak mluví o malém a velkém programování, každému odpovídá jiná oblast teorie programování.

Programování malých celků odpovídá teorie algoritmů, programovací techniky, algoritmy a programování jednotlivých modulů, techniky pro návrh struktury dat a algoritmů.

Tvorbou velkých programových systémů se zabývá SW inženýrství. Používá metody technologie programování, dekompozici systému na menší celky, až se dospěje k zadání modulů, řešitelných programováním v malém.

Metody a metodologie

Metoda je procedura nebo technika, pomocí níž se provede nějaká etapa nebo její část z životního cyklu programu (technika pro definici požadavků, pro návrh databáze, pro návrh programu ap.).

Metodologie je kolekce metod, které jsou vybrány na základě společné podstaty a společně podporují větší část, obvykle několik etap životního cyklu programového díla.

Metody používají různé **nástroje** pro záznam příslušné části životního cyklu (textové a grafické editory, formuláře, apod.). Nástroje mohou být univerzální (běžné editory textové) nebo přizpůsobeny pro danou metodu či metodologii (tools, CASE). Každá metoda vytváří jeden model systému, někdy víceúrovňový, hierarchický nebo jinak rozložený, většinou v grafické podobě.

Model je pohled na systém s následujícími vlastnostmi:

- popisuje **celý** zpracovávaný systém, tedy ne některou jeho místně, funkčně nebo časově vymezenou část,
- zobrazuje jen **některé aspekty** systému, ne všechny aspekty celého systému,
- zobrazuje určitou **abstraktní úroveň** zobrazovaných aspektů, jde jen do takových podrobností, které jsou v daném kroku účelné.

Pro IS se používají nejčastěji modely podle Yordonovy strukturované analýzy:

datový, statická struktura databáze (konceptuální schéma, ERD),

funkční, základní funkce nad daty (diagram datových toků DFD, diagram funkční struktury DFS a minispecifikace),

dynamický, časové návaznosti funkcí (STD, ELH).

Podrobněji probereme metody pro podporu etap životního cyklu IS v následujících kapitolách.

1.2. Klasifikace informačních systémů

Hlediska klasifikace informačních systémů

Jak víme, automatizované informační systémy jsou prostředkem pro sběr, uložení, uchování, zpracování, vyhledávání a formátování informací, to vše pro účely dalšího rozhodování.

Automatizovaných informačních systémů je celá řada typů. Zatím jsme probírali nejběžnější operativní IS, určené k evidenci rozsáhlých dat. Existují i další typy informačních systémů. Můžeme je dělit

- podle typu a struktury informací v nich uchovávaných,
- podle způsobu vyhledávání informací,
- podle operací nad údaji prováděných a
- podle prezentace výsledků vyhledávání.

Společným znakem všech typů je **existence databáze** pro uložení informací.

Charakteristiky typů informačních systémů

Následující rozdělení IS není možno chápat striktně, mnohé reálné IS mají vlastnosti neúplné vzhledem k jednomu typu nebo kombinované z několika typů IS.

1. **Operativní** (též transakční, OLTP) informační systémy

   důsledně strukturovaná informace na atomické atributy, tabulky ve 3NF,
   operace INSERT, SELECT, UPDATE, DELETE,
   přesné dotazy, vyhledávání informací je založeno na této přesné struktuře.

2. **Faktografické** informační systémy (FIS)

   informace strukturovaná i nestrukturovaná, relativně stálá,
   má funkce navíc pro provádění operací s informacemi vybranými z databáze,
   medicínské, právnické, inženýrské, geografické apod.

3. **Textové** informační systémy (též dokumentografické DIS, či vyhledávací)

   informace (částečně) nestrukturovaná formou volného textu, obrázků, zvuků,
   přibližné dotazy, vyhledávání textové informace dle vzorků (klíčových slov),
   metody vyhledávání s předzpracováním textů nebo vzorků nebo obojího.

4. **Geografické** informační systémy (GIS, systémy pro práci s 2D a 3D prostorem)

   informace strukturovaná, prostorová
   má funkce navíc pro práci s prostorovou informací.

5. Systémy pro **podporu rozhodování** (datové sklady DS, Decision Support Systems DSS, OLAP)

   informace strukturované i nestrukturované, odvozené, agregované,
   integrované funkce databázové, vyhledávací i expertní,
   netriviální metody navíc pro analýzy a zpracování informací (dolování znalostí),
   výsledky ukládá opět do databáze jako fakta nebo pravidla,
   prezentuje výsledky také formou grafickou,
   dokonalý interface, určeno pro manažery.

6. **Expertní** systémy

   databáze strukturovaná se specifickými informacemi z úzké dané oblasti zájmu,
   uloženy obecné znalosti a odvozovací mechanizmus,
   dotazy v téměř přirozeném jazyce, systém je analyzuje a na základě znalostí z databáze formuluje odpověď.

7. **Real-time** databáze

   databáze strukturovaná,
   dotazy termínované, funkce s prioritou dotazů,
   řízení transakcí dle priorit a kritické doby nutné k ukončení transakcí,
   fyzická organizace jiná, databáze uložená částečně v paměti.

Z vyjmenovaných typů IS si zopakujeme a rozšíříme své znalosti o operativních IS, podrobně probereme datové sklady a v závěru se stručně seznámíme s textovými IS. Všechny typu IS jsou podrobně probírány v samostatných navazujících předmětech.

1.3. Řízení prací při programování IS

❑ Pracovní týmy

Při vytváření jakéhokoliv velkého díla člověk jen výjimečně může být jediným tvůrcem, vystupovat jako sólista. I kdyby byl zvládal všechny dílčí etapy tvorby, zřejmě by neobstál v konkurenci přinejmenším z hlediska rychlosti.

Většinou řešitelé spolupracují s jinými lidmi a vytvářejí mezi sebou různé vazby. Chování skupiny lidí nelze vysvětlit z prostého souhrnu práce a chování jednotlivců. Každý člen skupiny má svou svéráznou a pro něj charakteristickou roli a jeho jednání se promítá do činnosti ostatních členů skupiny a zároveň je jimi zpětně ovlivňováno. V pracovním týmu se mohou rozvíjet nejrůznější vzájemné vazby v závislosti na tom, jak jsou jedinci úspěšní, jak je úspěšná celá skupina i jak postupně krystalizují vztahy vzájemných závislostí.

Pracovní tým se skládá z řady osob a obvykle ne všichni stejně dobře znají všechny SW profese. Jako v jiných oborech lidské činnosti je vhodné, pohodlnější i efektivnější, když se členové týmu specializují a tak dosáhnou jak vyšší kvality, tak efektivnější produktivity práce.

❑ Softwarové profese

Má-li být výsledného IS dosaženo efektivně, musí být práce vhodně řízeny. **Řízení prací** provádí **manažer projektu**. Ten provádí nebo řídí:

- Komunikaci se zadavateli: píše nabídky projektů potencionálním zákazníkům, specifikuje, k čemu projekt slouží, navrhuje postup věcně a časově, navrhuje a konzultuje finanční nároky; tato činnost je z velké části věcí zkušenosti, manažerských schopností i speciálního vzdělání vedoucího.
- Plánování projektu: po získání zakázky definuje harmonogram projektu, jeho etapy, časovou návaznost etap, případná omezení uživatelů v průběhu tvorby; plánuje testování, školení, předávání produktu, údržbu apod.
- Řízení rozpočtu: zpracovává a řídí rozpočet nejen na vlastní vývoj SW, ale i na nákup HW a případný podpůrný SW, na cesty, školení, tvorbu dokumentace, reklamu.
- Výběr řešitelů: vybírá řešitele, volí vhodný typ týmu a vedoucí týmu pro jednotlivé etapy; zkušenosti ukazují, že je výhodné vybírat do vedoucích pozic špičkové odborníky, protože na nich především záleží výkonnost celého týmu.
- Kontrolu stavu projektu: definuje způsob kontroly a plánuje kontroly v klíčových stavech řešení; někdy musí provádět okamžité změny řešitelů při mimořádných okolnostech, při nedodržování harmonogramu nebo nekvalitní práci, aby celkový harmonogram byl dodržen.
- Prezentaci výsledků: připravuje, organizuje předávání výsledků zákazníkovi, ve spolupráci s ostatními řešiteli provádí prezentaci výsledků.

❑ Typy řešitelů - specialistů

Jedním z úkolů manažera je výběr řešitelů. Zřídka je k dispozici dostatečný výběr univerzálních pracovníků, kteří umí vše a mohou podle potřeb provádět v průběhu ŽC kteroukoliv práci. Pak by se mohly práce dělit pouze podle objemu.

Častější i výhodnější případ je výběr specialistů na různé SW profese. Uvedeme si, které typické profese to jsou – přičemž může tentýž člověk ovládat více činností. Také se můžeme setkat u různých firem s různými názvy uváděných profesí.

- Vedoucí programátor, **manažer:** měl by to být špičkový programátor, který umí všechno. Z pochopitelných důvodů kapacitních nedělá vše, ale v případě potřeby je schopen jednak kvalifikovaně posoudit práci ostatních, jednak pomoci s řešením problémů.
- **Ideový programátor** je jakýsi „náhradní“ vedoucí; opět špičkový programátor, oponent vedoucího v situacích nejednoznačných a případná náhrada vedoucího. Jeho hlavním úkolem je vedení programátorských prací.

- **Analytik** provádí analýzu zadání a zpracovává modely systému (je to „architekt“ systému); pro IS to jsou základní modely datový, funkční a dynamický. Úzce spolupracuje prostřednictvím manažera se zadavatelem.
- **Návrhář** hotové analytické modely doplňuje o řadu technických a systémových detailů; provádí optimalizaci přístupu do databáze, indexovou, transakční analýzu, doplňuje systémové funkce; navrhuje modulovou strukturu funkcí.
- **Systémový programátor** je specialista na styk se systémovým prostředím – s operačním systémem. Konzultuje s ostatními řešiteli, případně realizuje speciální systémová makra.
- **Specialista pro internet** realizuje portál IS a jeho internetové a intranetové funkce, případně podobně jako systémový programátor konzultuje a pomáhá ostatním.
- **Specialista na jazyk** - programovací jazyk či SŘBD opět je konzultantem ostatním nebo programátorem pro některé funkce v SŘBD.
- **Výkonný programátor**, kodér je klasický programátor, realizátor „programování v malém“; při dobře provedené analýze a návrhu je to rutinní práce, vyžaduje jen dobrou znalost implementačního prostředí a teorie algoritmů.
- **Testér** připravuje a provádí testování hotových částí nebo celého IS. Je vhodné, aby to byla jiná osoba než tvůrce – analytik i programátor. Jeho snahou je rozbít program, najít všechny díry, chyby. Řešitel má obvykle snahu potvrdit správnost svého díla, případně nevědomky testuje již promyšlené cesty programem a nedomýšlí všechny situace. Testér působí také jako první oponent, dříve než se dílo dostane k uživateli.
- **Dokumentátor:** dokumentaci jednotlivých částí systému pořizují řešitelé, dokumentátor ji kompletuje, organizuje, integruje, je zodpovědný za její úplnost, aktuálnost, správnost.
- **Knihovník** má na starosti správu knihoven projektu; přejímá od řešitelů hotové komponenty, zpřístupňuje je ostatním. Také udržuje různé verze systému.
- Další **administrativní, technické** a jiné síly.

❑ Organizace týmů

Na organizaci členů řešitelského týmu, jejich vzájemných vazbách a spolupráci významně závisí úspěch výsledku. Vzájemná komunikace řešitelů zabírá dle zkušeností až 50% času řešení, proto je důležité vhodné rozložení práce a pravomocí.

Dle organizace týmů se někdy rozlišují následující hlavní typy [DK88]:

1. Nestrukturované (demokratické) týmy, bez vedoucího a dělící si práci dle objemu:

- **Osamělí vlci** jsou individuality, kde každá osoba umí vše – analýzu, návrh i implementaci a chce si to provádět odděleně od ostatních. Obvykle se domluví jen na rozdělení systému na části – subsystémy a na vzájemném předávání dat mezi subsystémy. Je zřejmé, že doba řešení i kvalita výsledku je odvozena od nejpomalejšího a nejhoršího člena.
- **Horda** je skupina řešitelů, kde se opět předpokládá, že každý umí vše a práci si dělí libovolně dle objemu – co je právě potřeba dělat. Autor tohoto dělení týmů uvádí, že výhodnost této organizace je založena na úvaze: když jeden parník překoná oceán za 4 dny, pak dva parníky překonají tuto vzdálenost za 2 dny.
- **Demokratická skupina** řešitelů si dělí práci dle dohody řešitelů, například podle profesí. Jde o dobrou organizaci, pokud jsou všichni členové týmu disciplinovaní, produktivní a zodpovědní. Jinak jsou lepší hierarchické týmy.

2. Strukturované (hierarchické) týmy s vedoucím týmu a dělením prací dle profesí:

- **Chirurgický tým** má hlavního řešitele, který je současně ideovým programátorem a vše je podřízeno jeho vedení. Ostatní členové jsou jemu k službám podle svých profesí a pracují podle jeho pokynů. Jde o velmi efektivní organizaci pro tvorbu, jen mohou být někteří členové dostatečně nevyužiti, když čekají na požadavky vedoucího.
- **Tým hlavního programátora** má hlavního programátora řídícího celý tým, i ideového programátora, který mu dělá partnera a oponenta v důležitých rozhodnutích; tým doplňuje stabilně administrativní pracovník. Ostatní profese jsou najímány podle okamžitých potřeb – etap ŽC a možností výběru. Jde o velmi dobrou organizaci, kde profesní specialisté jsou lépe využiti, protože v průběhu času mohou postupně pracovat na více projektech.

3. Vícetýmová organizace

Zkušenosti ukazují, že jeden člověk je schopen přímo řídit ne více než 6 – 8 lidí. Pro větší projekty a tedy i větší týmy je tedy vhodná vícetýmová organizace. Spolupracuje více skupin, které řídí manažer prostřednictvím jejich vedoucích. Každá skupina může být různě organizovaná. Nejvíce záleží na konkrétních osobnostech skupiny a odtud se odvíjí typ týmu. Nutit například výrazné samostatné individuality dělat asistenty v chirurgickém týmu může být kontraproduktivní.

❑ Shrnutí zkušeností s týmovou prací

Shrneme-li zkušenost z vývoje mnoha systémů realizovaných různými typy týmů, mohou se formulovat následující pravidla:

- Nalezení vhodných vedoucích osobností je nejdůležitější podmínkou úspěchu projektu. Na jejich schopnostech odborných, organizačních a na jejich produktivitě je nejvíc závislý úspěch projektu.
- Vybraným vedoucím týmů je potřeba dát k dispozici vhodnou sestavu specialistů - dle typů lidí a dle profesí. Podle typů lidí pak zvolit strukturu týmu a jeho řízení.
- Větší projekty řešit raději jako vícetýmové s opětovným výběrem vedoucích týmů a jejich typů.

❑ Empirické zákony SW fyziky

Pokusy normovat práci na vývoji software se vyskytly již tehdy, kdy se začaly vytvářet programy na zakázku. Není jednoduché ohodnotit nebo naplánovat práci analytiků a programátorů. Přitom pro práci na velkých programových projektech je správný odhad náročnosti prací (ceny i doby projektu) velmi důležitý. Podhodnocený odhad se prodraží, nadhodnocený může odradit zákazníka.

Především u velkých softwarových firem v USA byl tento proces studován systematicky. Vzniklo nové odvětví informatiky, nazývané **SW fyzikou**. Sledováním mnoha konkrétních SW projektů, jejich vývoje, údržby, zvětšování a nakonec statistickým vyhodnocením dospěli různí autoři k řadě výsledků.

Výsledky jsou nazvány **softwarovými rovnicemi** a formulovány jako empirické zákony, které zní jako zákony Murphyho. Ovšem zkušený informatik tuší, že formulují hlubokou pravdu. Uvedeme si jen jednoduché příklady. Z následujících příkladů vyplývá, že hlavními omezujícími faktory programování jsou meze lidské psychiky a problémy týmové práce.

Příklad: *Označme*

T - spotřeba práce v člověko-měsících,
N - počet lexikálních atomů programu (míra délky),
N1 - počet výskytů operandů v programu,
m1 - rozsah slovníku operandů (počet různých operandů)
m2 - rozsah operací v programu (počet operací a oddělovačů v programu)

P - produktivita práce (v jednotkách délky programu za měsíc),
D - doba realizace projektu v měsících,
S - průměrná velikost týmu při řešení projektu,
c – libovolná konstanta.

*Odhad spotřeby lidské práce **T** dle softwarové rovnice je:*

T # c*N*N1*m2/m1*log(m1+m2)

kde # značí „je odhadem“

♦

Příklad:

Dle jiného autora produktivita

P # c1*D(4/3)*T**(-2/3)**

protože N=P*T *(dle definice P)*

N # c2*T**(1/3)*D**(4/3) *(*)*

Realizujeme-li teoreticky program dvakrát s dobami řešení D1 a D2, kde D2<D1, obvykle pak program psaní ve spěchu bývá delší a můžeme předpokládat, že N2>N1. Vyděleníím rovnice (*) s dosazenými hodnotami N1,T1,D1 a N2,T2,D2 dostaneme:

T2 > (D1/D2)**4 * T1

pro D2 = 3/4*D1 *dostaneme* T2 > 3.2*T1,

***Závěr:** **zkrácení termínu o 25% zvýší spotřebu práce o 220%**, nebo obráceně:*

***je-li na řešení dost času, může dojít k podstatnému snížení nákladů**.*

Při delším termínu lze řešit projekt v menším týmu, problémy komunikace jsou lepší, myšlenky mohou dozrát. Na druhé straně je nutno motivovat řešitele k průběžnému řešení, ne odkládat z důvodu existence pozdějšího termínu.

♦

Podobných slovně formulovaných výsledků, vyplývajících ze SW rovnic, je řada.

Autoři Lehman a Belady formulovali následující zákony:

- **Zákon trvalé proměny**: Systém používaný v reálném prostředí se neustále mění, dokud není cenově výhodnější systém restrukturalizovat nebo jej nahradit zcela novou verzí.
- **Zákon rostoucí složitosti**: Při evolučních změnách je program stále méně a méně strukturovaný (vzrůstá entropie) a vzrůstá tedy jeho složitost. Odstranění narůstající složitosti vyžaduje dodatečné úsilí.
- **Zákon vývoje programu**: Rychlost změn globálních atributů systému se může jevit v omezeném časovém intervalu jako náhodná. V dlouhodobém pohledu se však jedná o seberegulující proces, který lze statisticky sledovat a předvídat.
- **Zákon invariantní spotřeby práce**: Celkový pokrok při vývoji projektů je statisticky invariantní. Jinak řečeno, rychlost vývoje programů je přibližně konstantní a nekoreluje s vynaloženými prostředky.
- **Zákon omezené velikosti přírůstku**: Systém určuje přípustnou velikost přírůstku v nových verzích. Pokud je limita překročena, objeví se závažné problémy týkající se kvality a použitelnosti systému.

Jiný autor, klasik softwarového inženýrství B. W. Boehm formuloval pravidla nazývaná **Top 10:**

- Nalezení a oprava SW problému je 100x dražší než odstranění chyby ve fázi zadání a analýzy.
- Údaje uvedené v plánu lze snížit maximálně o 25% (přidáním lidí, peněz, …), ne víc.
- Údržba stojí 2x tolik, co vývoj.
- Vývoj a údržba jsou především funkcí velikosti produktu.
- Variace lidských schopností je příčinou největších variací v produktivitě.
- Podíl ceny SW:HW v roce 1955 byl 15:85, v roce 1985 85:15 a stále roste ve prospěch SW.
- Pouze 15% práce tvoří kódování.
- Instrukce aplikačního programu je 3x dražší než instrukce individuálního programu, instrukce systémových programů 9x dražší.
- Inspekce programu zachytí 60% chyb.
- Mnohé SW procesy splňují Paretovo rozložení:

  20% modulů přispívá k 80% ceny
  20% modulů obsahuje 80% chyb
  20% chyb spotřebuje 80% rozpočtu na opravy
  20% modulů spotřebuje 80% času výpočtu
  20% nástrojů se používá 80% času.

Další zajímavé zákony

- Murphyho zákon: Pokud se něco může pokazit, pak se to pokazí.
- Projekty postupují rychle, až jsou dořešeny z 90%, pak zůstanou z 90% vyřešeny navždy.
- 90% projektu spotřebuje 10% úsilí. Zbývajících 10% projektu spotřebuje 90% úsilí.
- Chybové nástrahy převýší zkratky k úspěchu.
- Pro každý složitý problém existuje řešení – které nebude fungovat.
- Žádný problém není tak strašný, abyste od něj nemohli odejít.
- Kdykoliv někdo řekne „teoreticky“, má na mysli „nikoliv doopravdy“.

❑ Praktické využití SW zákonů

Tvůrce IS, zvláště nezkušený začátečník, zaplatí někdy vysokou cenu za přeceňování vlastních sil či vysoké sebevědomí o svých schopnostech. Mnoho řešitelů má tendenci rozsah prací a dobu řešení odhadovat velmi optimisticky, a to i po několikaletých zkušenostech. Řešením je buď si vysledovat a definovat „empirickou konstantu“, obvykle rovnu 2 až 4 i více, kterou si původní odhad vynásobí, nebo ještě lépe prostudovat a použít znalosti SW zákonů.

Odhad rozsahu prací a doby řešení je důležitý při **uzavírání smluv se zadavatelem**. Přitom již víme, že nefunguje například pravidlo, že příliš krátký termín se dá kompenzovat větším počtem řešitelů. Sebevědomí manažera se musí projevit v dobrém a věcném zdůvodnění zadavateli, proč není možný požadovaný termín pro požadovaný rozsah prací.

K některým praktickým otázkám se ještě vrátíme na konci kapitoly o zadání IS.

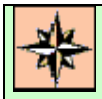
Shrnutí pojmů 1.

Systém, systémová analýza, systémový přístup k řešení projektů, informační systém.
Životní cyklus vývoje IS a jeho etapy.
Klasifikace IS podle druhu evidovaných informací a práce s nimi.
Řešitelské týmy, SW profese.
Metody a metodologie.
SW fyzika, empirické SW zákony.

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 a co automatizovaný informační systém?
5. Popište životní cyklus vývoje informačního systému.
6. Podle jakých hledisek se dělí informační systémy a které typy znáte?
7. Které SW profese pracující na vývoji IS znáte.
8. Jak je možno dělit týmy řešící rozsáhlé IS?
9. Co jsou metody, jejich nástroje a metodologie určené pro vývoj IS?
10. Co je SW fyzika a co formulují její zákony?

2. ZADÁNÍ A ANALÝZA INFORMAČNÍHO SYSTÉMU



Průvodce studiem

V Databázových a informačních systémech (DAIS) jsme probrali celý životní cyklus vývoje IS. Přesto pro rozsáhlé IS bude vhodné rozšířit tyto naše znalosti o další postupy a podrobnosti.

Abychom neopakovali stejnou látku z DAIS, uvedeme vždy jen přehled základních informací probírané etapy ŽC a podrobněji uvedeme nová rozšíření látky. Kdo některé partie zapomněl, může si je zopakovat podrobně včetně příkladů v učebnici DAIS.

Čas ke studiu kapitoly: 2x2 hodiny studium, 2x3 hodiny projekt

Cíl V této kapitole

- si připomenete životní cyklus vývoje IS a jeho etapy,
- seznámíme se s některými novými metodami analýzy, návrhu implementace, testování,
- vše si vyzkoušíte na vlastním větším projektu, řešeném v týmu.

Výklad

2.1. Zadání IS

Výchozí formulace zadání

Již víme, že zadání požadavků na budoucí IS zahájí zadavatelovo rozhodnutí využít počítač pro řešení nějakého reálného databázového problému. Zadavatelem zde rozumíme ne jen jediného člověka, s nímž se uzavírají formální dohody, ale všechny osoby, které na upřesňování zadání spolupracují. Obvykle to jsou zástupci budoucích uživatelů všech rolí.

Zadání je vhodné požadovat od zadavatele písemně, i když to je obvykle jen první verze zadání. Protože zadavatel sice zná věcnou stránku zadání, ale většinou neví nic o programování, analýze a formálních specifikacích, zadání obvykle formuluje slovně. Proto zadání bývá často nejednoznačné, neúplné, nepřesné. Cena za předělání programu po realizaci takového zadání by byla vysoká *(jako cena přestavby hotového 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.

Někdy řešitelé systému prosazují názor, že:

1. napřed se musí přesně specifikovat co se vlastně má dělat (když nevíme, co máme dělat, tak to přece nemůžeme dělat !)
2. pak se provede pořádná analýza;

3. pomocí typových prvků se slepí programy, použijí se moderní metody programování a tak je zaručeno fungování;
4. systém se otestuje, vyzkouší a je to; uživatelé se už nějak přizpůsobí nebo se s tím smíří.

Výzkumy i zkušenosti však ukazují, že možnost oddělení specifikace zadání od dalších etap životního cyklu je pouhá iluze. Napsat program při úplně a správně zadané specifikaci není žádný problém (pro koho je, tomu použití moderních metod moc nepomůže). Problém tedy není v ladění programů, ale v ladění specifikace. V tomto smyslu můžeme chápat vše obráceně: finálním produktem je specifikace, kterou jsme dostali v průběhu realizace systému.

Při tvorbě systémů se v požadovaných kritériích požaduje na prvním místě, aby systém odpovídal co nejlépe požadavkům uživatelů, na druhém, aby byl hotový včas. Na třetím je adaptabilita při případných úpravách a ta bývá v rozporu s efektivitou programů.

Ideálem tedy je **přesná, úplná a bezesporná specifikace**. Pro takovou specifikaci by bylo ideální použití nějakého formálního jazyka, to však zadavatel obvykle neumí. Proto se na této úrovni dělají kompromisy ve formě zpracování zadání, proto se používají různé pseudokódy a zpracovávají různé modely, upřesňující zadání.

Požadavky uživatelů z věcného hlediska dělíme obvykle na funkční a nefunkční.

Funkční požadavky

Pro počáteční etapy komunikace a upřesňování požadavků je vhodné vypracovat si systém otázek pro zadavatele. Uvedeme si jejich osnovu.

Funkčními požadavky rozumíme požadavky na věcný, problémový obsah systému. Definují

1. PROČ vůbec je zapotřebí nový systém – popis současného stavu, proč nevyhovuje, co má mít nový.
2. K ČEMU má sloužit – vnitřní evidenci, vnějšímu uživateli, managementu apod.
3. KDO s ním bude pracovat - běžně, příležitostně, občas – formou Use Case diagramu.
4. Jaké budou VSTUPY do systému, seznam evidovaných údajů, objektů a atributů.
5. Jaké budou VÝSTUPY ze systému, seznam výstupních sestav a pracovních výstupů.
6. Jaké FUNKCE bude systém plnit – formou tabulky Událost – Reakce.
7. Jaké je OKOLÍ systému, jeho zdroje a cíle informací – formou kontextového diagramu.

Aby měl zadavatel nějaké vodítko pro úplný popis požadavků, jsou mu k tomu nabídnuty prostředky, jako jsou **modely vnějšího chování** systému

- okolí – zdroje a cíle dat jako kontextový diagram (DFD)
- funkce jako tabulka událost v realitě - reakce systému zapsaných formou krátkých textů
- kdo s IS pracuje jako model jednání (Use Case)
- dělení funkcí do hierarchie a struktury, návrh menu systému s popisem elementárních funkcí (komunikace s uživatelem)

Tyto počáteční modely pak slouží jako výchozí pro první fáze analýzy, pro datový model (ERD nebo objektový model), pro funkční model (DFD) a dynamický model (slovní a grafické scénáře).

Nefunkční požadavky

Další požadavky, zvané nefunkční, jsou omezení kladená na systémové služby:

1. Požadavky na výsledný program: efektivita (rychlost, výkon, paměťová náročnost), spolehlivost, přenositelnost a použitelnost.

2. Požadavky na způsob řešení: často určují metody pro analýzu, návrh, použití standardů z oblasti metodiky vývoje, dokumentace, programovací jazyk, prostředí (celkem podmínky dodání, implementační požadavky a použití standardů).
3. Vnější požadavky: ostatní nefunkční požadavky, jako implementační požadavky, podmínky dodání, omezení daná legislativou, cenová omezení, požadavky na spolupráci nového systému s již existujícími systémy, daná organizační struktura, bezpečnost, platné zákony, vyhlášky, atesty apod.

❑ Prověření splnitelnosti požadavků

Validací požadavků (prověřením splnitelnosti) by měla začínat spolupráce na upřesňování zadání a měla mít 4 kroky:

- Validace potřeb uživatele: analýza může některé uživatelovy požadavky vyloučit jako zbytečné a naopak odhalit chybějící potřeby, případně navrhnout kompromis mezi rozpornými potřebami různých uživatelů.
- Prověření konzistentnosti požadavků, žádný by neměl kolidovat s ostatními.
- Prověření úplnosti požadavků: soupis by měl obsahovat všechna omezení a funkce požadované uživatelem.
- Reálnost požadavků: nesplnitelné vyloučit nebo upřesnit.

Pro **testování specifikací požadavků,** které je součástí zadání, by měla být i zadavatelem dodaná testovací data. Testovací data by měla zahrnovat všechny

- typické případy údajů pro kontrolu správnosti běžných funkcí,
- limitní případy údajů pro kontrolu krajních správných i krajních chybných dat,
- chybové případy všech možných typů pro kontrolu správné reakce systému s hlášením chyby.

U IS jsou po provedení datové analýzy definovány všechny domény atributů, proto stačí, když zadavatel dodá svá reálná data pro testování.

❑ Obrana proti nežádoucím proměnám zadání

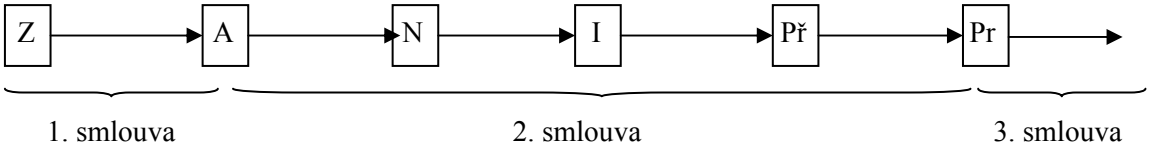
Víme, že se zadání v průběhu řešení může měnit, doplňovat. Aby tyto změny nebyly nekonečné nebo aby si zadavatel byl vědom nárůstu prací při změnách zadání, je vhodné zařadit bod o změnách zadání do smlouvy.

Jaká je obrana řešitele proti nežádoucím proměnám zadání – jak můžeme formulovat smlouvu:

- **Maginotova linie**: do smlouvy se uvede přesná specifikace zadání s dodatkem, že žádné změny nejsou možné. Taková smlouva je jistě pohodlná pro řešitele, ale zadavatel se zřejmě příště obrátí jinam.
- **Švýcarská obrana**: řešitel vybuduje naprosto otevřený systém dokonale zdokumentovaný, do nějž se dělají změny snadno. Do smlouvy se uvede, že změny si provede uživatel sám. Tento případ je vhodný pro firmy, které mají vlastní IT oddělení, ale jeho kapacity nestačí na vývoj celého IS. Po zaškolení jejich pracovníků však udržovat a doplňovat IS mohou sami.
- **Flexibilní systém**: systém je natolik chytře navrhnout a zdokumentován, že změny znamenají jen lokální zásahy, které se nedotýkají ostatních částí. Pak řešitel poměrně dobře provádí údržbu systému v rámci smlouvy i nadále.

Doporučený postup

Protože na začátku řešení obvykle není možné spolehlivě odhadnout dobu řešení a cenu prací, je vhodné rozdělit smlouvu o vývoji IS na tři podle etap životního cyklu takto:



1. Přimět zadavatele co nejúplněji a nejpřesněji (případně pomocí uvedených prostředků) formulovat písemně zadání IS.
2. Na základě písemného zadání sepsat a uzavřít 1. smlouvu o vypracování analýzy na IS, případně i v několika variantách z hlediska věcného i časového a cenového.
3. Na základě výsledku vybrané verze analýzy sepsat a uzavřít 2. smlouvu o realizaci IS včetně předání a zahájení provozu.
4. Pro údržbu provozu uzavřít 3. smlouvu se specifikací, co vše je do ní zahrnuto. Pro případné upgrade – větší rozšíření systému uzavírat samostatné dohody.

2.2. Analýza informačního systému

❑ Co je analýza

Analýza znamená studium problému (jeho poznání, popis, namodelování) dříve, než se začne provádět vlastní řešení. V informatice je předmětem analýzy aplikační oblast reálného světa. Na základě analýzy se většinou provádí implementace nového systému.

Předmětem analýzy je nejčastěji

- existující systém, jehož struktura a funkce jsou zřejmé zákazníkovi z praxe a který se bude automatizovat (poznání a popis současného stavu - automatizovaný či neautomatizovaný);
- existující systém, jehož struktura a funkce nejsou zřejmé ani zákazníkovi, ani řešiteli (vše včetně zadání nových potřeb je nutné analyzovat);
- neexistující systém, o němž má zákazník více nebo méně přesnou představu a neumí požadované funkce řešiteli vysvětlit;
- neexistující systém, o němž má zákazník dostatečně přesnou představu.

Softwarové inženýrství řeší často realizaci neznámých, mlhavě formulovaných systémů.

Výsledkem analýzy je specifikace problému, která definuje:

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

Specifikační dokument bývá součástí smlouvy a proto mívá právní platnost.

Pro informační systém se zpracovávají **3 základní modely: datový, funkční a dynamický**.

2.3. Datová analýza

Datová analýza a datový model

Datový model IS (též konceptuální schéma) popisuje strukturu databáze. Důležité je, aby tabulky databáze byly normované, bez redundancí. Postup detailně známe z předmětu Teorie zpracování dat:

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 (někdy 2 úrovně)
    1. konceptuální schéma modelující realitu
    2. transformovaný ERD pro databázové schéma v 3NF.
- úplné tabulky atributů – datový slovník DD
- seznam dalších IO týkajících se domén, entit a vztahů.

2.4. Funkční analýza

Funkční analýza a funkční model

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 2 úrovně:

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

Diagram datových toků

První vnější pohled na systém celý a jeho funkce se vytváří pomocí tzv. diagramu 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.

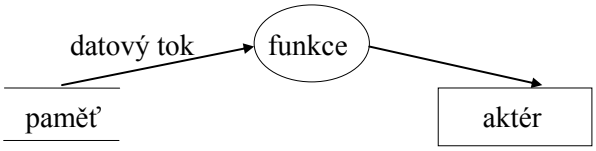
Již víme, že rozsáhlé IS není možno řešit najednou, ale že se dělí na menší zvládnutelné celky. Ve funkční analýze se navrhuje rozložení rozsáhlého systému na základní subsystémy a ty případně dále na části. DFD je tedy nástroj pro grafické rozkreslení těchto částí.

Proto model systému vyjádřený pomocí DFD má obvykle hierarchickou strukturu. Jen velmi malý systém je možno vykreslit jediným diagramem. 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.

DFD pojmenovává a 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



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 rozklad na 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).

U menších IS nebo u jejich subsystémů je možné postupovat i zdola nahoru: vykreslit všechny elementární funkce (jejich seznam by měl odpovídat seznamu funkcí zadaných v tabulce událost – reakce); potom vždy spojit „příbuzné“ funkce do jedné bubliny, až se dostaneme k jediné funkci celého subsystému či IS.

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. Popisuje postup, jak jsou data - datové toky do funkce vstupující transformovány na data výstupní.
3. Popisuje, co funkce znamená věcně, nemusí vyjadřovat způsob implementace funkce (proto není vhodný programovací jazyk). Popisuje však všechny podrobnosti algoritmu dat včetně základního formátování dat na vstupu a výstupu.
4. Nezavádí redundandní popisy, nevyjadřuje znovu, co je zobrazeno v DFD nebo popsáno v DD. Musí však být s nimi konzistentní, být s nimi v souladu.
5. Množina výrazů pro popis algoritmu je jednoduchá a nepříliš rozsáhlá, má používat standardní vyjadřování.
6. Algoritmus musí být srozumitelný analytikovi, programátorovi i uživateli.
7. Popis procesu musí být strukturovaný – vhodné je použití **strukturovaného jazyka**.
8. Minispecifikace jsou vstupem pro etapu návrhu implementace, tam jsou doplněny řadou implementačních podrobností, proto je hodné dodržovat následující pravidle pro jejich zápis.

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

Strukturovaný jazyk

Strukturovaný jazyk je často používaný prostředek pro popis minispecifikací. Je to přirozený jazyk doplněný o omezující pravidla tvorby vět (syntaxe), aby výsledný popis nepřipouštěl několik různých výkladů.

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) JE-LI <podmínka>
  PAK <činnost pro platnou podmínku>

  (2) JE_LI <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>

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).

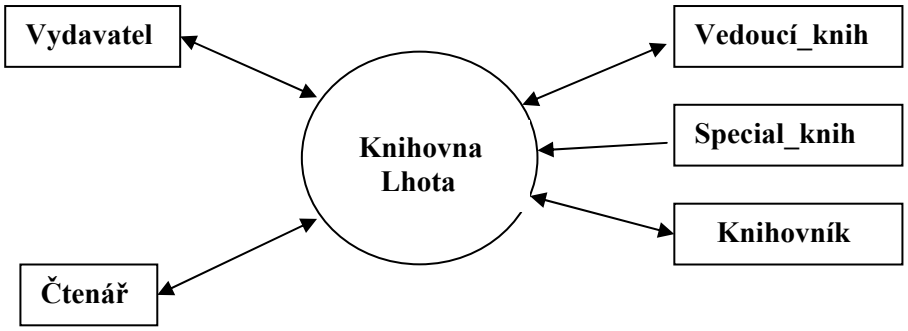
Diagram funkční struktury DFS

Diagram funkční struktury zobrazuje hierarchickou strukturu funkcí. Je dobrým doplňkem k DFD. DFD dělá zvětšeniny funkční struktury systému, DFS zobrazuje průřez hierarchií funkční struktury. DFS zobrazuje, ze kterých podřízených funkcí se popisovaná funkce skládá, nepopisuje jejich vzájemnou komunikaci. Je to přehlednější popis hierarchie funkce.

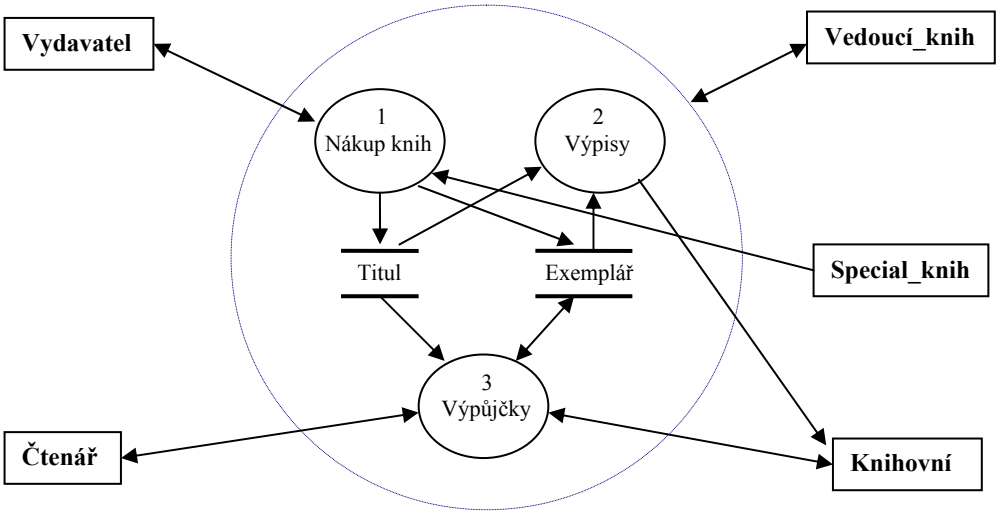
DFS je graf ve tvaru stromu, kde uzly jsou funkce („bubliny“) z DFD. Kořen stromu je název IS (odpovídá kontextovému diagramu), větví se stejně jako hierarchické stupně DFD.

Příklad:

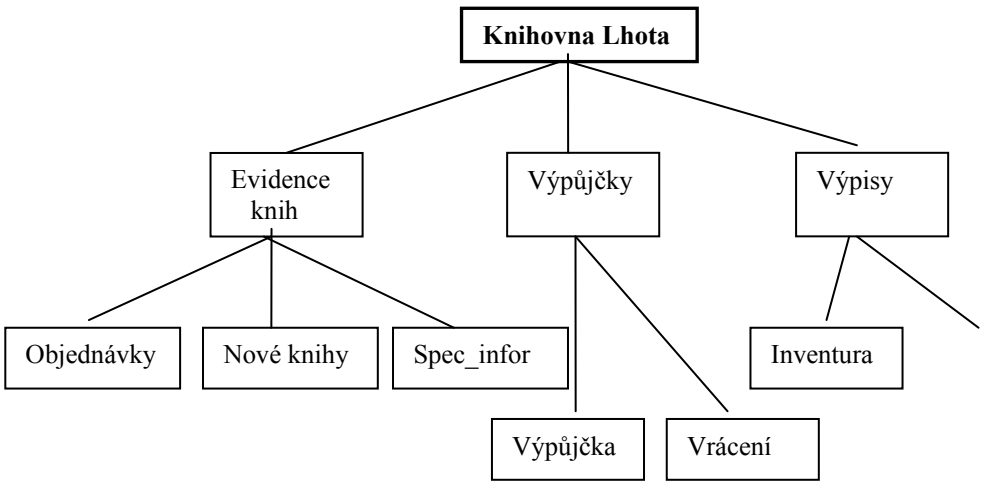
Připomeneme si kontextový DFD a DFD 0.-té úrovně pro IS Knihovna, k nim pak doplníme DFS:



DFD 0. úrovně:



Část odpovídajícího DFS:



DFS tedy funguje jako seznam funkcí a jejich hierarchie.

♦

Pro DFS lze formulovat následující integritní pravidla :

- pro každý DFD obsahuje DFS jeden neelementární prvek;
- pro každou elementární funkci obsahuje DFS jeden prvek nejnižší úrovně;
- rozklad jednoho prvku DFS (jedné funkce) musí počtem a identifikací podřízených prvků odpovídat DFD, popisujícímu strukturu této komunikace.

❑ Rozhodovací tabulky

Rozhodovací tabulky jsou nástrojem pro upřesnění některých částí algoritmů. Specielně jsou vhodné pro analýzu takových částí, kde se kumuluje více podmínek a jim odpovídajících činností.

Provést úplnou analýzu můžeme pomocí tzv. **rozhodovací tabulky**. V ní vypíšeme všechny podmínky a jejich kombinace i všechny akce, které jsou na podmínkách závislé. Kombinace podmínek jsou sestaveny jako kombinace hodnot tzv. **rozhodovacích proměnných**. Jejich použití si ukážeme na příkladě.

Příklad:

Úkolem procesu je rozhodnout, zda a za jakých podmínek mají cestující v letadle obdržet občerstvení. Pracovník letecké společnosti formuluje příslušná pravidla takto:
„Je-li let obsazen více než z poloviny a průměrná cena letenky přesahuje 350$, pak dáváme koktejly zdarma, pokud to není vnitrostátní let. U vnitrostátních letů účtujeme všechny koktejly, když je podáváme. Podáváme je, jen když je let více než z poloviny obsazen.“

Prostým přepisem zadání do strukturovaného jazyka např. tvar

```
VYBER ALTERNATIVU
    PŘ.1: obsazeno více než 50% a letenka stojí více než 350$ a není to vnitrostátní let
          podávej koktejly zdarma
    PŘ.2: vnitrostátní let
          účtuj podávané koktejly
          JESTLIŽE obsazeno více než 50%
             PAK    podávej koktejly
             JINAK  nic
```

Pravděpodobně máme pocit, že uvedená minispecifikace má nedostatky, že není jednoznačná nebo úplná.
V našem příkladě jsme použili 3 rozhodovací proměnné: vnitrostátní let, obsazení letu alespoň z 50% a cena letenky převyšuje 350$. Každá z nich může nabývat jedné z hodnot ANO/NE, celkem je 8 možností. Tabulka obsahuje 8 pravidel pro občerstvení (podmínky a odpovídající akce):

	PODMÍNKY			AKCE	
PRAVIDLA	vnitrostátní let	obsazen > 50%	cena > 350$	občerstvení	bezplatně
1	A	A	A	A	N
2	N	A	A	A	N
3	A	N	A	N	**?**
4	N	N	A	**?**	**?**
5	A	A	N	A	N
6	N	A	N	?	**?**
7	A	N	N	N	**?**
8	N	N	N	?	**?**

Vidíme, které případy nejsou popsány a musíme se na ně doptat.

♦

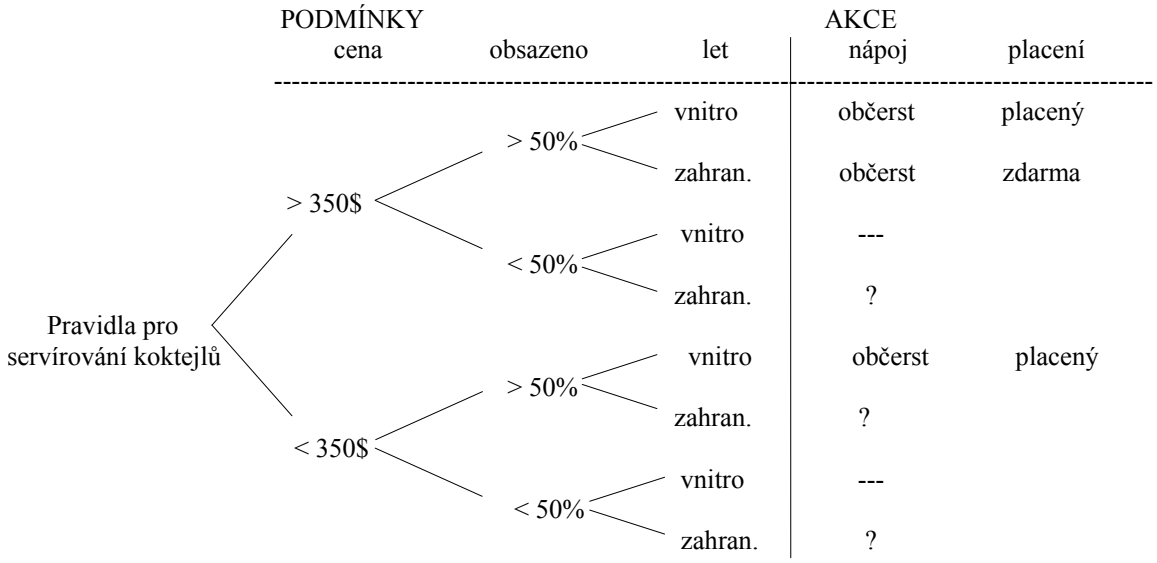
❑ Rozhodovací stromy

Pokud se zadavateli nezdá rozhodovací tabulka dost srozumitelná, může analytik použít ekvivalentní nástroj - rozhodovací strom. Je to grafické vyjádření rozhodovací tabulky a jeho sestrojení pro každou tabulku je mechanické. Jde tedy opět o nástroj pro analýzu části minispecifikace, kde se vyskytuje řada rozhodovacích podmínek, na kterých závisí vykonání nějakých akcí.

Zleva doprava odpovídá každý sloupec jedné elementární rozhodovací proměnné, další sloupce znamenají výslednou akci.

Příklad:

o podávání občerstvení v letadle, odpovídající výše uvedené rozhodovací tabulce:



♦

❑ Matice a analýza afinity

Mnoho vlastností systému lze analyzovat pomocí matic. Lze pomocí nich přehledně znázornit, které entity nebo akce ovlivňují jiné entity nebo akce, nebo zaznamenat výskyt jednotlivých entit či akcí ve funkcích apod.

Matice, která se dá použít pro kontrolu správnosti a úplnosti funkcí pracujících s databází, je ISUD matice.

ISUD matice

Funkce pracují s datovými paměťmi a vytvářejí v nich nové záznamy, mění a ruší záznamy nebo záznamy jen čtou. Tyto vztahy mezi funkcemi a datovými paměťmi jsou přehledně shrnuty v tabulce nazývané ISUD (**I**nsert, **S**elect, **U**pdate, **D**elete).

Řádky matice popíšeme atributy databázové tabulky, sloupce všemi funkcemi s ní pracujícími. Do polí tabulky pak zapíšeme písmena I, S, U nebo D, pokud příslušná funkce příslušný atribut zapisuje, čte, mění nebo maže. Z vyplněné tabulky můžeme snadno poznat

- **zda jsou všechny atributy správně zpracovány všemi funkcemi**
- **zda všechny atributy mají vstup i výstup**

Příklad:

V tabulce Titul z IS Knihovna se některé atributy zapisují ihned při uložení nového záznamu, jiné se doplňují později knihovníkem – specialistou, který zařazuje knihy podle různých klasifikací.Typ entity Titul je:

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

Funkce, které s touto tabulkou pracují jsou: nový titul, spec_infor, výpůjčka, vrácení, inventarizace, upomínka, výběr_informací atd.

Kontrola, zda jsou všechny atributy zapsány i využívány, se provede ISUD maticí.

Atributy \ Funkce	nová	spec	vypuj	vrac	inven	upom	vyber
ISBN	I		S		S	S	S
nazev	I		S		S	S	S
id_vydav	I						S
rok	I						S
zanr		I	S				S
jazyk		I					
zeme		I					
anotace		I					
cena	I				S		

Z tabulky například vidíme: atributy jazyk, zeme, anotace *jsou ukládány, ale nikde nepoužity – nikde není S, nikde není U - není možnost atributy měnit, zřejmě tyto možnosti chybí, atd. Je nutno znovu zvážit a doplnit.*

♦

Podobně můžeme využít ISUD matici pro kontrolu úplnosti funkcí nad celými entitami. Řádky matice popíšeme entitami - tabulkami, sloupce všemi funkcemi s nimi pracujícími. Do polí tabulky pak zapíšeme písmena I, S, U nebo D, pokud příslušná funkce příslušnou entitu - řádek tabulky zapisuje, čte, mění nebo maže.

Příklad:

ISUD matice pro IS prodejní firmy – SKLADu. Kontrola úplnosti funkcí nad tabulkami.

Datové paměti: D1: Zboží
D2: Sklad
D3: Prodej
D4: Dodavatel
D5: Objednávka

Procesy: F1. Vyřiď objednávku zákazníka = prodej
F2. Připrav bankovní příkaz = banka
F3. Vytvoř zprávu o prodeji = zpráva
F4. Objednej zboží = objedn
F5. Přejímka zboží = příjem

Akce: I = insert, S = select, U = update, D = delete

ISUD matice:

Datové paměti \ Funkce	Prodej	Banka	Zpráva	Objedn	Příjem
D1: Zboží	S				
D2: Sklad	U			S	U
D3: Prodej	I	S	S	S	
D4: Dodavatel				S	
D5: Objednávka				I	SU

Z matice vidíme:

Nikde není Delete – opravdu se záznamy žádné tabulky nikdy neruší ani nearchivují?
Tabulka Zboží má jen Select – odkud se berou informace? Opět je nutno pro tyto funkce doplnit analýzu.

♦

2.5. Dynamická analýza

❑ Dynamická analýza a dynamické modely

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.

Zavedeme si nový pojem, závislý na pořadí nebo čase, a sice stav systému či jeho částí.

Definice:

Stav je definován podmnožinou 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.

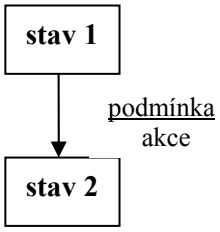
Uvedeme si 2 dynamické modely vhodné pro IS. Bude to obecný stavový diagram a životní cyklus entity, vhodný pro nejnižší úroveň popisu stavů entit.

❑ Diagram přechodu stavů (STD)

Stavový diagram STD (State Transition Diagram) nebo diagram přechodu stavů slouží k modelování chování systému v časových návaznostech, tj 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 nebo vnitřní stavy čekání na událost.

STD znázorňuje jednotlivé stavy a přechody mezi nimi opět pomocí grafu. Uzly tvoří stavy systému, hrany znamenají změny stavů. Stavy jsou znázorněny pomocí obdélníků. Stav systému vyjadřuje interval mezi jednotlivými akcemi, který platí v daném okamžiku.

Změna stavu znamená přechod modelovaného systému z jednoho stavu do druhého. Změna stavu nastane při rozpoznání, že je splněna nějaká podmínka *(příkaz přijat, heslo zadáno, uplynul zadaný čas ap.)*. 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.



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.

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

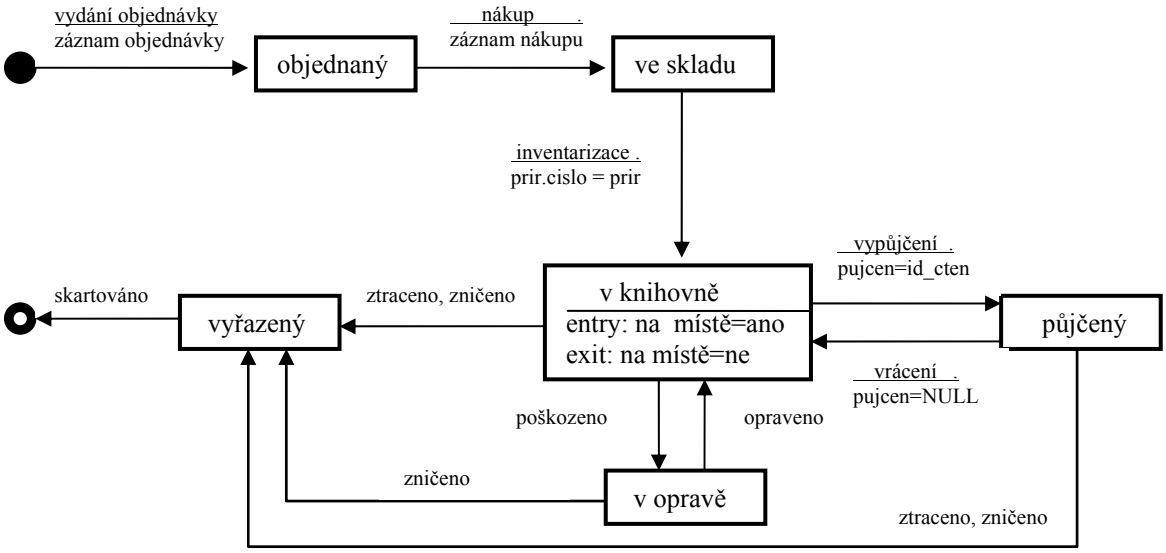
Stavové diagramy se vytvářejí buď pojmenováním všech stavů a hledáním podmínek přechodů mezi nimi, nebo systematickým rozvíjením počátečního stavu do dalších.

Hierarchie STD

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.

Příklad:

V IS Knihovna STD exempláře knihy.



♦

❑ Životní cyklus entity ELH

Životní cyklus entity (Entity Life History - ELH) je jiný způsob popisu transformací dat proti DFD - ten způsob má některé nevýhody: nelze ověřit, zda jsou popsány všechny podněty ke změnám dat nebo ověřit, jak se systém chová při neočekávaném pořadí výskytu dat.

Životní cyklus entity je opět grafický model, který znázorňuje život jedné entity od jejího vzniku až po zrušení pomocí stromového grafu. V kořeni je entita, uzly na nižších úrovních znamenají jednotlivé podněty, které působí na entitu během jejího života v systému. Zobrazuje, jak se mění existence a hodnoty této entity při uvedených událostech.

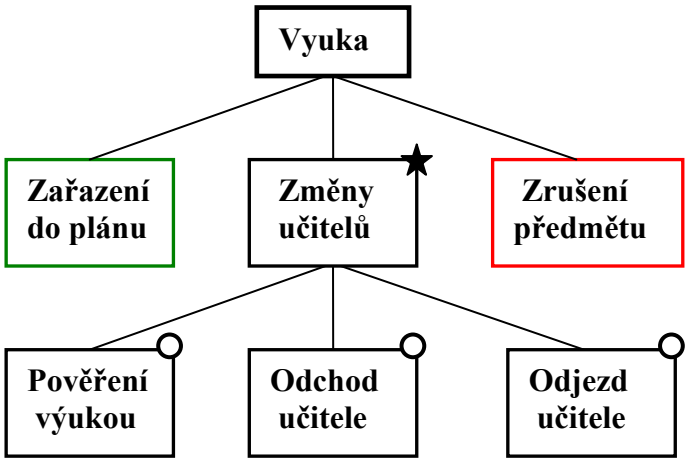
- uzly jsou obdélníkové, označeny entitou (kořen) nebo názvem události (další);
- posloupnost - sekvence je graficky vyjádřena seřazením událostí v každé úrovni stromu zleva doprava;
- větvení se zaznamenává kolečkem v pravém horním rohu obdélníka;
- opakování operace se zaznamená hvězdičkou v pravém horním rohu obdélníka.

Příklad:

Jsou dány entity Ucitel, Predmet a Vyuka *ze školní databáze. Životní cyklus jednoduché entity* Vyuka, *která je vazební mezi* Ucitel *a* Predmět *a zaznamenává kdo co učí a kolik hodin, tedy s atributy* ču, čp, hodin. *Funkce týkající se této entity jsou:*

zařazení předmětu do plánu,
pověření učitele výukou předmětu,
odchod učitele – definitivní,

odjezd učitele - na přechodnou dobu,
zrušení předmětu ze studijního plánu.



Lépe se kontroluje, zda jsou pokryty všechny události z reality funkcemi. První zelený uzel zleva odpovídá počátečnímu stavu (definován předmět, zatím bez učitele), prostřední černé uzly jsou vnitřní uzly, pravý červený je koncový. Nejsou tu zobrazeny přechody mezi stavy.

♦

❑ Vztahy mezi nástroji

Každý uvedený nástroj modelování IS je zaměřen na jiný aspekt modelovaného systému (ERD na datové objekty a vztahy mezi nimi, DFD na procesy a toky dat mezi nimi, STD na koordinaci a řízení procesů v čase, tedy časové charakteristiky systému). Jednotlivé skutečnosti se však zpravidla projevují ve více aspektech systému. Například data uložená o zákazníkovi se vyskytují jako entita v datovém modelu, jako datová paměť (nebo část či souhrn více pamětí) ve funkčním modelu. Stejná data se dále vyskytují i v datových tocích procházejících systémem od vstupu (získání dat) přes jejich ukládáni v pamětech až po výstup. Stále však jde o téhož zákazníka.

Proto je nutné prověřovat konzistenci každého modelu uvnitř i mezi modely navzájem. Ověřený konzistentní model pak nazýváme vyváženým systémem. Ověření konzistence návrhu, jeho různých modelů, je tedy vlastně daní za použití strukturalizace popisu. Dobrý CASE systém by měl takovou konzistenci kontrolovat nebo ji pomáhat udržovat.

Pravidla vztahů mezi jednotlivými modely

1. DFD - DD

- každý datový tok a datová paměť musí být definovány v DD, jinak jsou považovány za nedefinované;
- každý datový tok a datová paměť definované v DD se musí objevit někde v DFD, jinak jde o "ducha", něco definovaného, ale nepoužitého.

2. DFD - minispecifikace

- každá bublina v DFD musí mít podřízený DFD diagram nebo minispecifikaci, ale ne obojí, jinak by byl model zbytečně i nebezpečně redundandní;
- každá minispecifikace musí mít elementární (trampující) proces v DFD ;
- musí souhlasit vstupy a výstupy; pro každý datový tok vstupující do elementární funkce v DFD musí mít odpovídající fci READ (nebo čti, get, accept ap.), pro každý výstupní datový tok odpovídající operaci WRITE (nebo pod.).

3. Minispecifikace - DFD a DD

každý odkaz na data (na podstatné jméno) v minispecifikaci je odkaz **buď**

- na datový tok nebo datovou paměť, na které se elementární funkce odvolává **nebo**
- je lokálním pojmem, explicitně definovaným v minispecifikaci **nebo**
- je součástí datového toku nebo paměti spojené s elementární funkcí; pak musí být vidět v DD.

4. DD - DFD a minispecifikace

- vše, co je definováno v DD, musí být použito v minispecifikaci, DFD nebo v jiné definici DD.

5. ERD - DFD a minispecifikace

- každá datová paměť v DFD musí korespondovat s entitou, vztahem nebo s kombinací entity a vztahu z ERD; pokud je v DFD použita datová paměť, která nesouvisí s datovým modelem, nebo je v ERD entita, která není součástí žádné paměti, něco není dobře;
- názvy entit a pamětí si musí odpovídat; např. zákazníci={zákazník}
- popisy v DD se musí odvolávat jak na DFD, tak na ERD;
- v minispecifikacích musí být pro každou entitu a vztah vyskytující se v ERD operace rušení a vytváření;
- každý atribut má v alespoň jedné elementární funkci nastavenou hodnotu a alespoň jedna elementární funkce atribut používá (čte).

6. DFD a STD

- každý řídicí proces v DFD má svůj STD, který tento proces specifikuje a naopak, každý STD patří jednomu řídicímu procesu z DFD;
- každá podmínka v STD musí korespondovat s jedním vstupním kontrolním tokem do daného řídicího procesu a naopak, každý vstupní řídicí tok do řídicího procesu musí mít podmínku v korespondujícím STD;
- každá akce v STD musí korespondovat s výstupním řídicím tokem z řídicího procesu, k němuž STD patří a naopak, každý výstupní řídicí tok z řídicího procesu musí souviset s příslušnou akcí korespondujícího STD.

2.6. Návrh uživatelského prostředí

❑ Komunikace s uživatelem

Uživatelský vzhled programu je součástí specifikace; jeho vhodný návrh je důležitý zvláště s rozvojem možností grafiky, použitím speciálních vstupních a výstupních zařízení, hlasových výstupů apod. Měl by být konzultován s uživatelem a případně s odborníky na psychologickou stránku komunikace, ergonomii apod. Základní pravidla pro návrh komunikace jsou jednak popsána v DAIS, jednak jsou probírána v předmětu Uživatelská prostředí.

Zopakujme si jen základní principy:

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, ...

❑ 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“)*.

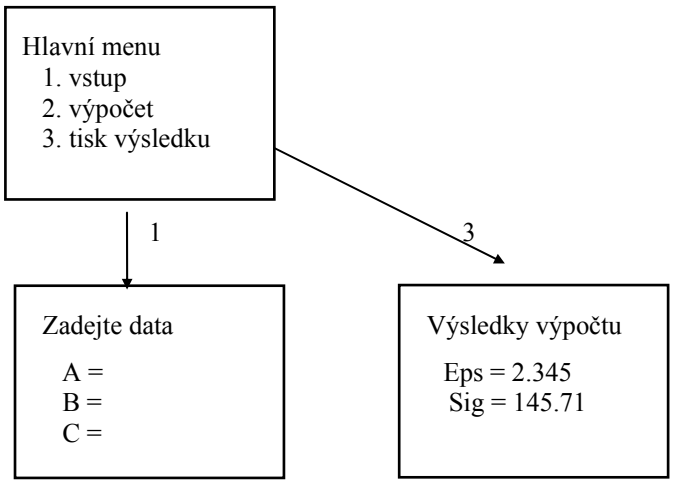
Analýza příčin chyb a principy správné reakce na ně

- 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.)*.

Diagram struktury komunikace

Alespoň pro základní kostru komunikace – základních typů obrazovek je vhodné zpracovat diagram struktury komunikace a prokonzultovat ho s uživatelem.

Diagram struktury komunikace slouží pro znázornění vzájemných volání a závislostí obrazovek a je nejčastěji používán při vytváření prototypu komunikace. Je to síťový diagram, kde uzly znázorňují jednotlivé obrazovky a hrany znamenají vyvolání obrazovky. Hrany jsou označeny způsobem volání obrazovky (stisk tlačítka, klávesy, zadání jisté hodnoty ap.)



Shrnutí pojmů 2.

Zadání informačního systému, funkční a nefunkční požadavky, modely vnějšího chování IS.
Kontextový diagram, seznam událostí a reakcí systému, model jednání.
Analýza a její druhy, modely IS.
Datová, funkční a dynamická analýza a nástroje pro jejich provádění.
Datový, funkční a časový model IS.
Komunikace programu s uživatelem. Pravidla pro ošetření chyb.

Otázky 2.

1. Jak se dělí požadavky pro zadání IS?
2. Co jsou funkční požadavky na IS a které informace obsahují?
3. Jaké formální modely se při formulaci zadání používají?
4. Co jsou nefunkční požadavky na IS a které informace obsahují?
5. Co je analýza obecně a jaké druhy analýzy se provádí při budování IS?
6. Co všechno obsahuje datová analýza a co tvoří její výsledek = datový model?
7. Co všechno obsahuje funkční analýza a co tvoří její výsledek = funkční model?
8. Co všechno obsahuje dynamická analýza a co tvoří její výsledek = časový model?
9. Co je návrh komunikace programu s uživatelem a jaké zásady by měly být dodrženy?
10. Jaká platí pravidla pro ošetření chyb v programu – uživatelských i jiných?

Příprava na tutoriál - Zadání semestrálního projektu

Zvolte si vlastní úlohu na vývoj informačního systému středního rozsahu.

Vytvořte pro IS 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.

Zpracujte úplnou analýzu zvoleného IS a navrhněte uživatelský vzhled aplikace.

Připravte prezentaci zadání a analýzy asi na 10 minut a prezentujte ji na cvičení nebo tutoriálu.

3. NÁVRH IMPLEMENTACE IS

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

❑ Cíle návrhu implementace

Výsledkem analýzy je několik modelů budoucího systému. Ty popisují **věcnou funkci systému**, tedy **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 následuje **model implementace,** upřesňuje se, **jak** se to vše bude dělat.

Model implementace může mít obecně jinou strukturu, než věcné modely (datový funkční a dynamický). Důvody jsou následující: implementační omezení nebo naopak rozšíření o různé protokoly, správu a údržbu databáze, návrh fyzické reprezentace dat, zpřesnění a optimalizace algoritmů, dělení systému až na separovatelné jednotky (programování ve velkém pro následné programování v malém). Separovatelným jednotkám říkáme moduly, jejich tvorbu popíšeme na konci etapy návrhu implementace.

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 = doplněná definice databáze, úplné algoritmy funkcí z minispecifikací, nové systémové funkce zabezpečující správu systému.

Dělení etapy návrhu

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

- **systémový návrh =** organizační, manažerská rozhodnutí o koncepci řešení, návrhu HW a SW prostředí, harmonogramu, ceně apod.
- **vlastní návrh implementace** = upřesnění, doplnění a optimalizace algoritmů, doplnění systémových dat a funkcí, rozdělení funkcí do modulů.

3.2. Systémový návrh

Úvodní koncepční část vychází z výsledků analýzy a z nefunkčních požadavků zadání.

Nejprve se řeší 2 **základní koncepční rozhodnutí** o realizaci systému:

1. koupit systém hotový nebo jej vytvořit
2. volba architektury systému

Dále se berou v úvahu všechny další nefunkční požadavky ze zadání.

❑ Koupit nebo vytvořit

1. Z analýzy je jasný rozsah systému a jeho dělení na funkční celky – subsystémy. Relativně málo systémů vzniká na zelené louce, úplně znova. Současnost je prorostlá informačními systémy a málokdy nalezneme oblast, kde je možno vytvářet systém nezávisle na čemkoliv.

Rozhodování o tvorbě IS lze popsat jako volbu mezi třemi alternativami:

- vytvořit vlastní systém,
- koupit hotový systém a přizpůsobit mu organizaci reality – upravit své požadavky,
- koupit hotový systém a přizpůsobit jej organizaci reality – upravit koupený systém.

Existuje mnoho teorií a rad z oblasti psychologie ap., které mají v této situaci poskytnout vodítko. Podíváme se na tento problém z druhého konce - z hlediska logiky problému a poskytnutých možností. Logický model systému - výsledek analýzy nabízí podklady pro koncepční rozhodnutí, jak systém realizovat.. Ve skutečnosti to je vždy trochu složitější, jednotlivé činnosti se nevylučují.

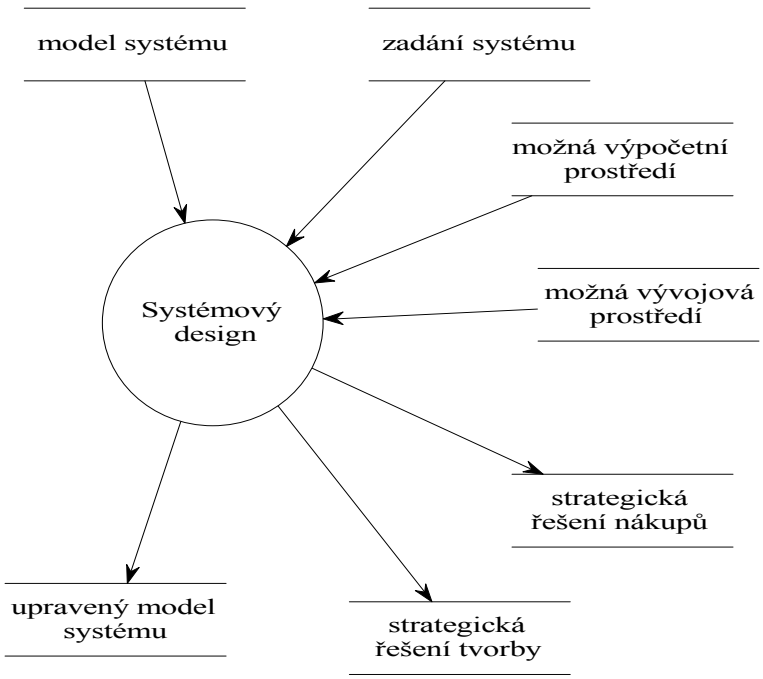
Koupě nového systému vždy znamená nějaké dopracování, ať již programové navázání na existující systémy nebo organizační řešení zapojení nového systému do reality. Přímá tvorba IS ve smyslu jednoduché posloupnosti *analýza - návrh - implementace* je možná jen u jednodušších úloh. U složitějších úloh je nutný **víceúrovňový návrh**, kdy návrh vyšší úrovně je vlastně *analýza - návrh - implementace* nižší úrovně.

U složitých a rozsáhlých úloh obvykle již v etapě analýzy bývá zřejmé, že problém nelze zvládnout na jeden zátah (*například podle velkého rozsahu databáze a funkcí nad ní*). Pak přejdeme na úroveň budoucích podsystémů a věnujeme hlavní pozornost soudržnosti jednotlivých podsystémů, jejich popisu a rozhraní. Tyto popisy budou tvořit zadání pro tvorbu podsystémů.

Provede se systémový návrh na úrovni subsystémů, to znamená méně podrobně. Zvlášť důležitý je krok, kde se rozhoduje o tom, zdali se subsystém vytvoří nebo nakoupí.

Pro jednotlivé podsystémy provedeme celý postup, rozborem zadání počínaje a implementací podsystému konče. Po vytvoření jednotlivých podsystémů je ještě etapa, jejímž úkolem je spojit podsystémy v systém.

Nejdůležitější datové toky a toky znalostí této části jsou uvedeny na obrázku.



Ve všech dalších úvahách budeme předpokládat, že jsme se rozhodli pro samostatnou tvorbu nového IS.

Architektura systému

2. Druhým koncepčním rozhodnutím je rozhodnutí o použité **architektuře IS** - rozložení dat databáze i instalovaného SW na jednotlivá média *(na jednotlivé servery nebo dokonce jednotlivé uzly distribuované sítě),*

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.

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.

4. 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.

❑ Zohlednění nefunkčních požadavků

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),*
- **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.

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 pro spojení funkcí a návrh rozložení funkcí do modulů;
- **transakční analýza**, doplnění minispecifikací o transakce nebo zamykání;
- **návrh rozmístění dat**, pokud jde o distribuovaný IS;

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 dat a 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. Tto 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.

♦

❑ Transakční analýza

Za transakci považujeme skupinu příkazů, měnících obsah databáze, které musí být provedeny všechny, aby nebyla porušena konzistence databáze (viz předmět DAIS). 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.

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).

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. Nejprve si uvedeme 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.

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

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. Před spuštěním transakce musí být k disposici všechny vstupy, pak se provede transakce a teprve pak jsou k disposici všechny výstupy. Během transakce nemá docházet k žádné interakci s vnějším světem;
  - 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;
  - příkazy, které jen čtou z databáze a nemění její obsah buď tvoří samostatnou transakci (to obvykle zabezpečí SŘBD automaticky), nebo nemusí být součástí transakce.
- 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.

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

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.

♦

Shrnutí transakční analýzy a realizace transakcí

Závěrem si 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.

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 nebo 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

RRRRnnnn

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 máme hotovu analýzu záloh a archivace, musíme vytvořit systémové funkce, které to vše 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 nebo modifikaci 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.

3.4. Návrh modulů a modulové schéma

Moduly a modularita systému

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 nových systémových funkcí (o kontroly stavů, zálohování, archivace, hlídání uživatelů a jejich přístupových práv atd.) a jsou doplněny některé systémové tabulky nebo atributy.

Následovat bude implementace, tedy jednak vytvoření databáze podle datového slovníku (příkazy CREATE TABLE), jednak programování všech popsaných funkcí. Než se předá výsledek návrhu programátorům, je vhodné rozdělit všechny popsané funkce (uživatelské i systémové) do jakýchsi skupin, „balíků“ funkcí, které spolu nějakým způsobem souvisí (jak, bude diskutováno níže).

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.

Závěrem návrhu se tedy 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.

Důležitým dalším důvodem, proč se předem navrhují 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 také pojem modularita programu:

Definice:

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 atd.,
- **dle datových toků** – funkce se stejnými přenosy dat tvoří modul.

Doporučuje se dodržovat následující **pravidla pro návrh modulů:**

- maximalizace soudržnosti modulů (zapouzdření dat do modulu)
- minimalizace souvztažnosti modulů (komunikaci mezi moduly)
- neomezování velikosti modulů (menší snad pohodlnější, větší levnější)
- sdružování centrálních dat v globálním modulu (jinak nutno předávat argumenty)
- omezení řetězu vnoření modulů dle principu 7±2
- eliminace nereferencovaných proměnných (jinak roste počet chyb)
- využívání modulů opakovaně (úspory a rychlejší odhalení chyb)

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.

Transakce, elementární 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.
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 konzistence databáze a jak může být porušena?
2. Jak může dojít při běhu aplikační úlohy ke ztrátě konzistence databáze?
3. Co je transakce a jak se zabezpečuje její atomická vlastnost?
4. Jakými metodami jsou v SŘBD podporovány transakce?
5. Jak SŘBD zabezpečuje databázi proti HW chybám?
6. Co je sériovost transakcí při víceuživatelském využívání databáze?
7. Proč se transakce při víceuživatelském provozu nespouštějí sériově za sebou?
8. Jakými metodami se zabezpečí sériovost transakcí?
9. Co je zamykání objektů databáze a k čemu se používá?
10. Jaké typy zámků objektů databáze existují?
11. Jak se zajišťuje sériovost transakcí zamykáním?
12. Co je uváznutí?
13. Jak se SŘBD s podporou transakcí brání proti uváznutí?
14. Jak se rozpozná uváznutí při běhu aplikací?
15. Co jsou plánovače transakcí a jaký je jejich princip?
16. Co je úkolem etapy návrhu implementace?
17. Jaké základní části návrhu rozeznáváme?
18. Jaké jsou základní typy architektury IS?
19. Co všechno patří k úkolům vlastního návrhu implementace?
20. Popište každý úkol návrhu implementace.
21. Co je modul a co je modulové schéma IS?
22. Podle jakých pravidel se seskupují funkce do modulů?
23. 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

4.1. Etapa implementace IS

Nejpozději v návrhu implementace bylo rozhodnuto o použitém prostředí pro implementaci - programovacím jazyce nebo systému řízení báze dat a architektuře informačního systému. 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. Zopakujme si SQL příkazy pro definici dat a řady deklarovatelných integritních omezení:

```
CREATE TABLE  Tabulka (
      ident1 {NUMBER|CHAR|DATE }(délka) PRIMARY KEY,
      ident2 … AUTO_INCREMENT,
      ident3 … NOT NULL,
      ident4 … DEFAULT i,
      ident5 … CHECK (ident5 IN (hod1, hod2, …)),
      ident6 … REFERENCES Tab2 [(ident)],
  ... )
```

Potom se přistoupí k vlastnímu kódování, u velkého IS většinou paralelně několika programátory. Každý má ke zpracování své moduly, které byly rozvrženy na konci etapy návrhu. Na základě doplněných minispecifikací uživatelských funkcí a dalších systémových funkcí se kódují ve zvoleném programovacím jazyce.

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, programovací techniky jsou náplní jiných předmětů.

Součástí etapy implementace je také vytvoření dokumentace k výsledným programům. Proto si zopakujeme ještě obsah dokumentace.

4.2. Dokumentace IS

Dokumentace budovaného IS postupně popisuje výsledné modely každé etapy životního cyklu. Jak víme, výsledný model jedné etapy je zadáním pro etapu následující. Je proto důležité popsat všechny podrobnosti, které mají na pokračování vliv. Obvykle se tedy vyskytují tyto druhy dokumentace:

❑ Dokumentace k zadání

Zpracovává se na začátku řešení, obsahuje globální popis 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, lineární zápis relací, úplný ERD s integritními omezeními + datový slovník + další integritní omezení.

Funkční analýza, obsahující hierarchii DFD + minispecifikace.

Dynamická analýza, obsahující stavové diagramy STD celého systému, jeho částí až po stavové diagramy důležitých entit. Místo STD entit je možno zpracovat životní cyklus entity ELH.

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.

❑ Dokumentace k implementovanému IS

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.

Následující doporučení o uživatelské dokumentaci sepsala před časem zkušená SW firma:

■ Jak by měly uživatelské příručky vypadat

Motto : V úvodu řekni,co chceš napsat, napiš to a na závěr řekni, že jsi to napsal.

1. Vzhled

Příjemný vzhled, pěkné grafické uspořádání, aby neodpudila předem; přehlednost, rozvržení informací na stránkách. Příručka často rozhoduje o prodejnosti programu!

2. Komu je určena

Mělo by být jasno, pro koho je určena, tomu odpovídat obsah i podoba; natolik úplná, aby pokrývala uživatelovu potřebu; nejlépe na obálce, uvnitř pak, za jakých okolností je vhodná, že jiné příručky obsahují to a to a jsou pro toho a toho; jediná příručka pro všechny připomíná dort pejska a kočičky; pozitivní motivace - autor má mít jasno, jak podat informace, aby v uživateli vzbudil zájem a získal pro práci s programem, nevzbuzovat pocit méněcennosti; konečně popsat vše, co bude tento uživatel potřebovat a tak přehledně uspořádat, aby to uživatel mohl znovu snadno najít; tedy vyzkoušet na několika uživatelích !

3. Logická struktura

Dobře logicky uspořádaná a přehledně členěná; potřebné informace snadno dostupné - volba názvů kapitol a odstavců, podrobný rejstřík. Pokud je v okruhu uživatelů, pro něž je příručka určena, zavedena nějaká terminologie a způsob výkladu, přizpůsobit tomu formu. Osnova ! zvolit názvy a obsah kapitol a nižších celků; při změnách ve struktuře nejprve zařadit nové části do staré struktury a pojmenovat je, pak rozmyslet globální vazby související se změnami, udržet přehlednou strukturu.

4. Styl

Příručka má mít dobrou kvalitu:
- zvolit základní terminologii, slovník základních pojmů, vysvětlit je uživateli a důsledně používat; pojmy také v rejstříku, případně slovník pojmů samostatný; nezavádět vlastní termíny, pokud existují již ustálené;
- podávat informace přesně a přehledně;
- psát jednoduchým, jasným, čtivým stylem.

5. Příklady

Příklady a obrázky hojné, praktické, názorné a poučné. Nejlépe poskytovat uživateli představu o použití programu, jeho možnostech, metodiku používání v praxi. Příklady jednoduché a správné, programy odladěné. Rozsáhlé výpisy jako příloha, aby nerozbíjely celistvost výkladu. Příklady rozsáhlé se obvykle nečtou, jen při výuce v nějakém kurzu.

6. Příprava k tisku projít recenzí, praktickým otestováním, redakční a jazykovou úpravou.

7. Profesionální postup při tvorbě příručky

- počáteční fáze mapování terénu, získání přesných a úplných informací o programovém systému (obvykle píše autor) a o typu uživatelů, kterým bude určena;
- přípravná fáze, plán rukopisu :
  Typ příručky - učebnice, systematický popis systému ?
  Styl, forma - popis, průvodce, kuchařka, scénář, odpovědi na předpokládané dotazy, kombinace ?
  Vzhled - uspořádání stránky.
- vlastní psaní připomíná strukturované programování, mělo by být shora dolů: nejprve základní pojmy, pak základní členění; v úvodu informace o používání programu - koupě, závazky prodejce a uživatele, porovnání s předchozí verzí nebo obdobnými programovými produkty na trhu, vysvětlení způsobu uspořádání a výkladu.
  Popis instalace. Uvedení ilustračního příkladu.
- po napsání vlastní vydání; má-li program úspěch, pak zahájení práce na nové verzi příručky.

Programátorská dokumentace

Jak je známo, každý program je nutné průběžně měnit, odstraňovat chyby, doplňovat o nové požadavky, upravovat a dolaďovat existující funkce i data. 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 především podrobné komentáře ve zdrojových kódech programu a v příručce dále 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.

Shrnutí pojmů 4.

Implementace IS a její dokumentace.
Druhy dokumentace, manuál a programátorská příručka.

Otázky 4.

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?

Příprava na tutoriál - Zadání semestrálního projektu

Připravte prezentaci implementace svého informačního systému.

Implementovanou vlastní úlohu informačního systému předejte podle pokynů svému cvičícímu nebo tutorovi současně s úplnou dokumentací systému.

5. TESTOVÁNÍ 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

Současně s implementací se začíná provádět kontrola správnosti výsledného programu. Kvalita IS je dána tím, do jaké míry splňuje kritéria správnosti a spolehlivosti, neboli kolik vážných chyb se může objevit během jeho provozování. Z psychologického hlediska je testování destruktivní činnost, protože nikdo nemá chuť odhalovat své omyly a nedokonalosti a testování k jejich odhalení má směřovat. Ve skutečnosti návrh dobrých a úplných testů může být stejně zajímavý jako počáteční návrh softwaru.

❑ Spolehlivost programu a chyby

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.

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..

❑ Druhy testování

Účelem testování je především zjistit, zda programový systém splňuje stanovené požadavky, tj. vyhovuje systémové specifikaci, za druhé odhalit co nejvíce chyb. Zkušenost ukazuje, že vytvoření úplně bezchybného programového systému je obecně nemožné.

Chybou rozumíme odchylnost od chování předepsaného programovou specifikací. Příčina chyby může spočívat ve **specifikaci, návrhu** nebo v **implementaci**. Pokud je program dostatečně robustní (odolný proti chybám uživatelů), nemůže být příčina chyby ve vstupních údajích; ty by měly být robustním programem odhaleny.

Kontrolu správnosti programu tedy 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 korektnost 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.

S testováním souvisí ladění (debugging). Úkolem testování je odhalit chybu, úkolem ladění je lokalizovat a odstranit chybu zjištěnou testováním.

V následujícím přehledu uvedeme některé techniky testování. Základní snahou je najít dostatečnou sadu testů, která by mohla mít naději na odhalení chyby.

❑ Úrovně testování

Metody testování se liší podle toho, **co** se má testovat: výsledek analýzy, jednotlivé moduly, interakce mezi moduly, integrace modulů do celého systému nebo akceptovatelnost celého softwarového produktu.

Testování systémové specifikace

Cílem testování systémové specifikace je ověřit úplnost, srozumitelnost, konzistenci a přípustnost systémové specifikace. Pro tento typ testování byla navržena technika „příčiny a účinku“. Příčinami se rozumí v této souvislosti **podmínky** nebo **kombinace podmínek,** které jsou dány nebo se mohou vyskytnout, účinky jsou **výsledky** nebo **aktivity**. Test specifikace by měl ukázat, zda neexistují příčiny (data nebo funkce), pro které nebyly definovány účinky, anebo naopak nevznikají-li účinky, aniž by byly dány jejich příčiny. Test specifikace musí být prováděn společně s uživateli celého systému nebo jeho části.

Testování modulů

Moduly zapouzdřují datové struktury a funkce, které na nich operují. Účelem testování modulů je odhalit rozpory mezi implementací a specifikací modulu. Nejdříve jsou testovány jednotlivé funkce, potom jejich interakce. Samotný modul nemůže být testován, protože obecně nemá exekutivní formu. Jeho provádění většinou předpokládá existenci jiných modulů. Testování lze proto uskutečnit pouze v simulovaném testovacím prostředí. Vhodné testovací prostředí umožňuje volání funkcí modulu, kontrolu výsledků zpracování a simulování vlivů jiných modulů. Testovací prostředí musí být co nejjednodušší, aby pravděpodobnost, že samo obsahuje chyby, byla co nejmenší.

Testování interakcí mezi moduly

Po testování modulů může následovat testování subsystémů. Subsystémem rozumíme funkční kombinaci jednotlivých (dříve individuálně protestovaných) modulů. Účelem je otestovat interakce mezi moduly subsystému, především správnost komunikace mezi moduly. I zde musí být vytvořeno vhodné testovací prostředí. Subsystémy jsou dále spojovány a testovány ve větších celcích. Tato forma testování se nazývá **hierarchické testování** nebo **integrační test**.

Testování celého systému

Testuje se integrace všech subsystémů. Cílem je objevit a odstranit všechny odchylky mezi chováním systému a chováním předepsaným systémovou specifikací.

Je třeba ověřit nejenom úplnost specifikace a správnost výsledků, ale i robustnost systému vzhledem ke vstupním údajům. V této fázi musí být testovány i nefunkcionální požadavky (např. požadovaná výkonnost). Rozsah i počet testů závisí na druhu aplikace.

Předávací test - akceptovatelnost systému

Předávacím testem končí testování celého systému. Systém je testován na reálných datech a za provozních podmínek. Účelem testu je odhalení všech chyb, které pramení například z nedorozumění v konverzaci mezi uživateli a dodavatelem softwarového produktu, z nesprávných odhadů objemů dat, nebo z nerealistických předpokladů o skutečném okolí programového systému.

Teprve při tomto testu se ukazuje, zda aktuální aplikace koresponduje s definicí požadavků a zda chování systému vyhovuje očekávání uživatele.

❑ Metody testování

Statické testování

Statické testování spočívá v hledání chyb, které mohou být nalezeny bez provádění testovaného objektu. Testování se vztahuje k **syntaktické, strukturální a sémantické** analýze testovaného objektu s cílem lokalizovat místa náchylná k chybám. Testovací aktivity:

- **Verifikace programu** se týká kontroly správnosti jednak návrhu programu a jednak konkrétní implementace návrhu formálními metodami.V praxi se formální verifikace aplikuje většinou pouze na návrh.
- **Inspekce kódu** je další vhodnou technikou pro nalezení chyb v návrhu a implementaci. Je známo, že řadu chyb lze objevit pouze dostatečně pozorným čtením textu programu, k čemuž však programátor není motivován. Doporučuje se projít kód (s cílem najít, nikoliv opravit chyby) týmem pracovníků tvořeným:

  zkušeným softwarovým inženýrem nepracujícím na projektu,
  projektantem testovaného objektu,
  programátorem zodpovědným za implementaci,
  specialistou na testování.

- **Analýza složitosti** zjišťuje stupeň složitosti programu. Získané informace umožňují predikci týkající se kvality softwaru a určení pravděpodobnosti výskytu chyby v jednotlivých částech programového vybavení. Sleduje se:

  míra složitosti modulů
  hloubka vnoření cyklů
  délka procedur a modulů
  importní a exportní čísla modulů
  složitost rozhraní procedur

- **Analýza struktury** slouží k odhalení strukturálních anomálií v testovaném objektu. Měla by například poskytnout informaci, zda všechny příkazy programu jsou dosažitelné od místa začátku programu a zda konec programu je dosažitelný z každého příkazu, zda program obsahuje cykly s vícenásobnými vstupy nebo nepovolené řídící struktury.
- **Analýza toku dat** by měla odhalit anomálie v toku dat. Například, zda je datovému objektu přiřazena hodnota dříve než je použit a je-li datový objekt použit poté, co jsou mu přiřazena data. Analýza by měla být provedena jak v těle testovaného objektu, tak pro rozhraní mezi objekty.

Analýza složitosti, struktury a datových toků vyžadují v případě složitých nebo velkých programových systému nástroje pro podporu této analýzy (automaticky udržované seznamy modulů, seznamy

procedur s příslušností k modulu, křížové referenční seznamy objektů importovaných a exportovaných, grafy volání, importní grafy apod.).

Dynamické testování

Při dynamickém testování jsou objekty prováděny nebo je jejich činnost simulována. Dynamické testování zahrnuje tyto činnosti:

- příprava testovaných objektů pro lokalizaci chyb (instalace ladicích prostředků)
- vytvoření testovacího prostředí
- výběr vhodných testovacích běhů
- provedení a vyhodnocení testů.

Každá funkce, modul, subsystém i celý systém musí projít dynamickým testováním i v případě úspěšného statického testování.

Testování metodou černé a bílé skříňky

V podstatě lze každý objekt testovat dvěma způsoby:

1. kontrolou vztahu mezi výstupem a vstupem bez ohledu na vnitřní strukturu objektu (test rozhraní), nazývá se také „black box“, „funkcionální“ nebo „exteriorní“ test;

2. totéž s ohledem na vnitřní strukturu, nazývá se „white box“, „strukturální“, nebo „interiorní“ test.

Black box - testuje chování rozhraní objektu během jeho provádění. Výběr testovacích běhů je založen pouze na specifikaci testovaného objektu bez ohledu na jeho vnitřní strukturu. Sledují se rovněž reakce na mezní nebo nesprávná vstupní data. Toto testování se nezabývá funkcemi nebo datovými objekty, které přímo neovlivňují vstup a výstup. Tato metoda je vhodná pro integrační testy.

White box - cílem je prověřit všechny možné průchody testovanou částí programu, používá se při testování jednotlivých modulů.

Testovací běhy se vybírají s ohledem na strukturu dat nebo na vnitřní strukturu programu.

Testování podle dat

Předpokládáme, že vstupní data lze rozdělit do tříd takových, že pro každou třídu stačí navrhnout jeden test (ostatní data skupiny by dávala stejný výsledek).

Příklad: Ve specifikaci je dáno, že data jsou celá čísla z intervalu $< 1, 999 >$. Pak lze definovat 4 třídy testovacích dat: necelá čísla, celá čísla menší než 1, celá čísla z intervalu 1 až 999, celá čísla větší než 999.

Obdobně se dají obvykle rozdělit i výstupní data.

Dalším krokem je nalezení vhodných reprezentantů těchto tříd, kteří by tvořili testovací data.

Testování podle struktury programu

Vycházíme z řídicí struktury programu, vykreslené orientovaným grafem, kde uzly jsou rozhodovací bloky, hrany jsou přechody mezi nimi. Testování podle struktury programu znamená vybrat takovou sadu testů, které by prověřily všechny průchody všemi větvemi programu - alespoň jednou, ideálně pro každou třídu testovacích dat. Provedou se testy tak, aby

každý modul a funkce testovaného objektu jsou provedeny alespoň jednou
každá větev je provedena alespoň jednou
provede se tolik řídicích cest programem, kolik je možné

Je důležité projít každou větev programu. Nestačí pouze zajistit, že je proveden každý příkaz.

Na první pohled se zdá, že ke stoprocentně správnému programu dospějeme, protestujeme-li všechny řídící cesty. To je ale často prakticky neproveditelné pro jejich ohromné množství.

Testování kompletního systému

Testování shora dolů a zdola nahoru

Předchozí metody jsou vhodné pro testování modulů a jejich integrace. Při testování kompletního programového systému můžeme postupovat zdola nahoru nebo shora dolů.

> **Testování způsobem shora dolů** se provádí rovnou při implementaci. Nejdříve implementujeme hlavní modul, a testujeme ho tak, že nahradíme importované části fiktivními objekty, které mají stejné rozhraní jako objekty skutečné a simulují jejich chování. Pokračujeme stejně u dalších modulů. Protože implementace a testování probíhají současně, odpadají integrační testy.
>
> Výhodou tohoto postupu je, že objevíme chyby v návrhu v nejkratším možném čase. Tím šetříme čas a cenu na vývoj, protože opravy v návrhu lze udělat předtím, než je implementován.
>
> Nevýhodou je, že je někdy velmi obtížné vytvořit užitečné objekty simulující chování reálných objektů.
>
> Zatímco program navrhujeme shora, **testování provádíme zdola**, tj. nejdříve testujeme funkce nejmenších částí systému a potom jejich integraci do modulu. Potom testujeme modul a dále integraci několika (už otestovaných) modulů do subsystému. Nakonec je testován celý systém.
>
> Metoda zdola nahoru se zdá spolehlivá. Testované objekty jsou známé do všech detailů, takže je snazší určit významné testovací běhy.
>
> Nevýhodou je, že testování probíhá až po ukončení implementace, takže chyba nalezená ve vyšší úrovni návrhu si vynucuje změny v nižších úrovních a opakování testovacích činností.

Nelze říct, která z metod je lepší. V praxi se opět obě metody kombinují.

Návrh testů a plánování testování

Testování programových systémů je složitá a nákladná činnost zaměstnávající mnoho osob pracujících na projektu, a proto je třeba ji plánovat, pokud možno už v době tvorby systémové specifikace. Testování vyžaduje přípravu testovaných objektů a testovacích běhů.

Objekty na nižší úrovni (moduly) jsou testovány samotnými programátory, zatímco integrační testy by měly provádět osoby nezainteresované na implementaci.

Plánování testování

Plánování testování znamená určit:

- druh testovací strategie, jež má být aplikována
- druh dokumentace testování
- testované objekty a požadavky kvality na ně kladené
- testovací kritéria, která mají být splněna (např. počet testovacích běhů, procento otestovaných cest v programu)
- za jakých podmínek je každý individuálně testovaný objekt akceptován
- angažování testovacích specialistů
- použité testovací nástroje

Příprava testovacích objektů na lokalizaci chyb

Při dynamickém testování mají být odhaleny chyby, které se nepodařilo odkrýt při statickém testování. Výskyt chyby však neříká ještě nic o její příčině. Proto je třeba připravit testované objekty na lokalizaci chyb. Neexistuje univerzální metoda, ale používají se techniky:

- trasování toku řízení
- trasování stavových hodnot, tj. hodnot důležitých proměnných reprezentujících stav programu
- monitorování stavových hodnot (hodnoty se sledují na překročení mezí, nesplnění nutných podmínek v jistých bodech programu apod.)

Využívá se ladicích prostředků příp. podmíněného překladu.

❑ Výběr testovacích běhů

Vyčerpávající testování je obecně neproveditelné i pro jednoduché objekty z důvodu potřeby velkého množství kombinací různých vstupních hodnot.

Pojem **testovací běh** znamená vstupní data testovaného objektu (funkce) a očekávaný výsledek. Správný výběr testovacích běhů vyžaduje zkušenost a intuici, nelze poskytnout obecný návod, v následujícím textu je pouze několik doporučení.

Testování programových jednotek

Testovací běhy musí být zvoleny tak, aby:

- se ověřilo, že všechny funkční jednotky a jejich rozhraní splňují specifikaci
- se prošlo všemi větvemi
- se použily mezní hodnoty vstupních dat
- každý cyklus byl otestován s minimální i maximální hodnotou opakování
- byly vyzkoušeny všechny chybové výstupy
- byla zkontrolována reakce testovaného objektu na náhodná vstupní data

Testování integrace programových jednotek

Při integračním testu několika programových jednotek do systému předpokládáme, že všechny tyto jednotky byly testovány a že je můžeme považovat za správné.

Subsystém, který má být testován je souhrn černých skříněk (programové jednotky) a testovací běhy jsou vybrány tak, aby se prověřila každá funkce subsystému aspoň jednou. Přitom jeden testovací běh může testovat více funkcí.

Při výběru testovacích běhů se uvažují jak kritéria orientovaná na úlohu, tj. kritéria závislá na specifikaci testovaného objektu bez ohledu na programový text, tak kritéria orientovaná na strukturu programu, tj. kritéria sledující programový text.

❑ Testovací prostředí

Testovací prostředí simuluje skutečné operační podmínky testovaného objektu. Musí umožnit:

- opakované provádění testovaného objektu
- přípravu vstupních dat
- simulaci nutných, ale nedostupných zdrojů (importované moduly) a jejich výsledků
- výstup, tisk výsledků testovacího běhu

Testovací prostředí musí být co nejjednodušší, aby pravděpodobnost, že samo obsahuje chyby, byla co nejmenší, avšak musí umožnit provedení požadovaných testů.

Vyhodnocení testu a lokalizace chyb

Po provedení testů se porovnávají výsledky testovacích běhů s očekávanými výsledky a zkontroluje se, zda všechny funkce uvedené v popisu testovacího běhu byly skutečně provedeny.

Je-li zjištěna chyba, nevíme zatím nic o jejím zdroji. Nejjednodušší lokalizace chyby je trasování testovacího běhu. Lokalizace chyby je mnohem jednodušší, je-li testovaný objekt konstruován modulárně s jednoznačně definovaným rozhraním.

Před vlastní opravou chyby je třeba prozkoumat vlivy opravy na okolí opravovaného objektu, tj. prostudovat nejen implementaci opravovaného objektu, ale také jeho abstraktní návrh, aby nebyly zavlečeny nové (někdy mnohem horší) chyby. Pravděpodobnost, že oprava chyby na první pokus je správná, je 50% (Pomberger, 1986).

Po opravě chyby je nutno opakovat příslušné testy.

Typické chyby

Z praxe je vidět, že jisté chyby se dělají opakovaně. Znalost zdrojů takovýchto chyb může velmi usnadnit jejich lokalizaci a opravu.

Některé typické chyby:

- ignorování speciálních případů (např. dělení nulou)
- překročení mezí polí
- nekonečný cyklus
- neinicializování proměnných
- nesprávný logický výraz v podmínce (zejména v souvislosti s operátorem negace)
- neodpovídající počet parametrů a typ skutečného a formálního parametru (řeší překladače jazyků s typovou kontrolou a oddělenou kompilací)
- neoprávněný přístup ke společným datovým oblastem (této chybě lze předejít důsledným použitím datových pouzder)

❑ Dokumentace testů

Pečlivá dokumentace testů umožňuje ekonomický průběh testování a kontrolu kvality, usnadňuje fázi údržby a je výchozím materiálem pro statistiku týkající se zlepšení plánování budoucích projektů.

Dokumentace testů by měla obsahovat:

- plán testování
- specifikace testovacích běhů
- popis použitého testovacího prostředí
- výsledky jednotlivých testovacích běhů
- počet a typ detekovaných chyb
- popis opravy chyby
- informace týkající se počtu požadovaných testovacích běhů pro každý testovaný objekt

Předávací test a prováděcí test

Předmětem předávacího testu je ukázat zákazníkovi, že zkompletovaný softwarový produkt splňuje požadavky systémové specifikace. Během předávacího testu je chování produktu testováno z hlediska uživatele a v reálném prostředí. Obsahuje tedy pouze testovací běhy vyplývající ze systémové specifikace (nikoli z programového textu) a formulované uživatelem, který se nejlépe orientuje v oblasti aplikace. Součástí předávacího testu je také kontrola uživatelské příručky na rozdíly v popsaném a skutečném ovládání produktu.

V případě, kdy software je určen větší skupině uživatelů (textový editor, kompilátor apod.), je nepraktické provádět předávací test s každým z nich, ale používá se technika alfa a beta testování. Alfa test je vyzkoušení softwaru v přirozeném nastavení v místě jeho vývoje, např. přímo členy projektového týmu nebo jejich kolegy. Beta test je „živé“ použití softwaru v prostředí jednoho nebo více eventuálních zákazníků. Zákazníci zaznamenávají všechny problémy (skutečné i domnělé), se kterými se setkají a v pravidelných intervalech je oznamují vývojovým pracovníkům, kteří provádějí odpovídající modifikace a připravují software k vypuštění do celé uživatelské základny.

Předmětem prováděcího testu je zkontrolovat nefunkční požadavky na softwarový produkt (reakční časy dialogů apod.) a spolehlivost (stabilitu) softwarového produktu, tj. jeho chování v delším časovém úseku.

Průběhy obou testů je vhodné zaznamenat do dokumentace testů.

❑ Hypotéza o chybě, problém ladění

Motto: Každý větší program obsahuje alespoň jednu chybu.

Je známo, že testování programů patří k nejobtížnějším problémům v programování. Testování musí být součástí návrhu projektu nástroje testování (testové soubory, simulační programy ap.) tvoří i 30% rozsahu výsledného produktu. Přesto zůstává ve výsledném programu tolik chyb, že údržba systému spolkne více než 60% celkových nákladů. Odstranění chyby v provozu je 50-krát dražší, než v etapě návrhu programu. Počet chyb, které proniknou testy je srovnatelný s počtem chyb, které testy odhalí. Specifikace cílů, návrh architektury a psaní programů se podílejí na chybách přibližně stejným dílem. To vše znamená, že nejsme mentálně schopni zachytit mnoho myšlenkových chyb v počátečních etapách prací.

Výsledky výzkumů udávají, že

- v době předání programů máme před sebou ještě více než polovinu prací,
- lidský mozek je schopen průběžně sledovat souvislosti ne více než 5 objektů,
- programy kratší než N = 280 mají dobrou šanci být bez chyb => je vhodné psát podprogramy i pro případ, že budou volány pouze jednou,
- nejúčinnější metodou ladění je čtení kódu,odhalující při správné organizaci až 80% vzniklých v etapách návrhu a psaní programů:

  1. Skupina odborníků (mezi nimi může a nemusí být autor a nemá být vedoucí autora programu) pročítá napsaný program a hledá chyby.
  2. Na jedno sezení se nemá číst více než 100-150 řádků programu (1-2 hod práce).
  3. Členové skupiny musí být motivováni k hledání chyb (např. budoucí uživatelé laděných programů).

Shrnutí pojmů 5.

Spolehlivost programu, chyby v programu.
Druhy testování, úrovně testování, metody testování.
Návrh testů, dokumentace testování.
Hypotéza o chybě.

Otázky 5.

1. Co je úkolem testování IS, co je pro něj zadáním a co výsledkem?
2. Jak definujeme spolehlivost programu?
3. Co je úkolem testování hotového IS a které typy, úrovně a metody testování rozlišujeme?
4. Co má obsahovat dokumentace testů?

6. 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

6.1. 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.

♦

6.2. 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.

6.3. Stupně vyspělosti SW firem dle úrovně SW procesu

Dlouholeté zkušenosti i cílený výzkum zaměřený na vývoj softwaru prokázaly, že pro kvalitu vývoje SW mají vliv lidské zdroje, technologie vývoje SW a tzv. softwarový proces. Ten je definován následovně:

Definice:

Softwarový proces je množina po částech uspořádaných procesních kroků, které směřují k vytváření a udržování softwarových produktů.

Pojem **procesní krok** je buď elementární procesní krok, tedy aktivita, která již nemá svou viditelnou vnitřní strukturu, nebo je to **subproces,** se všemi výše definovanými atributy a určený k plnění některého z dílčích cílů. Provádění těchto procesních kroků realizují **lidé a počítače,** tvořící součást procesu.

Pro pochopení uvedených pojmů je nutné si uvědomit rozdíl mezi softwarovým procesem a projektem. SW proces reprezentuje obecný předpis, abstrakci, jak řešit jednotlivé projekty. Ty pak lze chápat jako instance této abstrakce.

Podle úrovně definování a využití SW procesu je možno hodnotit vyspělost SW firmy. Následující model hodnocení se nazývá Capability Maturity Model a jeho jednotlivé úrovně lze stručně charakterizovat takto:

1. **počáteční** úroveň (Initial) - firma nemá definován softwarový proces a každý projekt je řešen případ od případu (ad hoc).
2. **opakovatelná** úroveň (Repetable) - firma identifikovala v jednotlivých projektech opakovatelné postupy a tyto je schopna reprodukovat v každém novém projektu.
3. **definovaná** úroveň (Defined) - firma má softwarový proces definován a dokumentován na základě integrace dříve identifikovaných opakovatelných kroků.
4. **řízená** úroveň (Managed) - na základě definovaného softwarového procesu je firma schopna jeho řízení a monitorování.
5. **optimalizovaná** úroveň (Optimalized) - zpětnovazební informace získaná dlouhodobým procesem monitorování softwarového procesu je využita ve prospěch jeho optimalizace.

Bezpochyby firma ve vyšších úrovních se stává konkurencechopnější a skýtá vyšší záruky úspěšného řešení zadaného projektu.

Shrnutí pojmů 6.

Předání informačního systému uživateli, zaškolení, dokumentace.
Konverze starých dat do nového IS.
Druhy údržby IS.
Vyspělost SW firem dle úrovně SW procesu.

Otázky 6.

1. Co všechno zahrnuje etapa předávání IS do provozu?
2. Co je nutné naplánovat, připravit a realizovat před vlastním předáním do provozu?
3. Co všech zahrnuje údržba IS a jaké typy údržby rozlišujeme?
4. Které úrovně vyspělosti SW firem rozlišujeme?

Příprava na tutoriál - Zadání semestrálního projektu

Rozplánujte si otestování svého IS, proveďte testy na potřebných úrovních, zpracujte dokumentaci k testům.

Připravte prezentaci svého informačního systému.

Implementovanou vlastní úlohu informačního systému předejte podle pokynů svému cvičícímu nebo tutorovi současně s úplnou dokumentací systému.

7. METODOLOGIE A CASE SYSTÉMY

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

7.1. Vývoj metodologií

Představme si, že každou etapu životního cyklu reprezentuje jiný člověk, postupně jsme je nazvali:
Zadavatel – **A**nalytik – **N**ávrhář – **P**rogramátor – **D**okumentátor – **T**estér – **Š**kolitel – **U**živatel,
všechny práce řídí manažer.

Dobře provedená jedna etapa se pozná tak, že formulovaným výsledkům rozumí zadavatel, souhlasí s nimi a jsou přesným a úplným zadáním pro řešitele následující etapy - i v případě, že ten se předcházející etapy nezúčastnil.

Formulování pravidel pro řešení velkých systémů si vynutila praxe a jednotlivé nástroje vznikaly postupně na základě zkušeností. Bez definovaných pravidel se vývoj systémů výrazně prodlužuje a prodražuje, přičemž jejich kvalita bývá nižší.

Zopakujme si, co jsme již uvedli výše:

Metoda je technika (technologický postup) pro podporu některé části životního cyklu; metody používají různé **nástroje** - textové nebo grafické editory, formuláře ap. k popisu nebo grafickému zobrazení jednotlivých modelů pomocí použité metody.

Metodologie je ucelená kolekce metod, pokrývající celý životní cyklus nebo jeho převážnou část.

Velké IS jsou vyvíjeny většinou většími firmami v týmech. Proto první metodologie byly formulovány právě u velkých softwarových firem (IBM). Některé metodologie jsou sice známé pod jménem svého autora, ovšem často je autor právě pracovníkem velké firmy.

Metodologií je celá řada a obvykle o poslední z nich se nejvíce mluví. Většinou však používají již dříve známých a osvědčených nástrojů, vylepšené a doplněné novými. Podstatné pro kvalitu metodologie je, aby práci celému týmu ulehčila, formalizovala formalizovatelné, zavedla organizaci a jasná pravidla při formulování výsledků všech etap životního cyklu, pokrývala všechny podstatné dimenze systému a tím vším urychlila a zkvalitnila vývoj systému.

Cílem dobré metodologie je pak automatizace všech automatizovatelných etap cyklu.

V následujících odstavcích naznačíme vývoj metodologií na několika nejznámějších metodologiích, i když jsou někdy nazývány metodami.

Metoda 4 modelů systému

Snad nejstarší metodologie, která je základem všech následujících.

1. Analytik nejprve zmapuje a popíše **současný fyzický, reálný systém** (ať "ručně" prováděný nebo automatizovaný),
2. Převede jej na **logický model stávajícího systému**, přitom zformuluje podstatu transformací dat a výstižně pojmenuje logické procesy.
3. Po konzultaci s uživatelem (dle jeho požadavků a dle vlastních návrhů na zkvalitnění systému) promítne do logického modelu změny a vytvoří **logický model nového systému**.
4. K němu je navržena vhodná implementace a vzniká **fyzický model nového systému**.

Přitom však nastává řada problémů :

- uživatel zná původní fyzický model mnohem lépe, než analytik a uživatel - platící zákazník má dojem že se analytik za jeho peníze problematiku teprve učí;
- uživatel odmítá spolupráci na vývoji nového systému; má pocit, že když analytik neumí sám vytvořit fyzický model stávajícího systému, nemůže dobře a včas ani navrhnout nový systém;
- někteří uživatelé i programátoři považují analýzu za oddechový čas, než začne "skutečná" práce, kódování a ladění.

Zkušené oko ve 4 modelech rozpoznává základ dnešní metodologie: první model odpovídá zadání, z něj popis současného stavu; druhý model ten první schematicky zakreslí; třetí model je výsledkem analýzy a návrhu implementace, čtvrtý hotovou implementací. Základní princip je dodržován stále, ale důležitost a podrobnost jednotlivých modelů se postupně výrazně změnila.

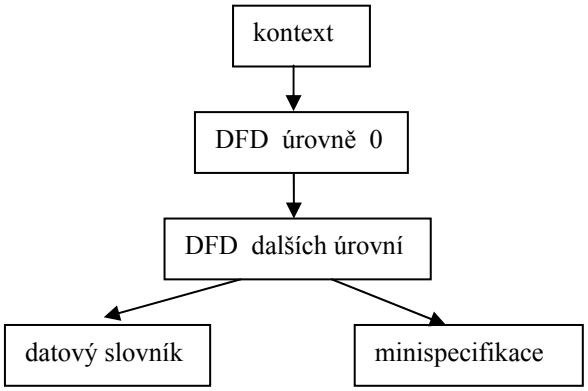
Metoda strukturované analýzy DeMarcova

De Marco poprvé použil DFD pro analýzy IS určených pro obchodní a správní organizace. Pomocí nich formuloval strukturovanou analýzu.

Na základě metody 4 modelů je strukturovaná analýza rozdělena do 7 procesů, cílem je vytvořit dobrou a přehlednou dokumentaci:

1. Studie stávajícího fyzického prostředí. Výsledkem je dokumentace stávajícího stavu tvořená **fyzickými DFD**.
2. Odvození logického ekvivalentu fyzického systému. Výsledkem je **logický DFD stávajícího** systému.
3. Odvození nového logického systému. Výsledkem je sada **nových logických DFD** navrženého systému, popis funkcí a datový slovník.
4. Odvození fyzických charakteristik nového systému, jejich zachycení v sadě **nových fyzických DFD**.
5. Kvantifikace cen a termínů na základě alternativ uvedených v DFD z bodu 4.
6. Výběr jedné z možností, volba platné sady nových fyzických DFD.
7. Začlenění nových fyzických DFD do **strukturované specifikace** společně s **datovým slovníkem** a **popisy procesů**.

Výsledná dokumentace má strukturu a návaznosti:



Součástí De Marcovy metody jsou doporučení týkající se

- volby jmen procesů, toků, pamětí ap.
- pravidel pro sestavení a dekompozici DFD
- pravidel pro sestavení datového slovníku
- přechodu od specifikace k implementaci

Výsledná dokumentace má vlastnosti:

- logický model je nutno vytvořit proto, aby se uživatel mohl seznámit se systémem před implementací;
- produkty analýzy musí být dobře udržovatelné, především cílový dokument;
- velké problémy musí být rozděleny na menší; cílový dokument ve formě románu je neúnosný;
- kde je to možné, používat grafické vyjádření, obrázky;
- je třeba rozlišit logické a fyzické aspekty a dle toho rozdělit zodpovědnost mezi uživatele a analytika.

❑ Metoda Yourdonova - moderní strukturovaná analýza (YSM)

Formulovaná E.Yourdonem roku 1989 jako výsledek dvacetiletých zkušeností s vývojem velkých systémů a výukou metod strukturované analýzy. Hledá tzv. **esenciální model** systému, vyjadřující podstatu systému a nezávisí na technických a jiných implementačních omezeních.

Model se skládá ze dvou částí:

- **model okolí systému** (enviromental model), který popisuje hranice mezi systémem a okolím, patří sem kontextový diagram, seznam událostí a stručný popis účelu celého systému.
- **model chování systému** (behavioral model), popisující požadované chování uvnitř systému, kterým má zajišťovat správnou interakci s okolím. K popisům se používají **ERD, DFD, datový slovník a STD.**

Modeluje, co má systém dělat, aby splnil požadavky a potřeby uživatelů bez ohledu na vnější okolnosti, nefunkční požadavky, jako je implementace, cena, technologie apod. Model se skládá z několika různých modelů, popisujících datovou a funkční část systému. Esenciální model má dlouhou životnost, mění se pouze, pokud nové požadavky uživatele mění podstatu chování systému. Jinak jej lze implementovat na základě dalších (cenových apod.) požadavků pomocí odpovídající technologie, případně při změně technologie znovu.

Postup podle Yourdona se liší od klasické strukturované analýzy tím, že se neprovádí dekompozice shora-dolů, ale používá jiný přístup k dekompozici systému. Tvorba strukturované analýzy se provádí

na základě **analýzy událostí.** Vychází se z předem vytvořeného seznamu událostí a cílem je vytvořit DFD, který je modelem chování systému.

Na základě esenciálního modelu proběhne **návrh řešení reálného systému** v dalších krocích:

- Vymezení rozsahu ručních a automatizovaných částí.
- Návrh uživatelského rozhraní, menu - systému, obrazovek a formulářů.
- Návrh činností nutných k ošetření chyb.
- Přechod do fáze návrhu, transformace DFD na structure chart - diagram struktury systému.
- Programování a testování.

Oproti klasickým metodám (původní DeMarco metoda, Chenovy ER diagramy, ...) používá Yourdon jejich kombinaci a rozšiřuje DFD o řídící procesy z STD. Pro specifikaci procesů se mohou použít klasické nástroje - minispecifikace, vývojové diagramy aj. Analýza vede k vytvoření esenciálního modelu a strukturogram (SC - structure chart) vytvářený ve fázi návrhu představuje funkční model složený z funkčních jednotek.

❑ Státní metodologie SSADM

Metodologie SSADM (Structured Systems Analysis and Design Method) byla státní metodologií, pochází z 80.let z Velké Británie. Původně byla definována jako standard při realizaci vládních projektů, postupně se pak rozšířila do mnoha zemí a do vývoje velkých softwarových produktů obecně. Jde o ucelenou metodologii velmi podrobně definovanou, která používá nám již známé nástroje, řadí je do jednoznačně definovaných kroků, jejich prověřování a korekcí. Pokrývá etapy analýzy a návrhu životního cyklu.

SSADM **vychází z datového modelu**. Předpokládá, že systémy obsahují základní datové struktury, které se v čase příliš nemění, i když jejich zpracování a využití se může měnit. Definuje tedy datový model již v počátečních etapách, průběžně jej porovnává s funkčními požadavky a požadovanými výstupy a na základě toho pak případně datový model ještě modifikuje.

SSADM **odděluje logický a fyzický návrh** systému, v počátečních etapách se analytik ani uživatel nezabývají fyzickou implementací, volbou hardware, programovacím jazykem ap.

SSADM již v průběhu analýzy podchycuje **nestandardní situace a chybové stavy** systému. To urychluje vývoj, protože většinou chyby jsou při vývoji systému odhalovány až během psaní programu (programátor uvažuje o možných chybách a ošetření chybových stavů), během ladění (programátor zjišťuje situace nepopsané ve specifikaci) nebo během testování či dokonce provozu (objeví se situace a chyby, se kterými se nepočítalo). Pro včasnou specifikaci možných chybových stavů používá techniky „životní cyklus entity“ a „process outlines“. Pomocí nich již v úvodních etapách projektu

- identifikuje události, které způsobují změnu dat
- identifikuje správné posloupnosti těchto událostí
- přesně specifikuje zpracování každé události.

SSADM obsahuje velmi podrobný **postup a popis jednotlivých kroků** včetně kontrol a předepsaných výstupů před přechodem na další krok. Do přípravy i řešení jsou **zapojeni uživatelé**, kteří po každé etapě potvrzují, že model splňuje jejich představy. Díky standardnímu postupu usnadňuje SSADM **odhad trvání** jednotlivých etap a tak sledování průběhu řešení vedoucími pracovníky. Konečně zdokumentovaný postup a řešení lze **opakovaně použít** u projektů s podobným zaměřením.

Metodologie UML

Po vzniku objektového programování se vyvinuly i metodiky pro objektově orientovanou (OO) analýzu a návrh. Různí autoři zavedli různé metody a nástroje (OMT doplňující YSM o OO prostředí, OOA autorů Coad & Yourdon, OOSE - Jacobsonův přístup založený na scénářích).

Po krátké době se autoři těchto OO metodologií dohodli, vytvořili skupinu a společně se dohodli na nové jednotné OO metodologii, kterou nazvali UML. UML je grafická notace pro objektové modelování (jak pomocí grafů přehledně a srozumitelně dokumentovat objektový software). UML vznikl v těsné spolupráci se zkušenými programátory velikých firem, které nyní jeho používání podporují. Za všechny tyto firmy jmenujme alespoň IBM, Microsoft, Hewlett-Packard či Oracle. Všechny tyto firmy byly již v roce 1997 rozhodnuty používat UML pro dokumentaci svých produktů.

Protože UML byl podrobně probírán v předmětech Úvod do SW inženýrství a Objektově orientované metody, zopakujeme jen jeho základní typy diagramů. Autoři je dělí na 4 skupiny, zobrazující různé pohledy na systém:

1. pohled funkční

- **diagramy případů užití** – ukazují možné způsoby použití systému, pomáhají popisovat uživatelské požadavky při analýze

2. pohled logický – datová struktura

- **třídní diagramy** – zachycují statistickou = datovou strukturu systému, nahrazují dosavadní ERD,
- **objektové diagramy –** popisují vztahy jednotlivých objektů pro konkrétní procesy,

3. pohled dynamický

- **diagramy aktivit** – popisují algoritmus procesu nebo činnosti, nahrazují dosavadní DFD,
- **stavové diagramy** – popisují chování systému či objektu v závislosti na čase nebo časové návaznosti,
- **interakční diagramy**
  - **sekvenční diagramy** – zachycují průběh určité činnosti v systému, komunikaci mezi objekty, které na této činnosti spolupracují a tok řízení mezi nimi
  - **diagramy spolupráce** – zachycují příběh činnosti v systému a komunikaci spolupracujících objektů; oproti diagramům posloupností jsou více zaměřeny na strukturu

4. pohled implementační

- **diagramy komponent** a **diagramy nasazení** – popisují rozdělení výsledného systému na funkční celky (komponenty, moduly) a jejich umístění na výpočetních uzlech informačního systému

Metodologie strukturované a objektové

Na rozdíl od relačních databází se rozvoj objektově orientovaných systémů, programovacích i databázových, příliš nestandardizoval. Existoval jediný relační model, ale celá řada objektově orientovaných databázových produktů, které nebyly vzájemně kompatibilní. V roce 1991 byla založena výrobci software skupina vývojářů, jejímž úkolem bylo vytvořit a dodržovat standard.

Druhou cestou ke standardizaci, vycházející z integrace obou přístupů (relačního a objektového), byla snaha o objektové rozšíření relačních databází. Touto cestou pak vedl hlavní rozvoj SŘBD. Současné SŘBD jsou tak označovány za **objektově – relační** a na rozdíl od klasických relačních mají řadu rozšíření

- v definici **nových datových typů**: připouštějí neatomické atributy jako array (klasické programové pole), fulltext (dlouhý nestrukturovaný text), row (skupinový atribut) atd., navíc používají datové typy BLOB či OLE (velké binární atributy, jako bitmapy, zvukové záznamy apod.)
- pro práci s nimi je **rozšířen jazyk SQL** o nové možnosti jako „odhnízdění“ pole UNNEST, prefixovou notaci pro práci se skupinovými atributy, nové logické operátory pro práci s textem, nové funkce pro práci s tezaurem atd.

Příklady práce s textem si uvedeme v kapitole o textových databázích.

Je zřejmé, že při těchto možnostech SŘBD je poněkud jiná nejen implementace IS, ale i datová analýza. Ve funkční analýze se jen berou v úvahu nové možnosti práce s databází, jinak se nemění.

7.2. Automatizace tvorby informačních systémů

Systémy CASE

CASE (Computer Aided Software Engineering) technologie představuje integrované nástroje, které umožňují automatizovat některé fáze životního cyklu projektu. CASE jsou kombinací programových nástrojů a strukturovaných metod pro návrh a vývoj programového vybavení. Proces výroby programů je definován metodami a nástroje CASE mohou některé fáze procesu zautomatizovat. V současné době se označení CASE používá pro systémy integrovaných nástrojů, které lze využít ve více fázích životního cyklu.

Obvykle se nabízejí tyto kombinace:

- analýza + návrh
- návrh + implementace
- analýza + návrh + implementace
- analýza + návrh + řízení projektu
- analýza + návrh + řízení projektu + metodologie

Taxonomie systémů CASE

Nástroje CASE se klasifikují podle rozmanitých hledisek a vlastností, které jsou dány činností a uživatelem. Uvedeme používané klasifikace

1. **podle funkce** - nástroj může být určen pro podporu strategického plánování, vedení projektu, analýzy, prototypování, implementace, ...
2. **podle role uživatele** - pro manažera, analytika, ...
3. **podle metodiky** - pro klasickou strukturovanou analýzu, pro YSM, SSADM, UML, ...
4. **podle architektury prostředí** - dělení podle technických nebo programových prostředků, tvořících prostředí, kde se využívá nástroj anebo pro které je vytvářen produkt (HW, SW, typ sítě, typ SŘBD)
5. **podle původu** - producent nástroje (Oracle, IBM, ...)
6. **podle ceny** - v jakém cenovém rozpětí je nástroj prodáván
7. **podle fáze životního cyklu** - zde se mohou rozlišovat nejen fáze, ale i modely, ve kterých bude nástroj využíván. Pokud se nepřihlíží k modelu, rozlišují se typy nástrojů:

- **Pre-CASE** - nástroje používané ve fázích před vlastním řešením (globální plánování, studie proveditelnosti).
- **Upper-CASE** - nástroje pro fázi specifikace požadavků a jejich analýzy (prostředky pro komunikaci se zadavatelem a pro dokumentaci požadavků).
- **Middle-CASE** - nástroje řešící přechod fáze analýzy v fázi návrhovou, tj. "co se má udělat" se mění v "jak se to má udělat", kooperace řešitelů na úrovni návrhu.
- **Lower-CASE** - nástroje podporující a dokumentující fázi implementace.
- **Post-CASE** - nástroje používané po fázi uvedení systému do provozu sloužící k údržbě nebo modifikaci systému.

Integrace nástrojů znamená, že jednotlivé nástroje si mezi sebou mohou předávat výsledky své činnosti, např.ve formě dat. Data se většinou soustřeďují do jednotné databáze projektu - **repozitáře**. Díky tomu je možná lepší kontrola chyb a výsledků práce jednotlivých nástrojů. Integrace se může týkat i začlenění nástrojů do společného prostředí, které nabízí vhodnou volbu nástrojů i jejich střídání. V některých CASE systémech je volba řízena metodicky a dochází tak k vedení uživatele a kontrole výstupů jednotlivých fází projektu.

Trendem CASE systémů je poskytnout uživatelům prostředky, které by pokryly celý životní cyklus projektu. Jednotlivé komponenty CASE pak spolu mohou komunikovat přes repozitář, ve kterém jsou uloženy všechny informace o tvorbě systému. Repozitář musí obsahovat reprezentaci tří pohledů na systém: datový, funkční a časový model, také popis prostředí a okolí systému, kde jsou obsaženy informace pro řízení projektu, konfigurace HW a SW.

Shrnutí pojmů 7.

Metoda, metodologie pro vývoj SW díla. Základní vývoj metodologií.
Yordonova strukturovaná analýza, UML.
CASE systémy a jejich dělení.

Otázky 7.

1. Co je metoda, její nástroje, metodologie?
2. Které metody se používají v současnosti pro vývoj informačních systémů? Stručně je charakterizujte.
3. Co je CASE systém a které jejich typy rozeznáváme?

8. PARADIGMA DATOVÝCH SKLADŮ

Čas ke studiu: 3 hodiny

Cíl Po prostudování této kapitoly budete umět

- popsat rozdíl mezi operativní databází a datovým skladem,
- popsat důvody, které vedly k oddělení datových skladů od operativní databáze,
- popsat rozdíl mezi klasickou databázovou úlohou a úlohami analyzování dat z databází
- popsat důvody, které vedly k rozdělení databází na dvě části – operativní databázi a datový sklad
- charakterizovat OLTP, OLAP a DS

Výklad

8.1. Motivace vzniku datových skladů

❑ Informační systémy a operativní databáze

Současný svět je protkán informačními systémy. Počáteční manuální evidence množství údajů, které bylo nutno uchovávat a zpracovávat pro potřebu praxe (*evidence osob, výrobků firmy, knih ve veřejných knihovnách apod.*) se přenesla na počítače, vyvinuly se vhodné technologie jejich zpracování. Vznikly nesčíslné databáze a nad nimi pracující informační systémy (IS).

IS tak shromažďují primární údaje z reality především proto, aby veškeré jejich zpracování probíhalo automatizovaně. Aby bylo možno údaje rychle a bezchybně ukládat, modifikovat a rušit, vyhledávat, provádět nad nimi výpočty, třídit, formátovat, aby je mohli současně používat různí uživatelé a aby je nemohli zneužívat jiní. Protože se s daty provádějí běžné denní operace = transakce, údaje v takových databázích nazýváme **operativními či transakčními daty**. Procesy v informačních systémech také nazýváme On-Line Transaction Process (**OLTP**).

IS tedy slouží člověku především k úspoře množství běžné rutinní práce. Protože jejich výhody jsou výrazné, je dnes uloženo v nejrůznějších databázích nesmírné množství nejrůznějších údajů a pokrývají téměř všechny oblasti lidských zájmů.

Příklad:

Každá firma, organizace zná každodenní objednávky, dodací listy, faktury, zákazníky a zakázky, sklad, účetní deník. Měsíčně provádí uzávěrku, ročně jednu navíc a pak data archivuje. Už se k nim vrací jen málo.

Struktura databáze (optimálně alespoň ve 3NF) je navržena tak, aby se jednotlivým uživatelům dobře a jednoduše s daty pracovalo – klasické entity se svými atributy, klasické vazební entity.

♦

❑ Ekonomické firemní aplikace a podpora rozhodování

Největší procento IS je používáno v ekonomických, firemních aplikacích. Operativní data po čase přestávají být aktuální a ukládají se do archivů, ve kterých se nejvýše občas vyhledává. A vznikají stále nové údaje. Pravděpodobně zde poprvé vznikla myšlenka tato dlouhodobě ukládaná data využít nejen ke svému původnímu účelu, ale i k získání dalších poznatků: analyzovat dlouhodobé hospodaření firmy, modelovat různé varianty dalšího rozvoje, optimalizovat je z různých hledisek apod., a tím vším podpořit rozhodování managementu firmy o jeho dalším vývoji.

Tak postupně vznikaly nové způsoby zpracování dat a později speciální SW systémy nazývané

Management IS (MIS),
Business IS (BIS),
Executive IS (EIS)
Decision Support Systems (DSS),
a ještě jinak.

Takových IS v posledních letech vznikla řada. Na rozdíl od operativních IS nejsou určeny řadovým pracovníkům firmy k jejich každodenní práci. Slouží řídícím pracovníkům firmy. Jim je také přizpůsobeno ovládání systému, formulování požadavků a prezentace výstupů: grafická, tabulková a slovní forma, hodnocení a komentáře výsledků, podpora ekonomickým slovníkem ap. Výstupy jsou výsledkem ekonomických a jiných analýz mnoha typů, vstupem jsou nejen operativní údaje, ale také archivovaná vlastní data i údaje externí o konkurenčních firmách, o poptávce trhu atd.

Protože se tyto systémy nezabývají denními transakcemi, ale analýzou a dalším využitím nahromaděných dat, nazýváme tento proces On-Line Analytical Process (**OLAP**).

Příklad:

Prozíravá firma si před uzavíráním smluv na nákup materiálu pro výrobu udělá rozbor z minulých let. Zjistí, ve kterých měsících se prodávaly které výrobky a kolik. Odhadne možné rozdíly pro letošní rok, spočítá průběžnou potřebu materiálu pro letošek a má dobré podklady k tomu, že jí nebude materiál ani chybět, ani ležet ladem.

Prozíravá pojišťovna si před poskytnutím pojišťovacích smluv udělá rozbor plnění pojistek z minulých let. Zjistí, kteří klienti smlouvu zneužívají, kteří bydlí v ohrožených oblastech apod.. Podle toho upraví své sazby v budoucnosti, případně některé klienty odmítne či naopak jiným nabídne slevy.

V obou případech je nutno vzít data jednak aktuální, jednak z archivů. Archivní data mohou mít jinou strukturu, než data aktuální, protože se v průběhu používání IS postupně databáze měnila. Zřejmě se tak nevystačí s jednoduchými SQL dotazy, ale bude nutné jich udělat řadu a „ručně" kompletovat výsledky.

♦

❑ Výzkumné projekty a analýzy dat

Potřebujeme-li zkoumat objekty (osoby, věci, jevy), které jsou zatím málo známé či příliš složité, že u nichž dosud neumíme popsat jejich vlastnosti a chování, často vycházíme z jejich pozorování. Sbíráme nebo měříme o nich údaje, které pozorovat můžeme, provádíme experimenty s různě nastavenými podmínkami a opět je pozorujeme a zaznamenáváme výsledky. Tak o nich nashromáždíme množství údajů a z nich se pokoušíme vydedukovat jejich další, skrytější vlastnosti.

Data speciálně sesbíraná či naměřená za účelem jejich dalšího zkoumání nazveme **cíleně sebranými daty**.

Tento postup prováděli lidé odpradávna. Zprvu jim za paměťové médium sloužila jen vlastní hlava, později papír, ještě později paměť počítačů. Pro vyhodnocování takových údajů a odvozování nových

sloužil zprvu jen selský rozum, později hlavně matematické disciplíny - především matematická statistika a konečně mnohé metody explorační (objevovací) analýzy dat, umělé inteligence, případně neuronových sítí. Metody analýzy dat vznikaly již dávno v době předpočítačové pro výzkum v nejrůznějších oborech (psychologie, biologie, technika atd.). Doba počítačů jim však teprve dala velké možnosti využití.

Sbírání údajů a jejich vyhodnocení používá nejen vědecký výzkum, ale i formálně jednodušší úlohy, například různé sociologické průzkumy pro účely politické, reklamní, marketingové, dopravní, psychologické atd.

Nazveme-li obecně všechny typy takových vyhodnocování a odvozování **metodami analýzy dat**, tvoří širokou škálu od nejjednodušších statistických charakteristik přes korelační a regresní analýzu, multidimenzionální statistické metody, diskriminační a faktorovou analýzu až k metodám formulujícím případné nové poznatky formou hypotéz, například hledání asociací, shlukovou analýzu, konstrukce rozhodovacích stromů.

Příklad:

Každý z nás se někdy setkal s dotazníkem. Sociologové zkoumají na zakázku organizací, politiků atd. jak se chováme v různých situacích, co si myslíme o jevech kolem nás apod., aby se podle toho mohl zadavatel zachovat pro budoucnost. Každý výsledek veřejného mínění o kdečem je výsledkem analýzy dat.

Firmy zkoumají, kterým výrobkům dávají přednost lidé mladí nebo starší, lidé nevzdělaní nebo vzdělaní, dělníci, zemědělci, studenti, politikové apod. – aby mohli cíleně a tedy efektivněji zaměřit svou reklamu. Reklamní nabídky na jméno jsou často výsledkem takových analýz. Informace získané z dat o zákaznících je možné využít k tvorbě trvalých vztahů se zákazníky, jejichž cílem je zvýšit pravděpodobnost opakovaných nákupů. Znalost těchto informací dává možnost nejenom přizpůsobit nabídku zákazníkovi, předvídat nákupy na základě pravidelností v chování kupujících nebo zkoumat vliv reklamních akcí na prodej. Je to také možnost, jak se vyrovnat se snížením tržeb, jak vhodně uvést na trh nové výrobky, segmentovat trh, předvídat poptávku nebo nacházet nové klienty, kteří by mohli přinést vysoké zisky.

♦

❑ Jak se k dolování znalostí došlo

Jako v mnoha jiných případech, vyústilo těchto několik původně oddělených oborů matematiky, ekonomie a informatiky do nového, integrovaného oboru. Ten je v současné době rychle rozvíjen jak teoreticky, tak budováním nových typů softwarových systémů. I terminologie se v této oblasti rychle vyvíjí a mění. A jako již často, jakmile něco (stará známá teorie, technologie nebo úplně nový nápad) dostane populární jméno, začne žít novým životem. Podobně je to s termínem **dolování znalostí z dat** a databází **(Data Mining)**.

Ovšem dolování znalostí je jen součástí rozsáhlého procesu, do něhož se spojují v integrovaný celek i další disciplíny. Základní data-miningové metody jsou náplní samostatného navazujícího předmětu Metody analýzy dat.

Příklad:

Trenér plavců zkoumá, jestli se dá otestovat měřením „plavecké nadání“ a je-li to jediná vlastnost, nebo jde o více nezávislých vlastností (například talent na krátké a talent na dlouhé tratě apod.). Měřením časů své svěřenkyně - 104 plavkyně testoval ve 4 stylech a 2 délkách tratě. Získal data se strukturou

Plavkyně (50 kraul, 200 kraul, 50 prsa, 200 prsa, 50 znak, 200 znak, 50 delfín, 200 delfín)

Použitím vhodných metod mu datový analytik zjistil, že existují 2 nezávislé typy či „talenty" – ne podle délky tratě, ale podle schopnosti plavat převážně „pomocí paží" – styl prsa nebo „pomocí nohou" – ostatní styly.

♦

Přirozeně se nabízí možnost použít na i pro nashromážděná data firemní nejen klasické ekonomické analýzy, ale i další metody analýzy dat. Ty mohou (samozřejmě nemusí, třeba tam žádné nejsou) odhalit nové, dosud nepovšimnuté souvislosti, závislosti a další skutečnosti o evidovaných objektech, skutečnosti příliš složité nebo málo frekventované nebo z jiného důvodu zatím neznámé. Již víme, že tomuto procesu se říká získávání či dolování znalostí z dat. Dokonce v okamžiku, kdy se dolování znalostí zmocnily SW systémy pro podporu rozhodování, dostala se do definice dolování klauzule „za účelem získání obchodní výhody".

Příklad:

Průzkum trhu a marketing umožňuje poznat potřeby a očekávání klientů, kontrolovat nakolik nabídka firmy tato očekávání splňuje. Předpovídat poptávku a prodej, případně konkurenční výhody, které vám přinese uvedení nového produktu.

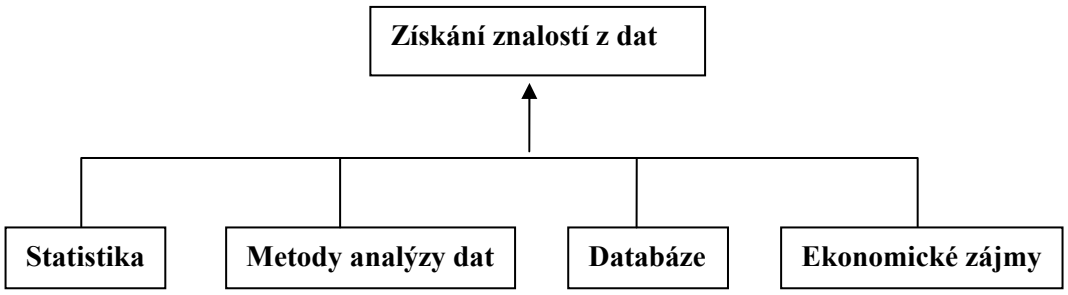
♦

❑ Motivace vzniku datových skladů

Shrňme si fakta dosud uvedená fakta, vedoucí v konečném důsledku ke vzniku nového typu databází – datovým skladům (DS). Další podrobné důvody oddělení databází a datových skladů budeme rozebírat postupně dále.

- existují data nashromážděná v databázích různých IS,
- existují data archivovaná, málo využívaná,
- většina databází je firemních, vývoj jejich IS je v podstatě uzavřen,
- existuje potřeba lépe řídit firmy, objektivně optimalizovat řízení,
- existují metody statistiky, využitelné pro data v databázích,
- existují ekonomické nástroje, metody pro řízení a jeho optimalizaci,
- existují metody pro získávání znalostí z dat.

Tak přišel nápad využít data z databází vlastních i externích, z archivů i dalších zdrojů pro další zpracování. Pro dlouhodobé vyhodnocování dat minulých, pro prognózování, pro podporu strategického rozhodování marketingu, managementu.



Výpočty takových analýz dat se nejprve přirozeně prováděly jako další funkční nadstavba nad operativní databází a jejím informačním systémem. Brzy se však ukázala řada nevýhod tohoto řešení a data pro analýzy byla od operativní databáze oddělena. Důvodů je mnoho a probereme je v následujícím odstavci.

❑ OLTP – DS - OLAP

V 80. letech 20. století vznikl nápad realizovat jakousi paralelu mezi materiální výrobou a výrobou informací.

Materiální výrobky se ze zdrojového materiálu vyrobí, převezou do skladu, tam se vhodně a na míru kupujícím zabalí a konečně prodávají. Příslušné schéma můžeme znázornit jako:

Výroba - Sklad - Prodej

Obdobně se zdrojová data z databází (~ zdrojový materiál) zpracují do ucelených a globálních informací (~ výrobky) a uloží do samostatné databáze, nazvané **datovým skladem**. Poté se na míru zákazníkům zabalí do vhodné, srozumitelné podoby a prodávají. Jednotlivé etapy jsou označovány jako

OLTP - DS - OLAP

kde

OLTP (On Line Transaction Processing) ... je klasický operativní IS se zdrojovými daty v databázi DB

DS, DW (datový sklad, DateWarehouse) … je datový sklad, databáze všech zdrojů, místo výpočtů a úložiště předpočítaných výsledků

OLAP (On Line Analytical Processing) ... je nástroj pro výběr a prezentaci nových informací, užitečných znalostí

Obdobně jako nad klasickou operativní databází pracuje informační systém se svými transakcemi, nad datovým skladem pracuje jiný typ programového systému, systém pro podporu rozhodování.
Zavedeme si pro ně zkratky

DB je operativní databáze ovládaná informačním systémem **IS**,

DS je datový sklad ovládaný systémem pro podporu rozhodování **DSS** (Decision Support Systém).

„Otcem“ datových skladů je kanadský matematik a databázista W. H. Inmon.

❑ Využití datových skladů nejen ve firmách

Konečně budování datových skladů s DSS a procesem dolování znalostí je možno výhodně použít nad databázemi i z jiných, než ekonomických oborů. Například dlouhodobé evidence v medicíně, psychologii, biologii, školství atd., údaje ze sčítání lidu a mnohé další mohou skrývat netušené skutečnosti.

Spirála vývoje se tak posunula o úroveň výše:

- původní cíleně sbíraná data pro výzkum a metody jejich zpracování inspirovaly k využití i data nashromážděná původně za jiným účelem v databázích,
- využívání v databázích firemních inspirovalo ke zřízení oddělených datových skladů a nad nimi pracujících DSS,
- princip datového skladování zpětně inspiroval i jiné než firemní databáze k oddělení svých dat do datových skladů za účelem dalšího analyzování.

Použití dat z databází však přináší několik nových problémů, o nich se zmíníme v následujících kapitolách.

8.2. Porovnání databáze a datového skladu

❑ Charakteristika OLTP – databáze DB ovládaná IS

Jak známe z teorie relačních databází, operativní databáze se navrhují optimalizovaně pro údržbu velkého množství dat a pro selektivní vyhledávání informací. To znamená bez redundancí, podle jasně definovaných pravidel. Pravidla nazýváme normálními formami a jsou známy algoritmy, jak takto dobře strukturovanou databázi navrhnout. Databáze, jejíž všechny relace (tabulky) mají alespoň 3. normální formu, nazýváme **normalizovanou databází**.

Důvody normalizace jsou také známy. Normalizované tabulky neobsahují redundance dat a proto u nich nedochází k nekonzistenci, případně dalším možným průvodním jevům, jako anomáliím při vkládání nebo anomáliím při vypuštění údajů.

Nad normalizovanou databází probíhá klasický operativní provoz IS se svými každodenními transakcemi. S daty pracují hlavně administrativní pracovníci, na databázi se kladou především **dotazy selektivní** (*najdi nezaplacené faktury, najdi materiál s aktuálním množstvím menším než požadované minimum apod.*). Rozmístění dat je proto obvykle optimalizováno dle potřeb operativních oddělení.

Data jsou pravidelně archivována, archivy dále příliš nevyužity.

❑ Problémy při využití OLTP pro podporu rozhodování

Pokud chceme data zdrojová (i z více zdrojů) i archivovaná využívat také pro podporu rozhodování, nastávají některé nové problémy.

Tradiční integrace: je možno ponechat dílčí databázová schémata a pomocí nich vytvořit globální schéma DS. Jde o přístup známý z teorie distribuovaných databází (viz obrázek).

Tento přístup je řízen uživatelskými dotazy. Pro splnění dotazu se vyhledají v globálním schématu odpovídající zdroje informací, načtou se, zpracují a výsledek se předá uživateli. Je to pomalý a obtížný postup, znamená složité vybírání dat přes několik dílčích schémat pro každý dotaz. Neefektivní je také opakování těchto akcí při opakovaných dotazech téhož typu.

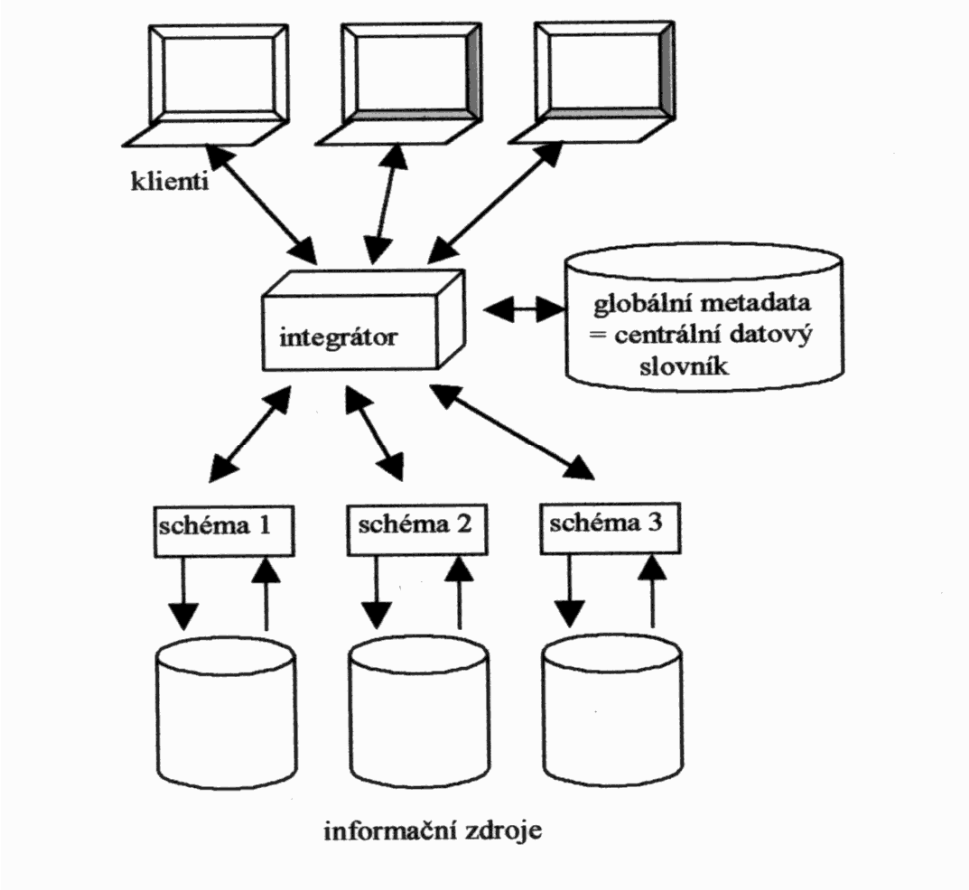
Pro provoz operativní databáze to znamená velké časové i kapacitní nároky.

Management produkuje jiný typ dotazů, než v uživatelé operativních databází. Jeho zájem obvykle není informovat se na jednotlivé záznamy, ale zajímají ho globální informace. Formuluje **dotazy intenzivní** (*kolik podvozků jsme dodali do SM kraje za poslední čtvrtletí a o kolik byl větší zisk v porovnání se stejným obdobím loni*). Tyto informace nejsou v databázi přímo obsaženy, je nutné je vypočítat ze zdrojových dat. Navíc **nejsou tyto dotazy předem definované** (přináší je okamžitá velmi různorodá a velmi se měnící potřeba vedení) a tedy jejich zpracování není v IS zabudováno.

I při možnosti formulovat libovolné SQL dotazy nad operativní databází existuje jistá nedosažitelnost dat skrytých v transakčních systémech. Někdy je nutné výsledky SQL dotazů ručně kombinovat pro dosažení požadovaných výsledků. Odtud tedy plynou následující problémy:

- do analýz mají vstupovat data bezchybná - konzistentní, úplná, často transformovaná, na různých úrovních agregovaná, vybíraná z různých zdrojů - také z archivu a externích dat,
- mimo vlastní data je nutno evidovat pro potřeby zobrazování výsledků i jejich definici, strukturu a další informace o datech (tzv. metadata),
- aplikační programy musí mít zcela odlišný charakter od algoritmů operativních,

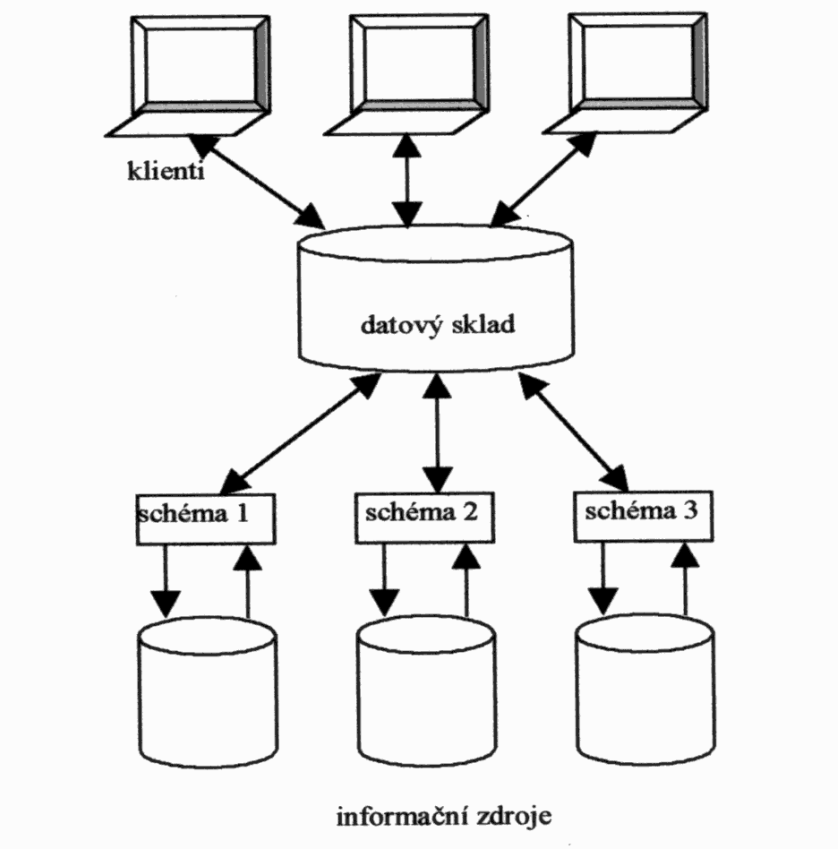
- algoritmy nad daty prováděné bývají časově náročné a zatěžují tak provoz základního IS, způsobují dlouhé prodlevy při provádění komplikovaných dotazů, soutěží o počítačové zdroje mezi transakčními dotazy a dotazy pro podporu rozhodování,
- údaje by bylo vhodné ukládat na médiu jiným způsobem, než data operativní,
- na druhé straně není nutné, aby data byla právě aktuální, podstata analýz je obvykle ve vyhledávání obecně platných skutečností z dlouhodobějších dat,
- dosavadní přístup k datům je složitý, je to uživatelsky nepříjemné rozhraní k databázovému SW (pro zadávání požadavků buď existující aplikace, nebo dotazovací jazyk SQL, QBE; to je vhodné jen pro pracovní výstupy; neexistuje automatické formátování, grafické vizualizační prostředky pro nepředpokládané dotazy),
- analýzy jsou drahé na práci v administrativě, střední články řízení firmy zpracovávají rozbory dat často „ručně“ a zdlouhavě, je velká složitost v podpoře vzdálených uživatelů.



Charakteristika datových skladů a DSS

Uvedené důvody vedly postupně k novému řešení, k definování nové samostatné databáze vyšší úrovně – datového skladu a nad ní pracujícímu systému pro rozhodování DSS.

Do datového skladu se uloží údaje z operativní databáze i z archivů a dalších zdrojů. Přitom se předzpracují tak, aby byly optimálně k dispozici pro OLAP a analyzující algoritmy Data Miningu. Předzpracování zahrnuje kontroly konzistence a úplnosti dat, transformace (do stejných měrných jednotek, stejných datových formátů ap.), agregace (sumace časové, prostorové ap.), naplnění metadat, vhodnou organizaci uložení dat na médiu.



Z databázového hlediska vzniká DS **integrací heterogenních dat**, odtud jeho definice.

Definice:

Datový sklad je integrovaná, subjektově orientovaná, stálá a časově rozlišitelná sbírka dat, uspořádaná pro podporu potřeb managementu.

Základní charakteristiky skladování dat

- Oddělené uložení dat, aby jejich náročné zpracování nenarušovalo provoz operativních systémů.
- Data jsou nejen z operativní databáze, ale i z archivů, z jiných externích zdrojů i z reality (nejen vlastní data, ale obecná data přeměněná ve strategické informace).Odtud velké rozměry databáze, až TB.
- Data se integrují jednorázově a uloží do DS s vlastními metadaty a vlastní organizací dat. Výhodou je rychlejší odezva při vyhodnocování dotazů, bez zatěžování operativních databází.
- Data jsou předpřipravena - transformována, sjednocena, zkontrolována, agregována, ve formátu vhodném pro systémy pro podporu rozhodování.
- Další možností jsou (dávková) doplňování dat, sumarizace, evidence historických dat.
- Data nejsou kategorizována podle požadavků jednotlivých útvarů, jako u operativních databází pro zpracování každodenních transakcí, ale jsou soustředěna (koncentrována) podle typu sledovaných objektů a podle potřeb managementu na různých úrovních. Jejich struktura, formát i sémantika a fyzické uložení jsou optimalizovány pro potřeby celého podniku a pro nástroje OLAP.

- Vlastní analýzy a jejich prezentace se nyní provádějí nad DS a jsou optimalizovány pro přímé dotazování vybraných typů otázek (pro OLAP).
- DSS mají možnost v krátké době odpovídat na otázky ad hoc = „přinesené životem“, na které není možno se připravit předem, předprogramovat je, ale které jsou důležité pro rozhodování podnikových stratégů; říká se, že informace, která přijde pozdě, již není informací.
- DS se používají pro prezentaci dat, testování hypotéz i objevování nových znalostí (dolování dat) prostřednictvím specializovaného software.

❑ Rozdíly mezi tvorbou IS a DS

Proces analýzy, návrhu i implementace datových skladů a systémů pro následné analýzy a prezentaci jejich výsledků se výrazně liší od analýzy a implementace operativních IS.

Oproti klasickým IS je nutno používat jiné technologie

pro modelování (tzv. multidimenzionální modelování)

- Data jsou organizována jinak, než v klasické DB, jsou integrována (sjednocena z různých zdrojů), uspořádána do homogenních pohledů pro následné vyhodnocení. Protože jsou data využívána pro analýzy v různých časových obdobích, musejí vytvářet časové řady. Pro potřeby managementu nebývají důležitá detailní operativní data, ale údaje souhrnné, agregované.

pro předzpracování dat

- Přenos dat ze základní databáze do DS se někdy nazývá **datovou pumpou**. Doplňování dat do DS novými údaji je možné provádět v (pravidelných) intervalech například mimo pracovní dobu a nezatěžovat tak provoz operativního IS nebo podle okamžité potřeby splnit některé požadavky uživatelů DS.
- Součástí doplnění dat je jejich integrace, filtrace a transformace. Po doplnění dat se provádějí přepočty dat odvozených a agregovaných.

pro uložení dat

- Ukládají se nejen vlastní operativní, detailní data, ale různé agregace do dimenzí (čas, geografické dimenze, komodita ap.), tedy na rozdíl od relačního normalizovaného uložení dat jsou zde data redundandní, nenormalizovaná;
- Základní rozdíl je v implementaci

  ROLAP (Relation OnLine Analytical Proces) - základem je relační model bez normalizace
  POLAP (Proprietary OnLine Analytical Proces) - fyzická implementace multidimenzionální databáze jako multidimenzionální kostka se speciálními přístupovými cestami)
  MOLAP (Multidimenzionální kostka) – jiný název pro POLAP

pro zpracování dat

- Nepoužívají se klasické DB operace insert, update, delete, ale jen pravidelné doplňování dat, data jsou stálá.
- Uživatelské rozhraní je optimalizováno pro neprofesionálního uživatele, má uživatelsky přívětivé možnosti dotazování, provádění statistických analýz, Data miningu.

pro prezentaci výsledků

- DSS má nástroje pro grafické, tabulkové i slovní prezentace a vizualizace dat, předzpracovaných agregací i výsledků Data Miningu, pro reporting.

Datová tržiště

Z hlediska architektury implementace DS nemusí existovat jediná fyzicky centralizovaná data celopodniková. DS může být buď fyzicky centralizovaný nebo logicky centralizovaný a fyzicky decentralizovaný nebo konečně fyzicky distribuovaný.

Mimo komplexní DS uložené v jediné obrovské databázi vznikají i částečně decentralizované tzv. **Data Marts (DM)** - datové tržnice nebo **Operational Stores** - provozní sklady.

Používají se většinou pro speciální a detailnější analýzy, soustředěné např. jen na některé aspekty podnikání. Datové tržnice nebo provozní sklady obsahují poměrně čerstvá data a slouží k on-line analýzám tam, kde je nutné reagovat na určité trendy v reálném čase (např. příprava surovin pro přípravu jídel v provozovně v závislosti na momentální poptávce, aby se maximálně zkrátila doba odbavení zákazníků).

Charakteristika OLAP pomocí DSS

Třetím krokem je „prodej dat“ předzpracovaných, sumovaných, kdy pohled na data i z nich odvozené znalosti jsou předkládány uživateli velmi srozumitelnou formou. Využívají se metody statistické, modelování trendů apod., aby se staly pro uživatele z hlediska jeho dotazů průhledné a srozumitelné.

Systémy jsou optimalizovány pro rychlý výběr dat, sumarizace, analytické zpracování velkých objemů dat. Cílem je poskytnout rychle hodnotné informace z velkých objemů dat. Při dotazování je uživateli předkládána struktura DS, uživatel zadává dotazy na informace v různých úrovních agregace:

- na detailní data
- na součtové tabulky

Agregované hodnoty mohou být předpočítány na různých úrovních podrobnosti (*součty cen za prodané zboží za den, za týden, za měsíc, rok, průměrná tržba za toto období, průměrná tržba za prodejnu, obvod, kraj, stát apod.).*

Samozřejmostí DSS musí být snadné a rychlé přepínání mezi hierarchickými úrovněmi – nahoru i dolů, nebo zobrazování časových či dimenzionálních řad (průběh zisků za řadu měsíců, za všechny prodejny apod.). Postupy horizontální a vertikální se nazývají

- Drill-down
- Roll-up
- Slicing & dicing

Dolování znalostí z dat (DM)

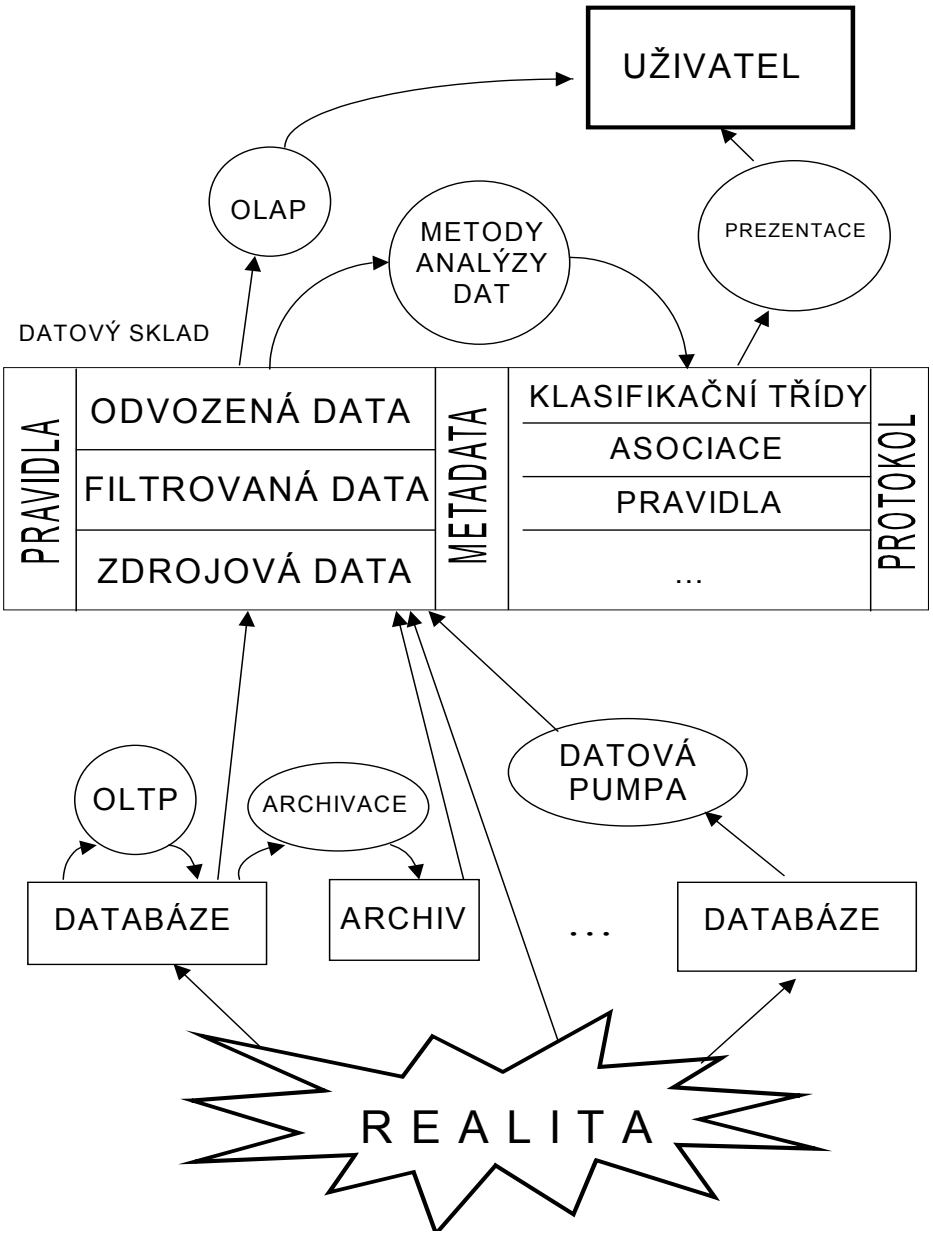
Čím dále častěji se DSS doplňují i o netriviální metody dolování znalostí z dat. Tyto metody hledají v datech nevyžádané a dosud neznámé informace, které mohou majiteli těchto informací přinést další výraznou výhodu v jejich zužitkování. Protože jde o samostatnou a velkou oblast metod, je jim věnován samostatný předmět.

K metodám dolování patří jak metody matematické statistiky, tak metody explorační analýzy.

- korelační analýza, jednoduché, parciální a kanonické korelace,
- regresní analýza, vícenásobná a logistická regrese,
- výběr proměnných, optimalizační metody,
- rozhodovací stromy,
- analýzy asociací, analýzy příčin a následků,
- klasifikační a shlukovací metody,
- neuronové sítě,
- a další metody.

Shrnutí procesů a datových toků IS – DSS - DM

Na následujícím obrázku je shrnut celý postup vzniku, ukládání, skladování a využití dat.



Úplný pohled na datový sklad a jeho okolí

Zdrojová data se vždy berou z reálného světa. Obvykle se ukládají z důvodů jejich operativní evidence do databází, obvykle ovládané příslušným informačním systémem. Mohou však existovat i v jiných formátech – v excelovských tabulkách, textových tabulkách, jen „na papíře“ apod. Data již neaktivní obvykle ukládáme do archivů. Jsou-li data v DB podpořené IS, nazýváme celý tento proces OLTP.
Takových databází může být mnoho i když je majitelem dat tentýž subjekt – firma, organizace, jednotlivec *(firma má zakoupeno několik IS – pro mzdy jeden, pro skladovou evidenci další, pro účetnictví další apod.)*
Rozhodne-li se majitel těchto dat pro jejich další využití, je vhodné vybudovat integrovaný datový sklad

Shrnutí pojmů 8.

Informační systémy, operativní databáze, On-line Transaction Process (OLTP). Dotazy selektivní a intenzivní.

Využití firemních operativních databází pro podporu managementu. On-Line Analytical Process (OLAP). Využití OLTP pro realizaci dotazů intenzivních a jeho problémy.

Datový sklad (DS) jako nový typ databáze, jeho charakteristiky. Systémy pro podporu rozhodování DSS. Rozdíl mezi operativní databází a datovým skladem.

Analýzy dat, výzkumné projekty, dolování znalostí z dat. Data Mining nad datovým skladem.

Otázky 8.

1. K čemu hlavně slouží klasické databáze se svými informačními systémy - OLAP?
2. K čemu dále mohou být využita dlouhodobě uložená data z operativních databází?
3. Která sféra má největší zájem na dalším využití dat z databází?
4. Jaký je hlavní důvod pro oddělení operativní databáze od databáze pro OLAP?
5. Co je datový sklad a jaké důvody vedly k jeho vzniku?
6. Je možno využívat datové sklady i mimo ekonomickou sféru?
7. Jaký je rozdíl mezi systémem pro podporu rozhodování a informačním systémem?
8. Proč není vhodné používat klasickou databázi s informačním systémem i pro řešení úloh pro podporu rozhodování?
9. Jaké jsou základní charakteristiky datového skladu?
10. Co je OLAP a které typy úloh patří k OLAP?
11. Jak je možné využít data v datovém skladu i mimo OLAP?

9. ANALÝZA DATOVÝCH SKLADŮ

Čas ke studiu: 3 x 2 hodiny teorie + 3 x 2 hodiny řešení úloh

Cíl Po prostudování této kapitoly budete umět

- popsat rozdíl mezi analýzou a modelováním databáze a datového skladu
- popsat úkoly konceptuálního modelování datového skladu
- prakticky rozhodnout, která data budou součástí datového skladu, která ne a proč
- popsat datové tržnice a jejich vztah k datovému skladu
- prakticky definovat pravidla pro datovou pumpu
- popsat multidimenzionální modelování a jeho úkoly,
- prakticky rozdělit atributy při zařazení do datového skladu na fakty, dimenze a ostatní atributy,
- prakticky definovat omezení na dotazy pro fakty vůči dimenzím a agregacím,
- charakterizovat ROLAP a MOLAP
- definovat a vysvětlit v ROLAP, co je hvězdicové schéma a souhvězdí modelu datového skladu,
- popsat a na příkladech z praxe použít modelování datových skladů s hierarchií dimenzí pomocí ROLAP.
- popsat, jak se v MOLAP řeší modelování dimenzionálních hierarchií a odpovídající hierarchie faktů
- popsat, co jsou multikostky a co hyperkostka.

Výklad

9.1. Fáze modelování datových skladů

Z teorie vývoje informačních systémů víme, že klasické modelování databáze při budování IS zahrnuje 3 fáze, modelování

- konceptuální, návrh logické struktury pomocí ERD nebo třídního diagramu,
- databázové, realizace konceptuálního schématu v konkrétním prostředí SŘBD,
- fyzické, vlastní implementace databáze, realizovaná samotným SŘBD.

Modelování datového skladu obsahuje 4 fáze modelování:

- konceptuální
- ***multidimenzionální***
- databázové
- fyzické

Již jsme si naznačili, že struktura datového skladu je jiná, než u operativní databáze. Především v tom, že obsahuje řadu předpočítaných (tedy redundandních) údajů. Předpočítání spočívá hlavně ve výpočtech součtů, počtů průměrů mnoha údajů (obecně agregovaných hodnot), a to i na několika

úrovních podrobnosti, tedy v jakési hierarchii agregací. Tyto údaje je nutné někam uložit, navrhnout pro ně nové datové struktury. To je úkolem multidimenzionálního modelování.

9.2. Konceptuální modelování DS

❑ Úkoly konceptuálního modelování

Při konceptuálním modelování datového skladu se nevychází ze zadání zadavatele a jeho potřeb evidence, jako u IS. Situace je jiná a můžeme ji charakterizovat následujícími body:

- DS vychází z existujících databázových schémat, **konceptuální modely zdrojových IS jsou pro DS zadáním**. Úkolem je na základě mnoha zdrojů (databázových schémat) vytvořit integrované schéma budoucího DS. To není jen sjednocením zdrojových schémat, ale úkolem konceptuálního modelování je rozhodnutí, **která data budou součástí DS** a která ne.
- Po výběru atributů je nutné definovat **pravidla pro datovou pumpu**: jak se při přenosu vybraných atributů bude každý atribut filtrovat, integrovat atd.
- Dalším úkolem je koncepční rozhodnutí, zda se bude tvořit jediné schéma pro **jednotný DS**, **nebo** jedno pro každý Data Mart, **datovou tržnici** (DM).

Tento proces se někdy označuje zkratkou ETL (Extrakce, Transformace, Loading) a pojmenovává tak dílčí úkoly.

❑ Výběr atributů pro datový sklad (Extrakce)

Prvním úkolem konceptuálního modelování je rozhodnout, co všechno (které atributy) se v něm budou uchovávat. Podkladem pro rozhodování jsou existující struktury zdrojových databází i případně dalších vhodných dat.

Výběr atributů z DB pro DS tedy znamená:

- shromáždit seznam zdrojových DB a jejich schémat (aktuální operativní DB, archivy, další zdroje DB i jiné (formátem obecně z textových, excelovských a jiných souborů, obsahem data získaná z cizích databází, z internetu apod., pokud mohou vhodně doplnit a zhodnotit DS),
- vybrat z nich informace (tabulky, atributy) vhodné pro DS; přesněji si popíšeme data vhodná pro DS v multidimenzionálním modelování,
- definovat strukturu dat atomických, tedy dat okopírovaných z jednotlivých záznamů zdrojových databází, ale již integrovaných a transformovaných.

Příklad:

*Dvacetiletá databáze **prodejny ABC**, prodávající dodávané zboží, měsíčně archivuje data o prodeji starším 2 let (po uplynutí záruky). Za těch 20 let prošla firma i její databáze a informační systém vývojem: původní systém ve FoxPro byl nahrazen po 10 letech jiným v Delphi s databází v InterBase a v posledních 4 letech byl zakoupen nový SW v Oracle. Každý ze systémů má poněkud jinou strukturu databáze, používá i jiné datové typy a jiné měrné jednotky na některé druhy zboží. V první databázi vytvářené na zakázku se postupně dokonce 2x změnila datová struktura několika tabulek – přibyly nové atributy. Proto ani archivní tabulky různě staré ve stejném SŘBD nemají totožnou strukturu.*

Prvním úkolem je získat struktury všech těchto zdrojových databází, porovnat je a navrhnout pro ně jednotnou (integrovanou) strukturu, která se pro všechny tyto zdroje použije v DS - strukturu dat atomických.

♦

Vycházet se zřejmě bude ze současné struktury operativní databáze, protože v ní budou data dále přibývat.

- Atributy, které se dříve nevyskytovaly a jsou dále pro DS důležité, zařadíme do struktury a ve starších záznamech se datovou pumpou označí jako chybějící.
- Atributy, které se dříve vyskytovaly a nyní ne, posoudíme a rozhodneme o jejich důležitosti a nezařazení či zařazení do DS, byť jen u starších dat.
- Některé atributy můžeme z DS vyloučit *(například číslo dodacího listu dodavatele – nejspíš umělý klíč v databázi dodavatele, pro náš DS dále nedůležitý, číslo bankovního konta dodavatele – nejspíš konstantní hodnota pro DS nedůležitá apod.)*.
- Atributy s různými měrnými jednotkami (například původně v kusech, nyní v balení po 4 ks, nebo původně v gramech, nyní v kg apod.) označíme a řešíme v datové pumpě.

Podrobněji se o jednotlivých atributech rozhodneme, až si probereme multidimenzionální modelování.

❑ Datová pumpa (Transformation)

Máme strukturu atomických dat DS a vybráno, které atributy budou v DS uloženy. Dalším úkolem je definovat **pravidla pro přenos, filtrace a transformace** zdrojových dat do DS nebo DM, tedy definovat funkce datové pumpy.

Každému zdrojovému údaji přiřadíme jeho integrační funkci do DS. Nejjednodušší případ je pouhá kopie zdrojové hodnoty do DS, pokud není třeba některé z následujících kontrol, úprav, transformací:

Definice všech funkcí čistících, integrujících, transformačních – pro každý atribut se provede

- kontrola správnosti údajů a případné **čištění dat chybných (cleaning)** – oprava chyb: pokud jsou data ve zdrojové databázi dobře kontrolována, není tato funkce potřebná; pokud tam existují data chybná, jsou nyní 2 možnosti opravy: buď odhalit chybu a odstranit ji až pomocí datové pumpy v DS, nebo spoluprací se správcem DB doplnit kontroly již zdrojového IS; chybná historická a existující data však je nutné v datové pumpě opravit vždy;
- řešení **chybějících hodnot** – opět se rozhodujeme mezi několika možnostmi: údaje téměř nevyplněné můžeme **ignorovat**, nezařadit do DS; jsou-li to důležité údaje pro DS a máme tu možnost, **doplnit** je z jiného zdroje; nejsou-li údaje zcela prázdné a atribut je pro DS důležitý, **označit** je speciální hodnotou.

  Příklad:

  Pro atribut s nezápornými hodnotami zvolíme hodnotu -1 pro chybějící údaj a pak s takovou hodnotou pracujeme i v DS; ♦

  Uživatel pak je dostatečně informován o tom, kolik údajů se účastní analýz, protože jejich průměry, sumy a další agregované hodnoty by byly zkreslené různým počtem použitých hodnot;
- řešení **dat konstantních** – mají-li některé atributy konstantní hodnotu, nebudeme je zařazovat do DS, protože nemohou přinést nic zajímavého; je-li atribut téměř konstantní, zvážíme rozsah a důležitost jeho různých hodnot a rozhodneme o jeho zařazení do DS.

  Příklad:

  Atribut stát *u dodavatelů zboží je z celkového počtu 358 dodavatelů jen 12x jiný, než ČR; zvážíme, jestli nás zajímá či v budoucnu bude zajímat, odkud se kolik zboží a za kolik peněz dováží a jak se tento údaj vyvíjí – a rozhodneme se atribut ponechat; firma zaměstnávající na těžkou fyzickou práci jen muže má atribut* pohlaví *konstantní a nebude ho přenášet;* ♦
- řešení **nejednotných hodnot údajů** - stejné hodnoty mohou být uživatelem postupně různě zaznamenány; nejčastěji jde o textové údaje *(jméno, adresa, názvy, slovně vypisované*

číselníkové údaje jako pohlaví, rodinný stav atd.), ale může jít i o další datové typy; úkolem je sjednotit stejné údaje z reality na stejně zapsané údaje v DS; obvykle je nutné

- definovat **konvence pro textové údaje** jako názvy, jména, adresy, ... (pořadí, velikost písma atd., viz. pravidla pro záznam do datového slovníku); pokud přenášené údaje do DS neodpovídají konvenci, upravit jejich hodnotu.

  Příklad:

  Jméno zástupce firmy je zapsáno jednou jako NOVÁK Josef, jindy jako Josef Novák nebo Ing. Josef Novák nebo Novák Josef, Ing. nebo J. Novák atd.; název firmy je jednou ABC spol. s r.o., jindy ABC s.r.o, jindy jen ABC atd.); ♦

  Někdy je nutný i ruční zásah při rozhodnutí, zda jde o stejnou hodnotu;

- obdobně definovat **formáty** některých dalších údajů nejen textových, ale i datumových nebo numerických, která mohou být v různých databázích zapsána různě, případně různými uživateli zapsána různě.

  Příklad:

  rodné číslo *jednou jako text s lomítkem, jindy jako text bez lomítka se starými devíticifernými rodnými čísly s mezerou na konci, jindy jako číslo s desetinnou tečkou nebo číslo deseticiferné bez tečky; podobně PSČ jako text s mezerou, jako číslo bez mezery atd.; datum narození jednou ve formátu dd.mm.rrrr, jindy jen dd.mm.rr nebo dd-mm-rrrr nebo dd/mm/rrrr atd.,…*; ♦

- definovat **měrné jednotky** numerických údajů a podle potřeby navrhnout přepočítací koeficienty pro starší nebo cizí zdrojová data – pro každý zdroj může být jiný;

- řešení **duplicitních záznamů,** pokud ve zdrojové databázi opět nebyla dostatečná kontrola; odhalit duplicity a odstranit je, případně opět do budoucna doporučit další kontrolu do IS;

  návrh **odvozených údajů**: jak uvidíme lépe v multidimenzionálním modelování, z některých zdrojových údajů bude užitečné odvodit data další a uložit je, byť jsou taková data redundandní; při velkých objemech dat v DS by bylo velkým zdržením vždy odvozené údaje nejprve počítat a pak dělat nad nimi dělat dotazy. Opět řadu příkladů lépe uvidíme dále.

  Příklad:

  Častým případem jsou data odvozená z datumových atributů – den, měsíc, rok, dekáda, čtvrtletí, den v týdnu apod.;

  Jiným příkladem je období mezi dvěma časovými údaji, jako věk pacienta v době hospitalizace = počet roků mezi datem narození a datem hospitalizace apod.; ♦

- **časové označení dat** je posledním krokem, každý záznam má své časové razítko, kdy byl do DS přenesen.

❑ Přenos dat do DS (Loading)

Analýza přenosu dat se zřejmě dělí podle zdrojů dat na 2 části:

- Jednorázový přenos dat z archivů a jiných starších zdrojů – data se jednorázově integrují, transformují a přenesou do DS. Situace může být složitá, je třeba pro každý zdroj provést integrace samostatně.

- Z aktuálních operativních dat se budou přenášet nové přírůstky dat opakovaně – buď se navrhne perioda, jak často a kdy se data budou pravidelně automaticky přenášet, nebo bude přenos dat proveden vždy po volbě uživatele – správce DS. Perioda závisí na velikosti přírůstku a důležitosti mít v DS data aktuální

Příklad:

U dat z prodeje asi bude důležité sledovat denně i okamžitý vývoj zájmu o jednotlivé druhy zboží;

U dat lékařských určených k výzkumu bude stačit doplňovat data 1-2x ročně; ovšem při chřipkové epidemii bude opět vhodné sledovat data denně.

♦

Celkem má funkce datové pumpy 3 fáze. V obou případech (data z archivů nebo data z aktuální databáze) se nová data

1. filtrují, integrují, transformují **do atomických dat** datového skladu; atomická data jsou sjednocenou, integrovanou kopií dat zdrojových s případným vyloučením dat v DS nepotřebných;
2. z atomických dat se integrovaná data překopírují do **nové struktury** datového skladu (její popis viz níže),
3. nad základními daty se spočítají navržené **agregace** a uloží do datového skladu.

Atomická data mohou v DS zůstat nebo jsou po celém procesu smazána jako mezivýsledek. Již víme, že dotazy nad DS se obvykle netýkají jednotlivých záznamů, ale hodnot sumovaných.

❑ Datový sklad a datové tržnice (Loading)

Součástí plánu pro přenosy dat je koncepční rozhodnutí o rozmístění dat datového skladu. Není totiž vždy nutné či účelné uchovávat data v jediné rozsáhlé databázi. Je možno také vytvořit ucelené části DS, které budou sloužit jisté části uživatelů:

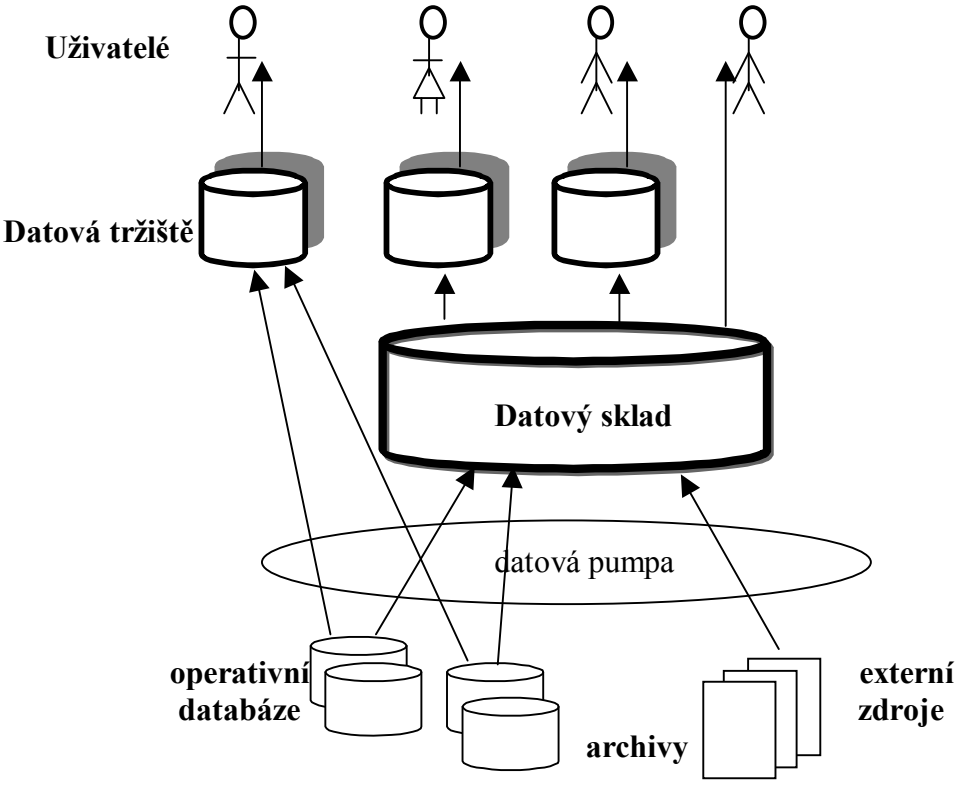
Definice:

Datová tržnice (Data Mart DM) je účelová, oddělená podmnožina datového skladu, určená pro jistou skupinu lidí, většinou oddělení nebo útvar.

Možnosti řešení je několik. Může existovat jediný DS, samostatné DM nebo jejich kombinace.

Mezi datovým skladem a datovými tržišti může být několik typů propojení:

1. Zdrojová data se načítají (integrují, filtrují, transformují) samostatně do všech datových tržnic, ve kterých jsou tyto informace zapotřebí. Existují tak jen oddělené tržnice, neexistuje integrovaný datový sklad. Výhody jsou v menších rozměrech jednotlivých tržišť a rychlejší odezvě uživatelům. Nevýhody jsou v redundandním předzpracování i v redundandních datech na více místech, odtud pak problémy s údržbou a konzistencí. Pokud doplňování dat není jednotně řízeno, ale každá tržnice si řídí doplňování sama, mohou se data časem rozcházet.
2. Zdrojová data se načítají do datového skladu, tam se jednorázově integrují, jejich části se pak přenášejí do oddělených tržišť. Výhody jsou v jednorázovém a jednotném zpracování dat, menší problémy s údržbou a v rychlém přístupu jednotlivých uživatelů k menším datům. Nevýhody jsou v redundanci stejných informací na mnoha místech.. Plnění centrálního DS se nazývá primárním procesem, plnění datových tržnic procesem sekundárním
3. Existuje pouze komplexní centrální datový sklad bez datových tržišť, ke kterému přistupují všichni uživatelé DSS, přistupují ke stejným datům. Výhodou je jediné předzpracování bez dalších redundancí v datech a centrální údržba, nevýhodou ohromný datový sklad a tím i nižší výkon a flexibilita, pomalejší odezva mnoha uživatelům.



Možná řešení datového skladu a datových tržišť

9.3. Multidimenzionální analýza DS

❑ Úlohy multidimenzionální analýzy

Tato etapa analýzy je nová, neznáme ji z modelování operativní databáze a zahrnuje tyto úlohy:

- rozdělení atributů DS podle jejich rolí v DS na fakty, dimenze a ostatní atributy,
- návrh struktury DS - definování vztahů mezi dimenzemi tvořících základ tabulky faktů,
- definování hierarchie dimenzí,
- určení aditivity faktů a definice omezení dotazů.

❑ Dimenze, fakty, atributy

Dotazy v IS nad operativní databází jsou převážně na úrovni jednotlivých entit. Dotazy v DS jsou téměř výhradně na hodnoty sumované, tedy převážně na hodnoty některých atributů agregované podle jiných atributů, na jejich tzv. dimenzionální řady a na jejich hierarchii.

Proto musíme rozdělit zdrojové atributy na ty, které budou agregovány (sumovány, průměrovány apod.) a ty, podle kterých se bude agregovat. Protože toto rozdělení budeme často používat, zavádíme nové názvy typů atributů pro jejich rozdílné role. Při výběru atributů do DS tedy bereme v úvahu i tyto jejich budoucí role.

Rozdělení atributů původního konceptuálního modelu:

- **dimenze** jsou atributy, podle nichž dává smysl sumovat a agregovat jiné atributy (*čas = denní, týdenní měsíční roční sumy, prodejce = suma za prodejnu, za celou firmu atd., výrobce, komodita = sumy za jednotlivé zboží, za typ zboží atd., místo = zákazník z obce, okresu, kraje, státu atd. , ...*); dimenze často tvoří hierarchie (ve vertikálním členění) a časové či dimenzionální řady (v horizontálním členění), v nichž nás zajímají agregované hodnoty;
- **fakty** jsou ty atributy, které jsou hlavním důvodem evidence a jejichž hodnoty nás hlavně zajímají pro rozhodování (*množství, cena, zisk, ...*), a to obvykle ne v jednotlivých zdrojových entitách, ale agregované podle dimenzí; agregace je klíčovým pojmem pro analýzy a hlavním důvodem pro vznik DS;
- **atributy** jsou ostatní atributy, nepatřící k dimenzím ani faktům (*jméno, název firmy, číslo faktury, ...*), ty mají význam jen pro zdrojové entity, obvykle ne u odvozených výsledků DS. Mohou to být také popisy dimenzí nebo se z nich doplňují metadata.

Příklad:

Databáze firmy ABC pro skladovou evidenci nakupovaného a prodávaného zboží obsahuje tabulky

Sklad (sklad_karta, nazev_zbozi, id_dodavatel, cena_jedn, mnoz_aktual)
Nakup (sklad_karta, datum_nakup, mnoz_nakup, cena_jedn_nakup)
Prodej (sklad_karta, datum_prodej, mnoz_prodej, cena_jedn_prodej, id_zakaznik)
Zakaznik (id_zakaznik, nazev_firmy, jmeno_zastupce, obec, ulice, psc, stat, telefon)

K faktům, tedy atributům, jejichž sumované hodnoty nás budou především zajímat, budou patřit:

všechna množství – mnoz_aktual, mnoz_prijate i mnoz_vydane*, celkové ceny za nakoupené a prodané zboží, tedy odvozené údaje* cena_nakup = mnoz_nakup * cena_jedn_nakup, cena_prodej = mnoz_prodej * cena_jedn_prodej*; dále zřejmě bude zajímavý zisk z prodeje* zisk = cena_prodej – cena_nakup.

Dimenzemi budou základní atributy sklad_karta, id_dodavatel, datum_nakup, datum_prodej, id_zakaznik, obec nebo psc, stat *a případně další odvozené atributy z datumových atributů:* den. mesic, rok, den_tydne*, případně další. Podle každého z nich mohou být sumovány fakty množství, ceny, zisk.*

Zůstaly atributy nazev_zbozi, nazev_firmy, jmeno_zastupce, telefon*, které jen upřesňují informace o dimenzích.*

♦

Někdy rozdělení atributů není jednoznačné. Pro různé účely tentýž atribut může být jednou faktem, jindy dimenzí.

Příklad:

Atribut věk *je možno zařadit jako fakt* (*dává pro něj smysl počítat* min, ... avg) *v některé dimenzi (průměrný věk učitelů na škole v průběhu minulých let, minimální a maximální věk účastníků soutěže za minulá léta) apod.*

Jindy může být věk v roli dimenze (výkonnost věkových skupin žáků základní školy – tedy průměrné výkony podle věku, ...). *Odvozený atribut kategorizovaný věk dospělých nazvaný* věk_k *nabývající hodnot 1 = věk 18-23, 2 = věk 24-26, 3 = věk 27-30 atd. je již dimenzí jasnou dimenzí.*

♦

Poznámka: Pomocným návodem k tomu, jak rozeznávat atributy–fakty a atributy-dimenze je sestavení SQL dotazu – SELECTu s klauzulí GROUP BY:

Fakty jsou ty atributy, které se mohou agregovat (počítat, sumovat, průměrovat, ...), tedy dávají smysl za klauzulí SELECT v závorkách agregačních funkcí za

SELECT SUM(...), COUNT(...), AVG(...), MIN(...), MAX(...)

Dimenze jsou atributy, které dávají smysl za klauzulí GROUP BY, podle nich se fakty grupují a pak agregují.

Ostatní atributy jsou ty, kterými mohou být doplněny informace o dimenzích.

Příklad:

Je dáno relační schéma

Zkouška (**login, id-předmět, id-učitel, datum, pokus**, **body, známka**)

Dává smysl sestavit dotaz na počty zkoušek nebo počítání známek a bodů a jejich průměrování podle předmětů (průměrná známka, průměrný počet bodů za předmět), podle login = studentů, podle učitelů, případně podle datumu = termínu zkoušky nebo podle pořadí pokusu = na 1. termín, 2., 3.

SELECT COUNT(*), **AVG(body), AVG(známka), MIN(body), ..., MAX(body), ...**

FROM Zkouška
GROUP BY **id-předmět**
id-učit
login
...
id-předmět, pokus
...

Tedy zelené atributy jsou dimenze, červené jsou fakty.

Všimněme si, že nedává smysl počítat sumu = součet známek ani součet bodů pro žádnou dimenzi. K této vlastnosti se vrátíme u tzv. aditivity atributů níže.

♦

❑ Datová struktura datového skladu

Po rozdělení atributů na dimenze a fakty se modeluje struktura datového skladu, navrhují se struktury tabulek. Úkolem je vytvořit datovou strukturu tak, aby ke všem údajům, především k jejich sumám podle různých dimenzí, byl co nejjednodušší přístup.

Každá dimenze (dimenzionální atribut a jeho případné další atributy) tvoří jednu tabulku, nazvanou **dimenzionální tabulkou** či **D-tabulkou**. Dimenzionální tabulka musí mít jednoatributový klíč; pokud neexistuje přirozený klíč, definuje se umělý. Mezi dimenzemi se tak nepředpokládají žádné funkční závislosti.

Vazbu mezi dimenzemi tvoří informace o faktech: k n-tici dimenzí jsou připojeny příslušné fakty. V **ERD** je to tedy množina tabulek dimenzí s kardinalitou vazby mezi sebou typu M:N:K. Vazba se realizuje vazební tabulkou s použitím víceatributového klíče (jeden klíčový atribut za každou dimenzi) a fakty tvoří další atributy této vazby. Takovou tabulku nazýváme **tabulkou faktů** či **F-tabulkou**.

Kardinalita je jediné IO, ostatní jsou zaručeny zdrojovou databází a datovou pumpou.

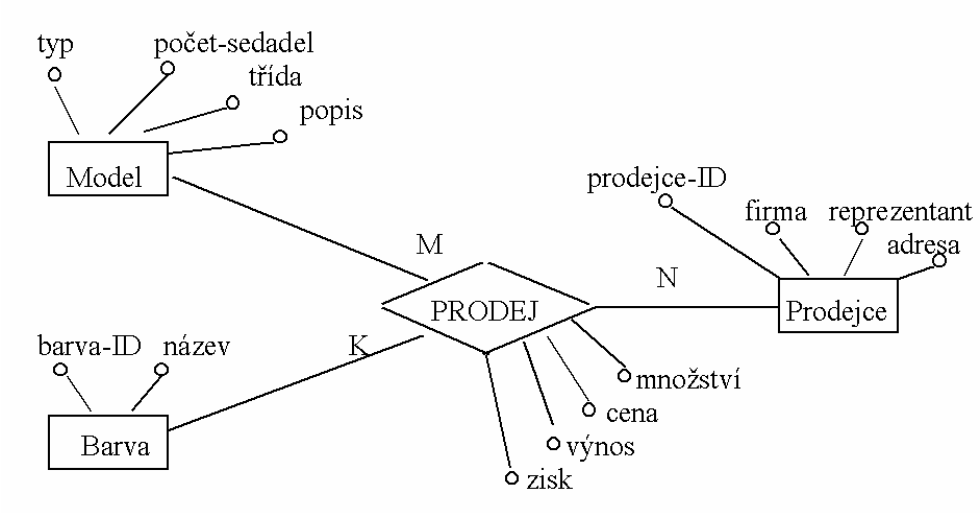
Příklad:

Část konceptuálního schématu zdrojové databáze prodeje aut má tabulky:

Auta (typ, pocet-sedadel, trida, barva, popis)
Prodejce (prodejce-id, firma, reprezentant, adresa)
Prodej (typ, prodejce-id, mnozstvi, cena)

V DS chceme evidovat jednotlivé prodejce, dále model auta, barvu, množství a cenu prodávaných aut a konečně výnos a zisk z prodeje.

Tedy existují dimenze Prodejce *(= prodejna),* Model *auta,* Barva *auta (všimněme si, že z atributu auta, který se stává dimenzí, bude samostatná D-tabulka) každá v jedné dimenzionální tabulce. Jednotlivé prodeje tvoří vazbu mezi dimenzionálními tabulkami s dalšími atributy-fakty, kterými jsou prodané* množství, cena *prodeje a další spočítané atributy-fakty* výnos a zisk *prodeje. ERD vypadá takto:*



ERD datového skladu PRODEJ

Už si dovedeme představit, že zobrazená vazba se může realizovat vazební tabulkou mezi D-tabulkami, kde každá dimenze je zastoupena svým klíčem a fakty jsou dalšími atributy této vazby:

Prodej (prodejce-id, typ, barva-id, mnozstvi, cena, vynos, zisk)

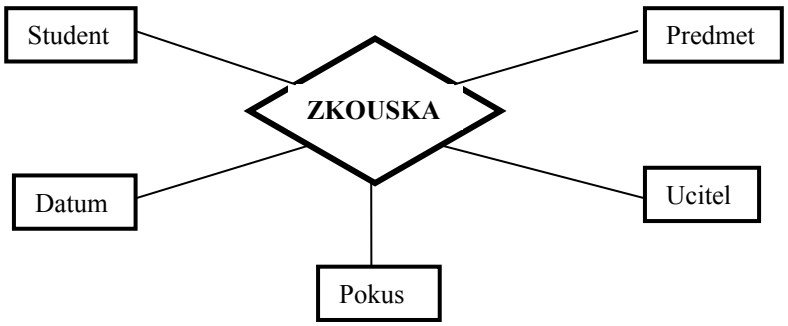
♦

Příklad:

Mějme znovu relační schéma zdrojové databáze

Zkouška (login, id-predmet, id-ucitel, datum, pokus, body, znamka)

s fakty body a znamka *a dimenzemi* login, id-předmět, id-učitel, datum, pokus. *Ve zdroji je to tedy jediná tabulka, ERD v datovém skladu bude*



a při realizaci vazby tabulkou to bude 6 tabulek, každé z 5-ti dimenzí jedna D-tabulka a jedna F-tabulka s 5-atributovým klíčem a dvěma atributy-fakty znamka a body.

♦

Dva přístupy k procesu agregování dat

Už jsme uvedli několikrát, že z DS zajímají uživatele především údaje sumované nebo jinak agregované. Agregacemi chápeme především hodnoty klasických agregačních funkcí například z SQL jazyka, tedy počty COUNT, součty SUM, průměry AVG, minima a maxima MIN a MAX.

Dalším problémem k vyřešení v datových skladech je to, jak a kdy se budou požadované agregované údaje počítat.

1. První možností je ponechat jen atomická data v relacích ve 3NF a při každém dotazu uživatele provádět operativně požadované agregace a další zpracování nad daty. Tento přístup má následující výhody a nevýhody:

- při velkých objemech dat je zpracování neúnosně pomalé a náročné na HW i v samostatném DS,
- při opakovaných dotazech se pokaždé znovu počítají stejné sumy,
- v DS se ale ukládají jen základní data a tedy je malá spotřeba paměťové kapacity.

2. Druhou možností je data **předpočítat a uchovat všechny možné agregace** pro všechny definované dimenze a jejich hierarchické stupně. Výhody a nevýhody tohoto přístupu:

- při dotazech se požadované údaje nepočítají, jen vyhledají a zobrazí nebo vhodně vizualizují například formou grafů;
- tento postup je náročnější na předběžnou analýzu budoucích možných požadavků, takže vyžaduje modelování dimenzionální, které již nedodržuje 3NF, protože obsahuje údaje odvozené;
- data obsahují redundance dvojího typu: odvozené údaje na atomické úrovni, aby se nemusely stále znovu počítat při opakovaných dotazech a sumované údaje ze stejného důvodu;
- je vyžadována optimalizace přístupu k datům vzhledem k novým typům analýz, tedy je nutný jiný typ řízení práce s DS než u operativního informačního systému s databází.

Druhá varianta je výhodnější, rychlejší při provozu, a proto se u DS používá.

Hierarchie dimenzí

Dimenzionální atributy někdy nejsou jen jednoduché, ale mohou tvořit celé hierarchie. Buď se v atomických datech vyskytují na nejnižší úrovni hierarchie a jako další doprovodné atributy mají své vyšší úrovně, nebo se vyšší úrovně dají ze základních údajů dopočítat – odvodit.

Příklad:

Velmi často se vyskytující a užitečnou hierarchii tvoří datumové údaje. V příkladě PRODEJ aut bude navíc proti minulému příkladu jednou z dimenzí datum *prodeje se základní D-tabulkou* Datum (id-datum, datum).

Mimo sumy prodaných aut za jednotlivé dny budou jistě zajímavé i sumy za týden, měsíc, rok, možná i jiné – čtvrtletí apod. Z datumu lze tyto další údaje spočítat. Podle pravidel DS se spočítají předem a uloží do D-tabulky.

Odpovídající hierarchii můžeme znázornit takto: den – týden – měsíc – kvartál – rok.

Zatím si představíme všechny odvozené údaje jako další atributy D-tabulky Datum.

♦

Příklad:

Jiným příkladem je také často se vyskytující a užitečná hierarchie územního rozložení. V příkladě PRODEJ aut si rozložíme atribut adresa *na atributy* ulice, cislo, obec, psc, stat *a už tušíme hierarchii:* ulice – psc – obec – okres – kraj – stat*, kde okres a kraj se zřejmě dá získat z vhodných číselníků podle obce a psc. PSČ jak víme odpovídá poště, tedy u velkých obcí může jít o část obce.*

Mimo sumy prodaných aut za jednotlivé prodejce pak budou jistě zajímavé i sumy za obec, okres, kraj, stát. Opět si zatím představíme všechny odvozené údaje jako další atributy D-tabulky Prodejce.

♦

❑ Aditivita atributů

Máme definovány dimenze a fakty a očekáváme dotazy na sumy a jiné agregované hodnoty faktů podle jednotlivých dimenzí nebo jejich kombinací *(například suma prodaných aut a jejich cena, ... podle měsíců v jednotlivých krajích)*. Máme-li několik dimenzí, pak i jednoduchých agregovaných hodnot je velké množství, přidáme-li všechny kombinace dimenzí, je jich ještě násobně více.

Princip DS spočívá v tom, že se předpokládají jakékoliv smysluplné dotazy na kterékoliv možné agregované hodnoty – i když to uživatel předem nepožadoval (dotazy ad hoc = dotazy přinesené životem). Teoreticky tedy můžeme předpokládat dotazy na všechny fakty podle všech dimenzí a všech jejich kombinací podle všech agregačních funkcí COUNT, SUM, AVG, MIN, MAX.

Přitom jsme již u příkladu tabulky Zkouška viděli, že některé tyto hodnoty v realitě nedávají smysl *(například součet známek za zkoušky)*. Je tedy potřeba definovat další integritní omezení, která omezí množinu možných dotazů nad DS – aditivitu atributů.

Definice:

Aditivita faktu znamená smysluplnost agregací jeho hodnot vzhledem ke všem dimenzím.

Platí:

- Fakty jsou numerické hodnoty a teoreticky mohou být pro skupiny řádků sumarizovány.
- Pro jednoduchost se v tabulce faktů implicitně tato vlastnost předpokládá.
- Jakákoliv výjimka je explicitně zadána ve schématu.
- Někdy se rozlišují fakty semiaditivní a neaditivní.

Definice:

Atribut v tabulce faktů se nazývá **semiaditivní**, jestliže není aditivní vzhledem k jedné nebo více dimenzím. **Neaditivní** atributy nejsou aditivní vzhledem k žádné dimenzi.

Rozdělení faktů lze tedy rozdělit dle adivitity na

- **aditivní** - agregace bez omezení, možné všechny typy agregace pro všechny dimenze
- **semiaditivní -** částečně aditivní, smysl mají jen některé agregace pro některé dimenze
- **neaditivní -** nemá smysl je agregovat podle žádné dimenze

Součástí definice aditivity atributů-faktů bude i tzv. omezení na dotazy, jehož zápis si definujeme v následující kapitole.

9.4. Databázové modelování

ROLAP a MOLAP

V DS se používá předpočítání a uchování agregovaných dat. Víme, že jde o další redundanci v DS, ale vzhledem k jeho využití tato redundance nehrozí nekonzistencí a naopak urychluje odezvu uživateli – jen předpočítané údaje vybírá, už je nemusí počítat.

Ani tento přístup však není jednoduchý, nastávají nové problémy s tím, jak agregovaná data a jejich hierarchie modelovat na různých úrovních.

V úrovni databázové se používají prakticky dva **základní přístupy** a některé jejich varianty:

- **ROLAP** (Relační OLAP) znamená implementaci DS pomocí relačních tabulek (tabulky dimenzionální a tabulky faktů) organizovaných do hvězdicových schémat.
- **MOLAP** (Multidimenzonální OLAP) implementuje DS pomocí hyperkostky nebo multidimenzionálních kostek. Též se označuje POLAP (proprietární OLAP). Tato technologie musí být zabudována v SŘBD.
- HOLAP (Hybridní přístup ROLAPu a MOLAPu)
- DOLAP (Desktop OLAP, datový sklad na klientském počítači)

Databázista si lehce představí realizaci schématu v relačním datovém modelu. Bohužel pro multidimenzionální analýzu by byly výsledné normalizované tabulky příliš těžkopádné. Tabulky agregovaných hodnot by se při každé změně základních dat musely počítat. Počet stupňů hierarchie dimenzí je pevný z analýzy a není snadné rychle dynamicky měnit zobrazování sum „nahoru a dolů“ podle těchto hierarchií.

Efektivnější je uložení databáze pomocí multidimenzionální kostky. Kostka má tolik dimenzí, kolik je zapotřebí zaznamenat atributů = dimenzí, každé pole v této kostce obsahuje odpovídající hodnoty atributů = faktů. Na první pohled jde o vícerozměrné programátorské pole (array). Ovšem zde bude mnoho kombinací hodnot dimenzí prázdných, pole bude řídké a proto se používají speciální implementační techniky pro realizaci dimenzionálních kostek. Princip kostky dále pak vede k jednoduchému přístupu při výpočtu různých agregací.

Probereme si oba přístupy podrobněji.

9.5. Model datového skladu v ROLAP

Hvězdicové schéma (hvězda, star)

Jak už tušíme z předcházejícího výkladu, k realizaci datového skladu pomocí relačního SŘBD se používá speciálních sestav relací (tabulek), nazývaných hvězda. Definujme si její dosud jen intuitivně popsanou datovou strukturu.

Definice:

Hvězdicové schéma **S** je trojice <**D**, F, **CC**>, kde **D** je množina dimenzionálních tabulek, F je tabulka faktů a **CC** je množina kardinalit.

Hvězda zobrazuje realitu z ERD modelovanou pomocí vazební tabulky, nazývané zde jednou tabulkou faktů F, množiny ostatních n tabulek dimenzí **D** a množinou kardinalit **CC**. Vybraný atribut z každé dimenzionální tabulky D je klíč a je označen KD. Klíčem tabulky F je sjednocení $\cup_{i=1..n}$ KD_i. Neklíčové atributy z F jsou fakty.

Definice:

Kardinalita CC_i je definována pro F a dimenzi D_i takto: je-li F* tabulka faktů a D* dimenzionální tabulka, pak CC_i je splněno, jestliže pro každý řádek *u* z F* existuje pouze jeden řádek *v* v D* takový, že $u.KD_i = v.KD_i$.

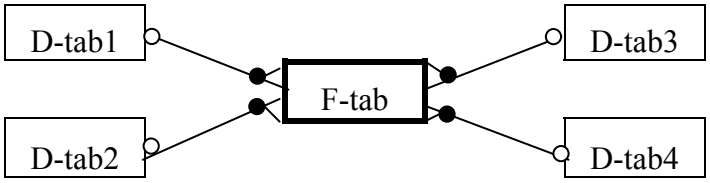
Tedy řádky tabulky faktů a řádky každé dimenze jsou ve vztahu N:1. KD je v F cizím klíčem, proto nesmí být NULL. Jinak řečeno, na straně faktu je povinné členství N:1 (právě 1).

Definice:

Nechť S je hvězdicové schéma. Pak **multidimenzionální databází** nad S nazýváme množinu tabulek D^*_i , i=1..n a F*, které vyhovují kardinalitám z **CC**.

Platí:

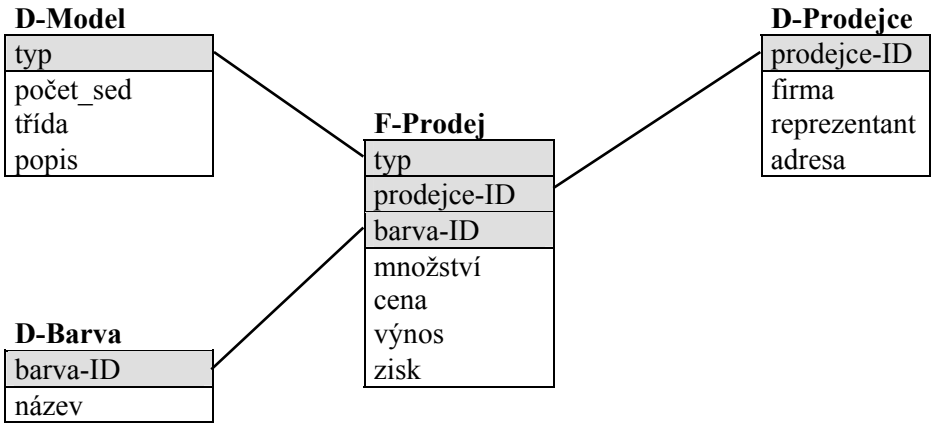
- každá dimenze Di má 1 klíč KDi (třeba umělý), je reprezentována **dimenzionální tabulkou Di**
- existuje 1 **tabulka faktů F**, jejíž klíč tvoří cizí klíče KDi všech souvisejících dimenzí Di, její ostatní atributy jsou fakty f1, f2, …
- celé struktuře **F** a **{Di}** se říká **multidimenzionální databáze.**



Obecné hvězdicové schéma

Příklad:

Již známý příklad datového skladu PRODEJ *aut by se tedy modeloval pomocí vazební tabulky = F-tabulky a několika D-tabulek následovně:*



Příklad hvězdicového schématu Prodej

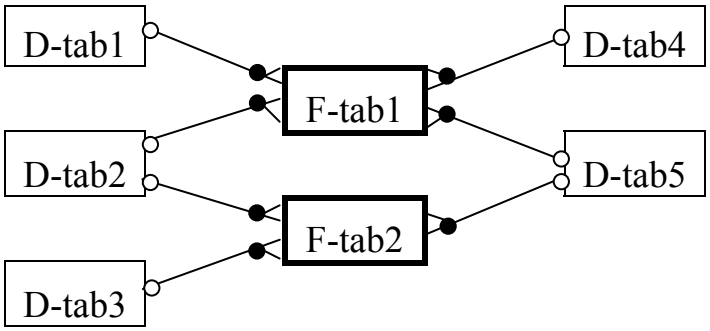
♦

❑ Schéma souhvězdí

Datový sklad s jedním hvězdicovým schématem je jednoduchý, ale obvykle nestačí na všechny potřeby skladování. Často je potřeba v DS zaznamenat více tabulek faktů. Přitom některé dimenze pro různé F-tabulky mohou být společné, některé jsou rozdílné. Obecně lze tedy rozšířit princip hvězdy na **schéma souhvězdí**.

Definice:

Schéma souhvězdí je trojice <**D**, **F**, **CC**>, kde **D** je množina dimenzionálních tabulek, **F** je množina schémat tabulek faktů a **CC** je množina kardinalit. Pro každé F ∈ **F** existuje podmnožina D' ⊆ **D** a CC' ⊆ **CC** tak, že <**D'**, F, **CC'**> je hvězdicové schéma.

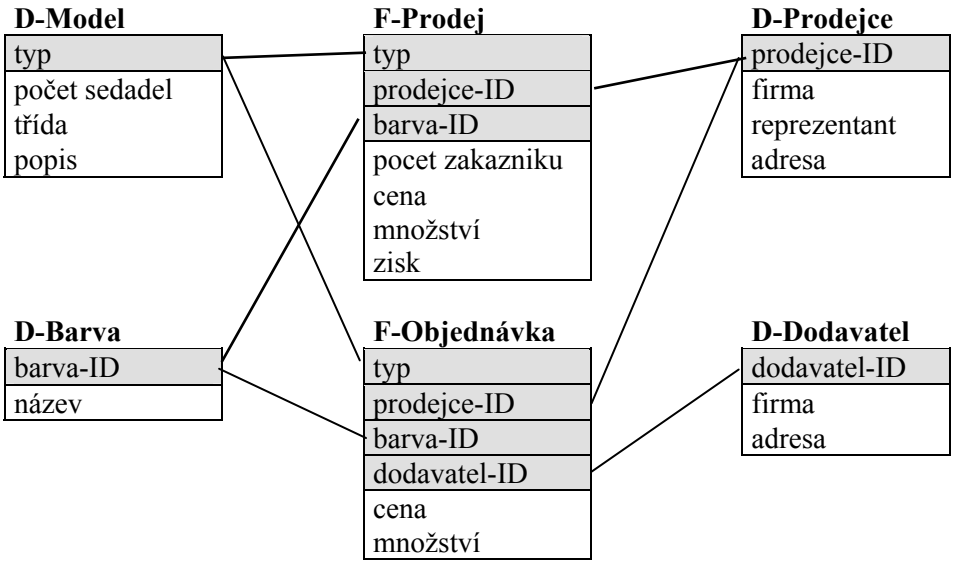


Obecné schéma souhvězdí

Příklad:

Dosud uváděný příklad datového skladu PRODEJ *aut je zjednodušený. Dovedeme si představit, že uživatel bude mít zájem sledovat nejen průběh prodeje, ale i nákupu aut, objednávek aut od dodavatelů a případně dalších jevů.*

Následující schéma zobrazuje souhvězdí s F-tabulkami Prodej a Objednávka*. Přitom využívají některé společné dimenze.*



Příklad souhvězdí tabulek faktů Prodej a Objednavka

♦

❑ Omezení na dotazy

Obecně v DS předpokládáme využití všech smysluplných agregací faktů vzhledem ke všem dimenzím a jejich kombinacím, a to podle všech agregačních funkcí COUNT, SUM, AVG, MIN, MAX. Již ale jsme si všimli, že někdy některé agregace nemají reálný smysl, byť se dají spočítat *(například u uvedeného příkladu* Zkouska *lze spočítat součet známek, ale smysl toto číslo nemá)*.

Většina agregovaných hodnot smysl má a v souladu s posláním datového skladu všechny tyto agregace se budou počítat, byť to zadavatel přímo nespecifikoval. DS tak je připraven i na všechny neočekávané dotazy = ad hoc dotazy = dotazy přinesené životem a předem nepředpokládané. Proto se jako nová integritní omezení zadají ty agregace, které v realitě nemají smysl a proto se ani nebudou počítat. Formálně je nazveme omezení na dotazy a rozšíříme o ně definované hvězdicové schéma.

Definice:

Množina omezení na dotazy **DO** nad schématem souhvězdí <**D, F, CC**> je množina trojic (F, D, **Agg**), kde F je fakt z **F**, D je dimenze a **Agg** je množina agregačních funkcí, pro které nemá smysl agregovat F vzhledem k dimenzi D.

DO = **{(Fi, Dj, Ag1,Ag2,...), (Fk, Dl, Ag1, Ag3, ...), ... }**

Interpretací každé trojice je tedy negativní informace, že funkce z Agg **nelze použít** na atribut A vzhledem k dimenzi D. Schéma souhvězdí bude čtveřice <**D, F, CC, DO**> s již známým významem.

Příklad:

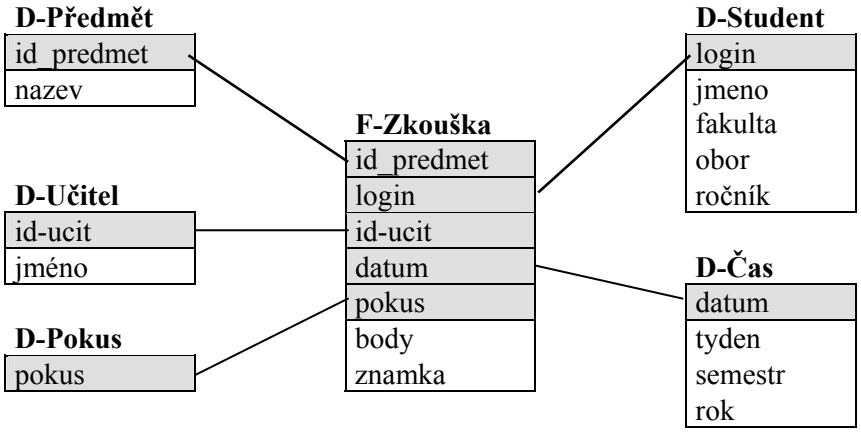
Je opět dáno schéma Zkouška (login, id-předmět, id-učitel, datum, pokus, body, známka).

*Zvažme smysl všech agregovaných údajů v dotazu následujících typů (**pozor**, není úkolem zvážit, na co se asi bude nebo nebude uživatel ptát, ale která hodnota nemá jako číslo smysl):*

SELECT COUNT(*), SUM(body), SUM(znamka), AVG(body), AVG(znamka), MIN..., MAX...
FROM Zkouska
GROUP BY id-predmet
id-ucit
login
id-předmět, pokus **...**

Zřejmě má smysl počítat: počet zkoušek za předmět, učitele, studenta, datum, pokus
průměr, min, max bodů i známek za všechny dimenze

Ale nemá smysl počítat: součet známek pro žádnou dimenzi
součet bodů pro některé dimenze



Závěr: *atributy body a známka jsou semiaditivní, omezení na dotazy bude*

DO = {(známka, Předmět, SUM), (známka, Učitel, SUM), ...}

♦

9.6. Hierarchie dimenzí v ROLAP

Modelování dimenzionální hierarchie

V neformálním popisu jsou dimenze popisovány pomocí svého klíče a popisných atributů (jméno, popis,...). V některých dimenzích mohou být skryty **dimenzionální hierarchie,** atributy v hierarchii se nazývají členy hierarchie. Obecněji mohou být členy hierarchie typy entit s vlastními atributy.

Příklad:

V DS PRODEJ má dimenze Model *atributy* typ *(klíč),* počet_sedadel, třída, popis.

Dimenzionální atributy model, datum, prodejce *databáze* PRODEJ *mají hierarchie*

Model: zboží - třída - vše

Některé dimenze mohou mít dokonce více než jedinou hierarchii, typickým příkladem je
Datum: den - měsíc - kvartál - rok - vše
den - týden ---------- vše
den - den v týdnu ---------- vše

Prodejce: adresa - obec - okres - kraj – stát - vše

Na konceptuální úrovni tvoří hierarchie řetězec typů entit, kde kardinalita dvou sousedů je 1:N.

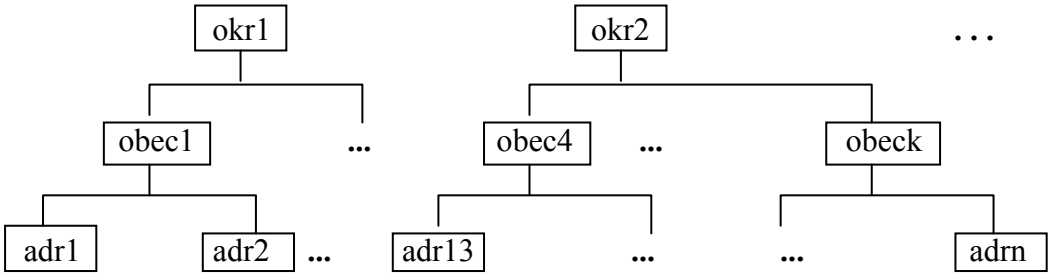
♦

Typické použití pro dimenzionální hierarchie jsou agregace. V hierarchii lze při požadavku postupovat o úroveň výše (roll up) nebo o úroveň níže (drill down), případně postupovat vodorovně podél **dimenzionální řady.** Častým případem dimenzionálních řad jsou **časové řady** – agregované hodnoty faktů za jednotlivé dny nebo týdny nebo měsíce atd.

Rozšířením hierarchie může být množina stromů, kde list každého stromu obsahuje data o jednotlivých entitách.

Příklad:

Část hierarchického stromu dimenze adresa – obec – okres*, vodorovně jsou dimenzionální řady.*



♦

Při modelování hierarchií dimenzí nastává nový problém, **jak hierarchické stupně v DS uložit.** Používají se k tomu v RDM následující techniky.

❑ Hierarchie dimenzí v jedné tabulce faktů

Nejjednodušším způsobem z hlediska počtu použitých tabulek je následující:

1F = 1 tabulka faktů pro všechny hierarchické stupně a 1D = 1 tabulka pro každou dimenzi

Nejmenší prostor pro data spotřebuje

- umístění celé hierarchie dimenze do jedné D-tabulky (nenormalizovaná, s velkou redundancí a umělou identifikací)
- celá hierarchie je chápána jako doména v jedné D-tabulce

Celý DS se realizuje jako hvězda nebo souhvězdí s jednou F-tabulkou se zdrojovými i agregovanými údaji o všech stupních hierarchie.

Pro rozlišení hodnot dimenzí i jejich hierarchických stupňů se užívají dvě metody:

A. záznamy každé dimenze i její hierarchické stupně mají generovaný klíč

Příklad:

D-tabulka Prodejce*, hierarchické stupně jsou barevně rozlišeny, umělý klíč je automaticky generován nejen na nejnižší úrovni, ale i pro všechny agregační úrovně.*

gen_klíč	prod_id	adresa	reprez	obec	kraj	úroveň
...						
233	prod_123	A	Horák	Ostrava	sever_mor	adresa
234	prod_234	B	Janák	Ostrava	sever_mor	adresa
235	prod_345	C	Hever	Brno	jih_mor	adresa
236	prod_456	D	Novák	Opava	sever_mor	adresa
237	prod_567	E	Kovář	Znojmo	jih_mor	adresa
238	NULL	NULL	NULL	Opava	sever_mor	obec
239	NULL	NULL	NULL	Ostrava	sever_mor	obec
240	NULL	NULL	NULL	Brno	jih_mor	obec
241	NULL	NULL	NULL	Znojmo	jih_mor	obec
242	NULL	NULL	NULL	NULL	jih_mor	kraj
243	NULL	NULL	NULL	NULL	sever_mor	kraj
244	NULL	NULL	NULL	NULL	NULL	všechno

D-tabulka Prodejce se skrytou hierarchií a generovanými klíči

Obdobně mají generovaný klíč dimenze barva a model*. Pak tabulka faktů obsahuje všechny stupně hierarchie všech dimenzí i jejich kombinací:*

typ	prod_ID	barva_ID	poč_zákaz	množ	cena	zisk
...						
1	233	333	2	3	200000	20000
1	234	333	3	3	300000	30000
1	235	333	5	7	500000	50000
1	236	333	12	14	1200000	120000
1	237	333	8	10	800000	80000
1	238	333	5	6	500000	50000
1	239	333	5	7	500000	50000
1	240	333	12	14	1200000	120000
1	241	333	8	10	800000	80000
1	242	333	10	13	1000000	100000
1	243	333	20	24	2000000	200000

F-tabulka Prodej s hierarchií podél dimenze Prodejce

B. záznamy každé dimenze mají samoidentifikovatelný klíč

Příklad:

Jiná varianta používá nehomogenní klíč, který současně definuje hierarchickou úroveň.

dim_klíč	adresa	reprez	obec	kraj	úroveň
prod_123	A	Horák	Ostrava	sever_mor	adresa
prod_234	B	Janák	Ostrava	sever_mor	adresa
prod_345	C	Hever	Brno	jih_mor	adresa
prod_456	D	Novák	Opava	sever_mor	adresa
prod_567	E	Kovář	Znojmo	jih_mor	adresa
Opava	NULL	NULL	NULL	sever_mor	obec
Ostrava	NULL	NULL	NULL	sever_mor	obec
Brno	NULL	NULL	NULL	jih_mor	obec
Znojmo	NULL	NULL	NULL	jih_mor	obec
sever_mor	NULL	NULL	NULL	NULL	kraj
jih_mor	NULL	NULL	NULL	NULL	kraj

Dimenzionální tabulka Prodejce se samoidentifikovatelnými dimenzionálními klíči

typ	prod_ID	barva_ID	poč_zákaz	množ	cena	zisk
1	prod_123	333	2	3	200000	20000
1	prod_234	333	3	3	300000	30000
1	prod_345	333	5	7	500000	50000
1	prod_456	333	12	14	1200000	120000
1	prod_567	333	8	10	800000	80000
1	Opava	333	5	6	500000	50000
1	Ostrava	333	5	7	500000	50000
1	Brno	333	12	14	1200000	120000
1	Znojmo	333	8	10	800000	80000
1	sever_mor	333	10	13	1000000	100000
1	jih_mor	333	20	24	2000000	200000

Tabulka faktů Prodej s hierarchií podél dimenze prodejce

Výhody této realizace jsou

- malý počet tabulek: existuje jen jedna tabulka faktů se zdrojovými fakty i s agregovanými hodnotami, rozlišují se klíčovými hodnotami dimenzí,
- F-tabulka může obsahovat nejen agregace podle jednoduchých dimenzí, ale i podle jejich libovolných kombinací.

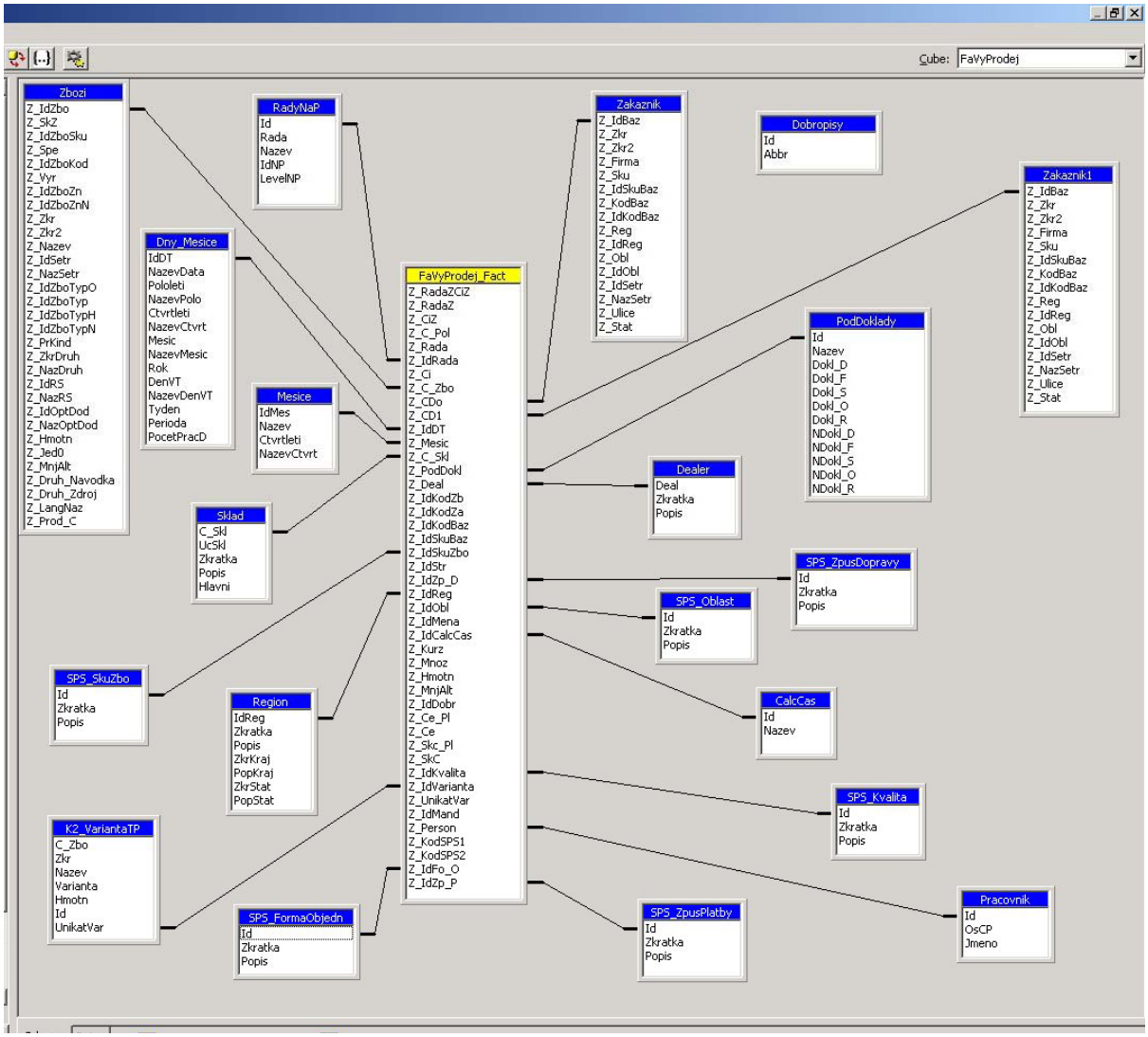
Nevýhody

- tabulka faktů je velmi rozsáhlá a vyhledání odpovídající množiny řádků je tak zpomalováno.

Příklad:

Na následujícím obrázku je příklad hvězdicového schématu skutečného datového skladu firmy. Vidíme, že počet dimenzí může být vysoký. Tento sklad je implementován v MS SQL serveru 2005 a schéma je také z tohoto prostředí.

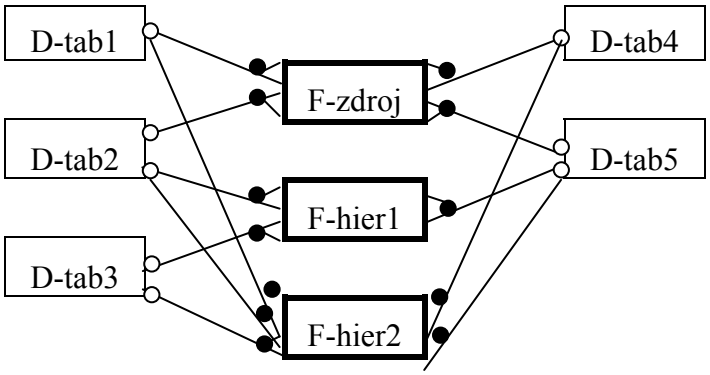
Všimněme si, že se v dimenzionálních tabulkách vyskytují hierarchie dimenzí. Typicky například u časových údajů Dny_mesice *nebo u* Zbozi.



❑ Hierarchie dimenzí pomocí souhvězdí F-tabulek

Druhou možností je rozdělení tabulky faktů – F-tabulka není jedna, ale původní je vodorovně „rozsekána“, je nová pro každý stupeň hierarchie. To znamená, že pro každou dimenzi tvoří F-tabulky řetězec tabulek. Dimenzionální tabulky obsahují (jako výše) i hierarchii dimenze.

Obecné schéma



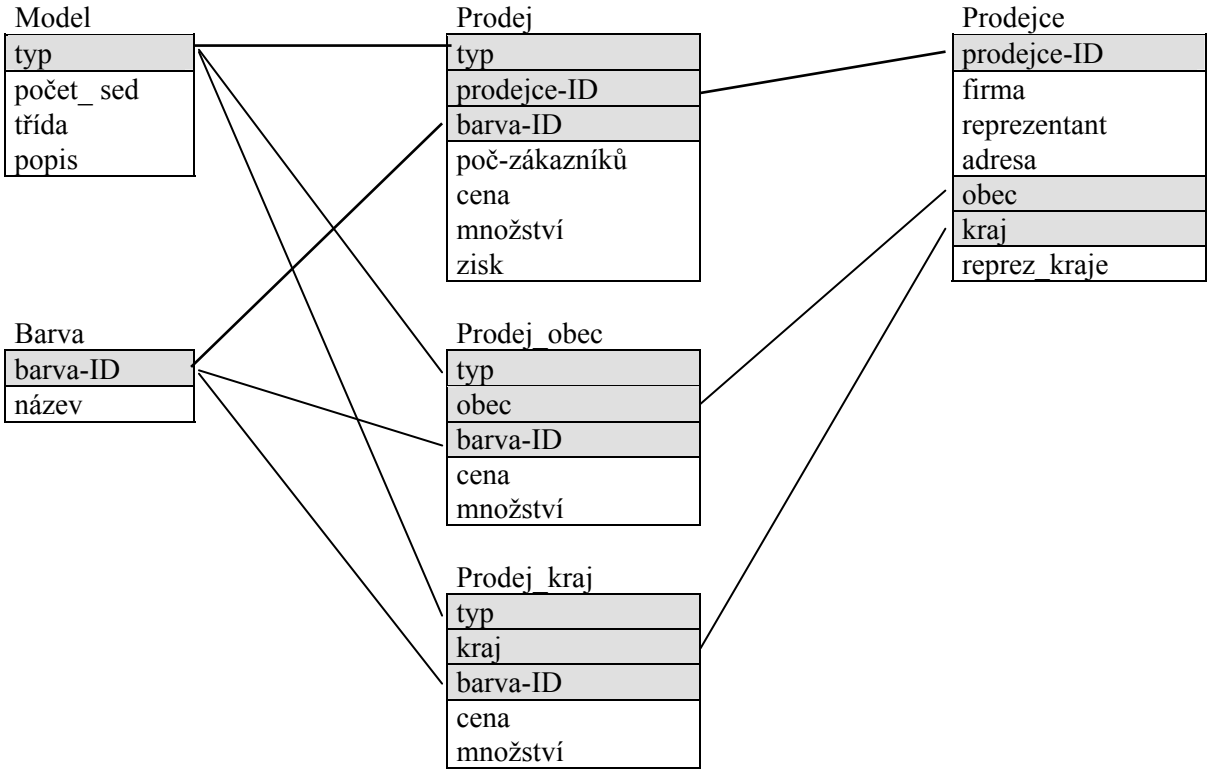
Obecné schéma souhvězdí pro hierarchii faktů (pro jednu základní F-tabulku)

Schéma souhvězdí je zde využito v jiném smyslu než výše – pro hierarchické řetězce tabulek faktů. Tato řešení vytváří mnoho nových tabulek faktů, což je náročné pro konstruktéra DS – při realizaci dotazu uživatele je nutno vybírat ze správných tabulek. Ovšem pak se většinou použijí data celé tabulky, není nutno je dohledávat..

U tohoto i následujících řešení obsahují takto odvozené (agregované) tabulky faktů redundandní informaci, sloužící ke zrychlení odezvy na dotazy uživatelů.

Příklad:

Opět schéma PRODEJ*, ale předpočítané agregované (sesumované) údaje o počtu zákazníků, ceně, prodaném množství a zisku jsou uloženy v samostatných tabulkách faktů. Pro každou úroveň dimenze nebo kombinace dimenzí je jedna nová tabulka faktů.*



Příklad hierarchie dimenzí pomocí souhvězdí tabulek faktů

♦

Příklad:

Část vyplněných tabulek faktů Prodej *s rozdělením F-tabulek podle úrovní.*

Prodej - *tabulka faktů s generovaným klíčem dimenze* Prodejce

typ	prod_ID	barva_ID	poč_zákaz	množ	cena	zisk
1	233	333	2	3	200000	20000
1	234	333	3	3	300000	30000
1	235	333	5	7	500000	50000
1	236	333	12	14	1200000	120000
1	237	333	8	10	800000	80000

Prodej_obec - *tabulka faktů s generovaným klíčem dimenze* Prodejce

typ	obec	barva_ID	poč_zákaz	množ	cena	zisk
1	238	333	5	6	500000	50000
1	239	333	5	7	500000	50000
1	240	333	12	14	1200000	120000
1	241	333	8	10	800000	80000

Prodej_kraj - *tabulka faktů s generovaným klíčem dimenze* Prodejce

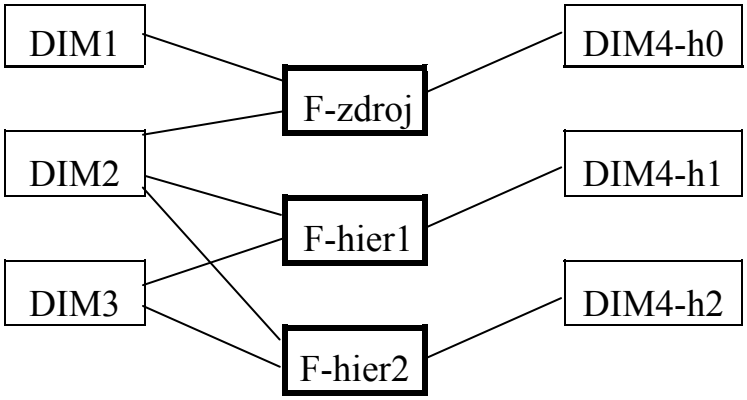
typ	kraj	barva_ID	poč_zákaz	množ	cena	zisk
1	242	333	10	13	1000000	100000
1	243	333	20	24	2000000	200000

❑ Hierarchie dimenzí pomocí sněžení (sněhových vloček)

Alternativou předcházejícího řešení je rozložení hierarchie nejen u faktů, ale i u dimenzí do více tabulek:

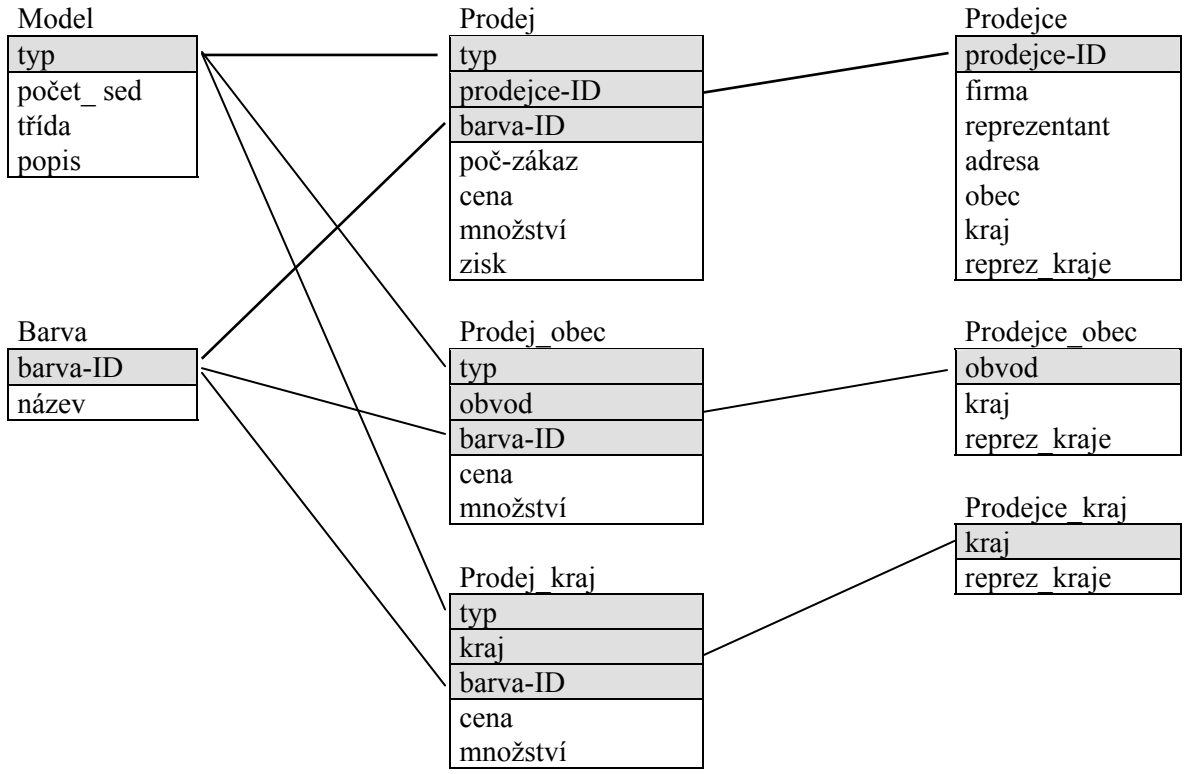
- nejen F-tabulky, ale i D-tabulky jsou rozloženy dle hierarchických stupňů
- každý hierarchický stupeň tabulky faktů má vazbu na příslušný hierarchický stupeň dimenze
- klíče hierarchických stupňů ukazují do menších dimenzionálních tabulek.

Obecné schéma



Obecné schéma sněhové vločky pro hierarchii faktů i hierarchii dimenzí

Příklad:



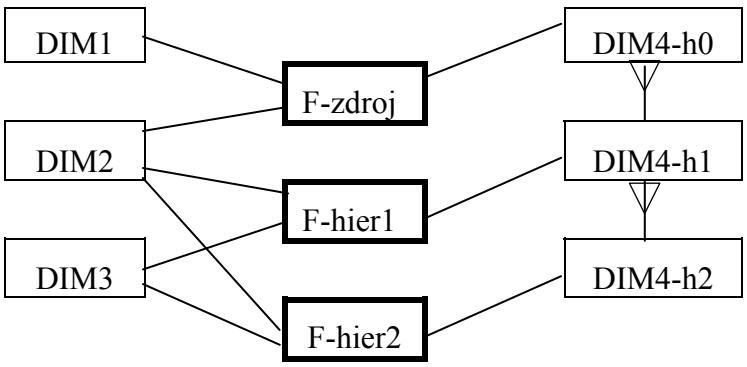
♦ Příklad sněhových vloček podél dimenze Prodejce

❑ Hierarchie dimenzí - souhvězdí s explicitními hierarchiemi

Teoreticky nejlépe vyřešená struktura s konceptuální úrovní přehlednou a přesnou je tato:

- tabulky dimenzí jsou rozloženy podle hierarchických stupňů s explicitní definicí hierarchie pomocí vazeb mezi hierarchickými stupni,
- pro každou dimenzi a její stupeň upřesňuje množinu relevantních faktů, zachovává normalizaci (redundance je skryta v opakování agregované informace v redundandních tabulkách faktů).
- Množství F-tabulek a D-tabulek s explicitními vazbami hierarchie dimenzí však kladou velké nároky na řešitele – aplikačního programátora datového skladu při realizaci dotazů.

Obecné schéma

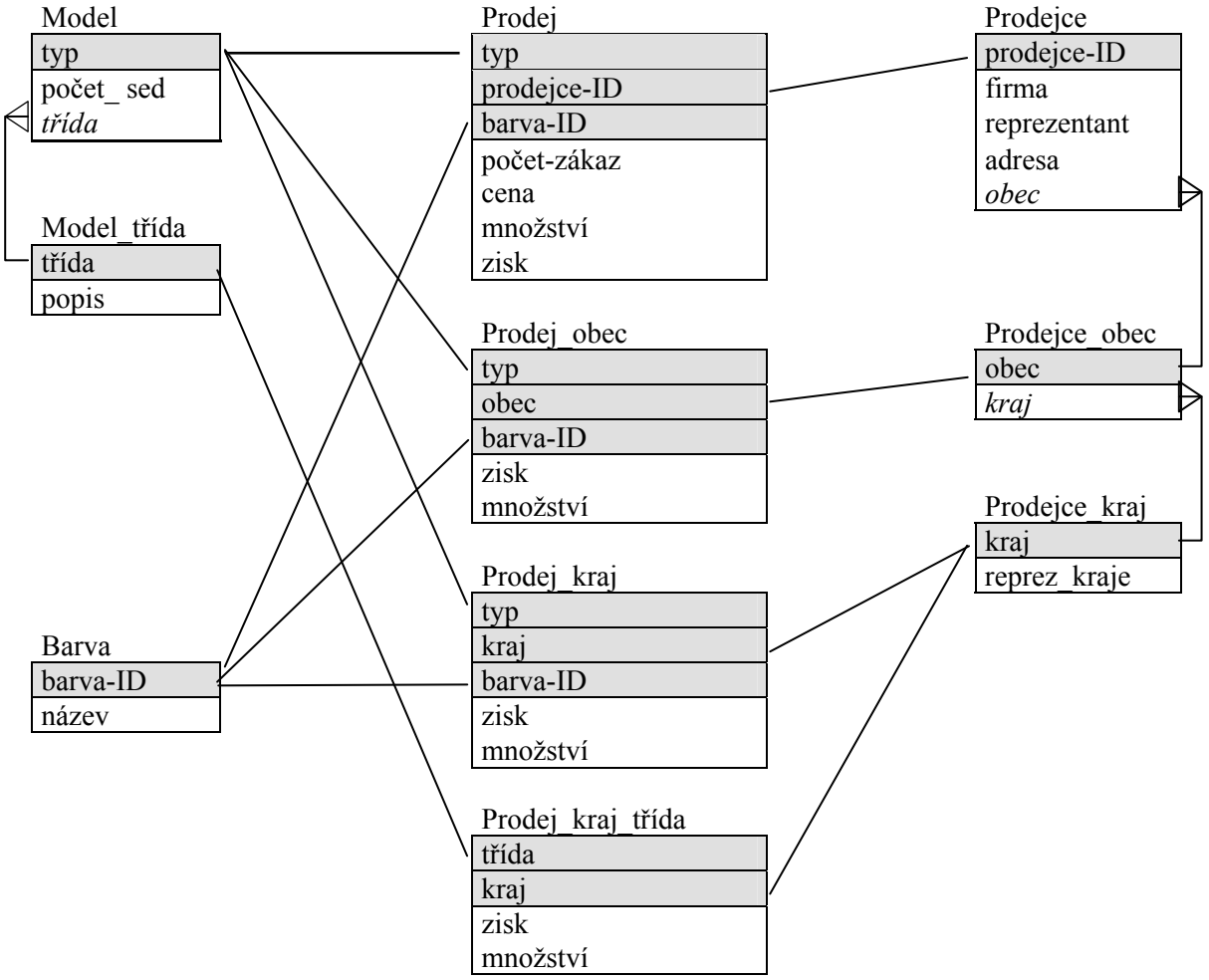


Obecné schéma souhvězdí pro hierarchii dimenzí 1:M

Příklad:

D-tabulka Prodejce *je normalizovaná na 3 tabulky s vazbou 1:M, a to na základní prodejnu, prodejny obce a prodejny kraje.*

Obdobně D-tabulka Model *auta je rozdělena na 2 s vazbou 1:M, a to na konkrétní typ auta a na třídu aut.*



♦ Příklad explicitní hierarchie podél dimenzí Prodejce a Model

9.7. Dimenze a jejich hierarchie v MOLAP

❑ Hyperkostka a multikostky

Místo databázového modelu DS pomocí množství tabulek se užívá také multidimenzionální databáze. Jde o zobecněný princip excelovských tabulek, kde je možno nastavit počítání řádkových a sloupcových sum, případně dalších agregovaných hodnot. Místo dvourozměrné tabulky je datový sklad realizován tzv. multikostkou, tedy m-dimenzionální „tabulkou“, která má obdobné vlastnosti jako uvedená excelovská dvourozměrná tabulka. Do základních polí kostky se ukládají hodnoty faktů nejnižších úrovní. Protože faktů v jedné F-tabulce bývá více, ukládají se do těchto polí případně celé n-tice faktů. Přitom každý rozměr tabulky odpovídá jedné dimenzi.

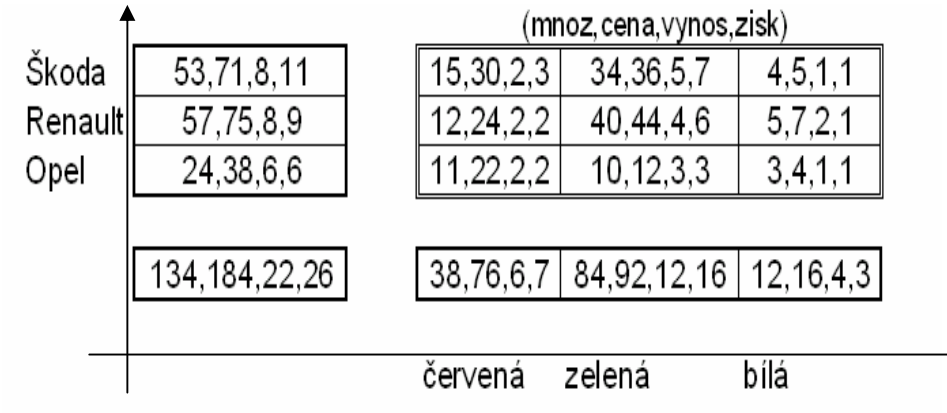
Definice:

Multidimenzionální databáze neboli **hyperkostka** je m-rozměrný prostor dimenzí, nad nímž existuje n-rozměrný prostor faktů. Nad nimi mohou existovat k-rozměrné prostory agregovaných dimenzí.

Hyperkostka obsahuje agregované (v tomto případě sumované) údaje, tedy data odvozená. Dimenze tvoří osy kostky, popsané každá jednoatributovým klíčem, n-tice faktů jsou uloženy v jednotlivých polích kostky. Jinak formulováno, hodnoty n-tice faktů závisí na příslušné m-tici dimenzí.

Příklad:

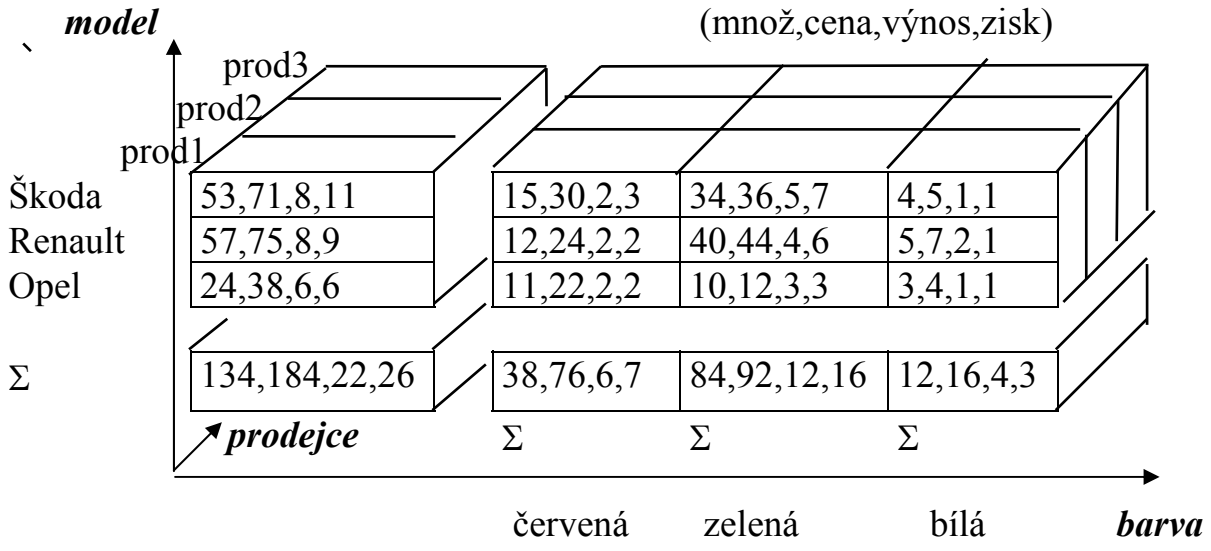
Velmi jednoduchý příklad prodeje pouze pro 2 dimenze Model a Barva. *Hyperkostka je jen 2-rozměrná tabulka, tedy 2-rozměrný prostor se 4-rozměrným vektorem faktů.*



♦

Příklad:

Pro 3 jednoduché dimenze Model, Barva *a* Prodejce *tvoří hyperkostka 3-rozměrný prostor, kde hodnoty3 dimenzí tvoří 3 souřadnice*



3-rozměrná hyperkostka se 4-rozměrným vektorem faktů

♦

Model kostky umožňuje zaznamenat hodnoty faktů pro **všechny kombinace hodnot dimenzí** (prvky kartézského součinu odpovídajících domén dimenzí). Z praxe však víme, že každá kombinace dimenzí zdaleka nemusí být v datech zastoupena a neexistuje k ní n-tice faktů, tedy **kostka je řídce obsazena**. Stanovení řídkosti by mělo být součástí dimenzionálního modelování.

Aby nebylo plýtváno kapacitou paměti, jejíž potřeba by byla pro větší počet dimenzí a jejich rozsáhlé domény velmi vysoká, chápe se tato hyperkostka někdy jen jako logický model. Její realizace pak může být upravena tak, že se implementují jen její „obsazené“ části. Definujme si tedy tyto části.

Definice:

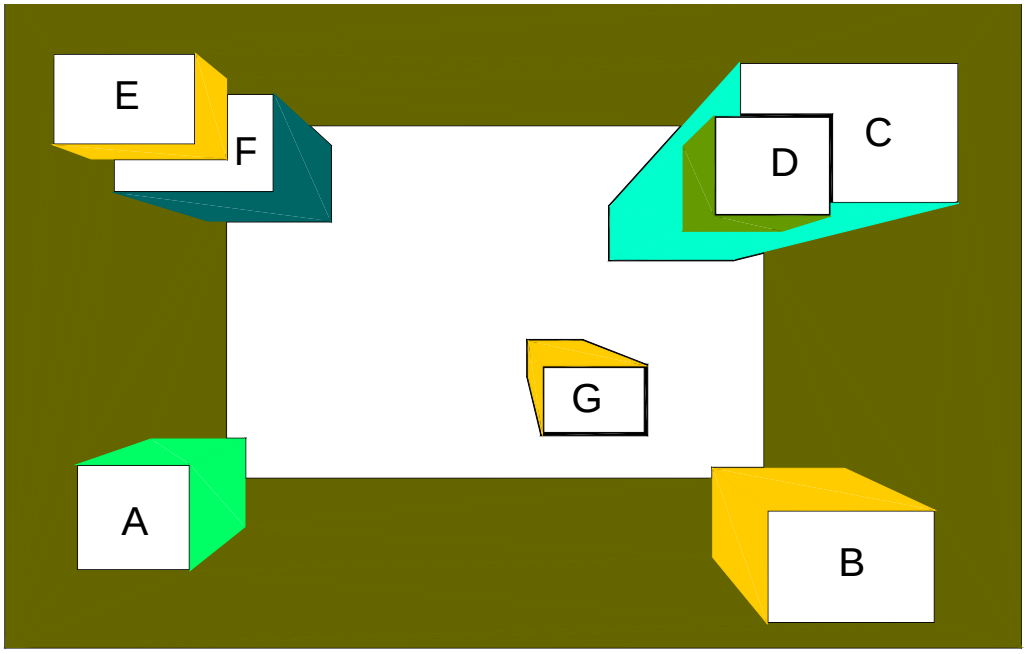
Multikostka je podprostor hyperkostky, v němž nejsou neexistující kombinace dimenzí.

Hyperkostku je tedy možno chápat jako sjednocení multikostek, jakýsi jejich „obal“ .

Hyperkostka je logicky i implementačně jednodušší, multikostky implementačně efektivnější. Proto se používá kombinace obou modelů – na logické úrovni mluvíme o hyperkostkách, ale na fyzické úrovni jsou implementovány multikostky. O to se postará již implementace datového skladu.

Někdy se rozdělují multikostky na

- **blokové** multikostky – fakty i dimenze včetně času tvoří rozměr kostky,
- **sériové** multikostky – pro každý fakt existuje samostatná kostka se všemi dimenzemi, fakty tak tvoří dimenzionální řady (pro dimenzi časovou časové řady).



Hyperkostka a její multikostky

Kterou variantu databázového modelu použijeme, záleží na implementaci datového skladu. Některé mají jen jednu variantu, jiné obě a uživatel si ji může při vytváření DS volit.

9.8. Shrnutí modelování datového skladu

Konceptuální modelování

1. **obsah DS** – entity, atributy, ERD pro základní typy entit a typy vztahů
2. rozhodnutí o **centrálním DS, tržištích DM, komunikaci** mezi nimi
3. **přenos** ze zdrojů, **filtrace, transformace, odvození**

Dimenzionální modelování

4. rozdělení atributů na **dimenze, fakta, atributy**
5. definování typů vztahů, tvořících základ tabulek faktů - ERD
6. definování hierarchie dimenzí
7. určení **aditivity faktů**, definice omezení dotazů

Databázové modelování

8. **transformace ERD** do (buď – nebo)
   - ROLAPu (hvězda, souhvězdí, sněhové vločky, ...)
   - MOLAPu (hyperkostky nebo multikostek)
   - HOLAP (hybrid MOLAPu a ROLAPu)
9. řešení **hierarchií dimenzí,** volba modelu pro hierarchie

   pro ROLAP: transformace ERD do
   - hvězdy či souhvězdí tabulek faktů s hierarchií dimenze v 1 tabulce (F + Di) + generovaný klíč nebo samoidentifikovatelný klíč
   - souhvězdí tabulek faktů dle dimenzí, dimenzionální tabulka obsahuje hierarchii ({F1 + Fd1h1 + Fd1h2 + ...} + Di)
   - sněžení = sněhových vloček, navíc proti souhvězdí tabulek faktů s rozdělením tabulky dimenze podle hierarchie dimenzí ({F + Di}, {Fd1h1 + D1h1, Di}, {Fd1h2 + D1h2, Di}, ...)
   - souhvězdí s explicitními hierararchiemi dimenzí a vazbou mezi stupni dimenze ({F + Di}, {Fd1h1 + D1h1, Di}, {Fd1h2 + D1h2, Di}, ...)

   pro MOLAP multikostky - hyperkostka

Fyzické modelování

stručně probereme v kapitole Implementace DS.

Shrnutí pojmů 9.

Fáze modelování struktury DS. Konceptuální, multidimenzionální a databázové modelování DS. Zadání DS. Výběr dat pro DS.

Jednotný DS a datové tržnice. Datová tržiště a jejich možné vztahy k datovému skladu.

Definování datové pumpy a jejích funkcí pro integraci, filtrace a transformace dat.

Multidimenzionální modelování. Dimenze, fakty, atributy.

Možnosti zpracování agregovaných dat – průběžný výpočet nebo předpočítání.

Konceptuální modelování tabulek dimenzí a faktů. Dva přístupy k databázovému modelu DS. Relační OLAP – ROLAP. Multidimenzionální OLAP - MOLAP.

Hvězdicové schéma. Tabulky faktů a tabulky dimenzí. Kardinalita tabulky faktu F a tabulky dimenze Di. Schéma souhvězdí. Aditivita atributů. Fakty aditivní, semiaditivní, neaditivní. Omezení na dotazy agregovaných hodnot faktů. Hierarchie dimenzí a uživatelský pohyb po údajích. Roll up, drill down. Technologie realizace dimenzionálních hierarchií v ROLAP. Hierarchie dimenzí v jedné tabulce faktů. Generovaný a samoidentifikující klíč. Hierarchie dimenzí pomocí souhvězdí F-tabulek. Hierarchie dimenzí pomocí sněhových vloček. Hierarchie dimenzí - souhvězdí s explicitními hierarchiemi.

Hyperkostka, její dimenze a obsah. Multikostky.

Otázky 9.

1. Popište průběh modelování datového skladu od zadání úlohy po rozhodnutí o prostředí pro realizaci.
2. Popište jednotlivé úkoly modelování konceptuálního.
3. Popište jednotlivé úkoly modelování multidimenzionálního.
4. Popište jednotlivé úkoly modelování databázového.
5. Jaký je rozdíl mezi třístupňovou architekturou databáze a modelováním datového skladu?
6. Co je úkolem konceptuálního modelování DS?
7. Co je datová tržnice a jaké jsou možnosti vztahu DS a DM?
8. Které atributy nazýváme dimenzemi, které fakty a které znovu atributy?
9. Které atributy agregujeme a podle jakých pravidel?
10. Kdy se počítají agregované hodnoty?
11. Jak se modeluje na úrovni ERD datový sklad vzhledem k rozdělení atributů na dimenze a fakty?
12. Pomocí jakých typů SŘBD a jejich technologií se realizují datové sklady?
13. Definujte a vysvětlete pojem hvězdicové schéma DS.
14. Co nazýváme tabulkami dimenzí a co tabulkami faktů?
15. Jaká je kardinalita mezi tabulkami dimenzí a faktů?
16. Co nazýváme multidimenzionální databází?
17. Definujte a vysvětlete pojem souhvězdí DS.
18. Co je aditivita faktů a jaké typy aditivity rozeznáváme?
19. Co jsou omezení na dotazy a proč je definujeme?
20. Co jsou hierarchie dimenzí, jak se poznají?
21. Proč se v DS ukládají agregované údaje na všech úrovních hierarchie dimenzi?
22. Kterými technologiemi se řeší ukládání agregovaných údajů na více úrovních agregace?
23. Popište uložení faktů pro hierarchii dimenzí v jedné tabulce s generovaným klíčem.
24. Popište uložení faktů pro hierarchii dimenzí v jedné tabulce se samoidentifikujícím klíčem.
25. Popište uložení faktů pro hierarchii dimenzí v hierarchii tabulek faktů.
26. Popište uložení faktů pro hierarchii dimenzí v hierarchii tabulek faktů pomocí sněhových vloček.
27. Popište uložení faktů pro hierarchii dimenzí v hierarchii tabulek faktů i explicitně zadanou hierarchií tabulek dimenzí.

28. Co je hyperkostka a jak souvisí s realizací datového skladu?
29. Proč je hyperkostka řídce obsazena?
30. Co jsou multikostky a jaký mají vztah k hyperkostce?
31. Kde se ukládají agregované údaje různých úrovní v hyperkostce?

Úlohy k řešení 9.

1. Vysoká škola potřebuje dlouhodobě sledovat a vyhodnocovat průběžně v čase po dnech, týdnech, měsících a rocích spotřebu energií a služeb – elektřiny, plynu, telefonů apod. Sledování je nutné rozdělit podle jednotlivých kateder a fakult. Navrhněte úplné konceptuální schéma příslušného DS = seznam faktů, dimenzí a ostatních atributů pro DS a určete aditivitu jednotlivých faktů. Navrhněte realizaci v relačním datovém modelu pomocí sněžení.

2. Totéž navrhněte pomocí ostatních technologií pro realizaci hierarchie dimenzí.

10. METADATA

Čas ke studiu: 1 hodina

Cíl Po prostudování této kapitoly budete vědět

- co jsou metadata obecně
- jaké typy metadat rozlišujeme v datových skladech
- co jednotlivé typy metadat zahrnují a k čemu se používají

Výklad

Metadata

Z mnoha důvodů jsou zapotřebí v databázi i v datovém skladu mimo vlastní data i údaje o jeho struktuře, obsahu, funkcích a další informace. Tato „data o datech“ nazýváme **metadata**. Slouží vývojářům, administrátorovi, uživateli i vlastnímu programovému systému DSS.

Aniž jsme používali tento pojem, známe metadata již z klasického informačního systému. Je to například datový slovník databáze, který vytváříme v datovém modelu. Vlastní metadata o relačním schématu (= struktuře tabulky) obsahují: identifikátor atributu, jeho datový typ, délku, je-li klíč, může-li být NULL, je-li cizí klíč a odkud, je-li indexován a jakým typem indexu, titulek jak se bude zobrazovat uživateli, popis s případnými dalšími integritními omezeními, definujícími jeho doménu.

Také víme, že SŘBD si vede vlastní evidenci o databázi a jejích částech v tzv. systémovém katalogu. Ten obsahuje také metadata o IS. Mimo již zmíněný seznam atributů seznam tabulek a jejich vlastností, seznam indexů, definovaných reportů a formulářů atd.

Metadata datového skladu

Přirozeně i v DS budou nutná metadata, dokonce jejich ještě více. V operativních databázích jsou metadata koncovým uživatelům v podstatě skryta, pracují s nimi jen vývojáři a správci databáze. Uživatelé pracují s IS prostřednictvím uživatelského rozhraní jen jako s černou skříňkou.

O obsahu datového skladu = o jeho datových strukturách však musí být uživatelé - analytici předem informováni, aby mohli DS správně a efektivně využívat. Proto musí pracovat i s jeho metadaty, rychle pomocí nich vyhledat požadovaná data i jejich interpretaci.

Podle obsahu můžeme metadata pro DS rozdělit na několik druhů:

Metadata pro správu DS

jsou informace, které slouží analytikům, návrhářům při vývoji DS a správci DS při provozu. Jsou to:

- **metadata zdrojových dat** pro potřebu analýzy a návrhu DS
  - rozmístění databází na serverech,
  - struktury zdrojových databází,
  - struktury a popisy entit a jejich vazeb,
  - definice a popis atributů, jejich datových typů, domén včetně měrných jednotek, klíčů, indexů,

  - informace o vlastnictví dat a případných vazbách mezi zdrojovými daty (kdo komu poskytuje data);

- **metadata datového skladu** - katalog DS obsahuje
  - seznam serverů,
  - rozmístění databází DS na serverech,
  - definice tabulek a pohledů,
  - definice a popis atributů, primárních a cizích klíčů, indexů,
  - rozdělení atributů na dimenze a jejich hierarchie, fakty a popisné atributy, omezení na dotazy,
  - definice tabulek dimenzí a tabulek faktů;
- **metadata pro datovou pumpu** – mapování zdrojových dat z operativních databází do cílových atomických dat v primárním datovém skladu
  - pro každý atribut DS pravidla pro jeho kopírování, integrační funkce, transformační pravidla, změny formátů, verifikace, odvozování, omezení na dotazy;
  - měrné jednotky a konverzní koeficienty používané mezi jednotkami, zvláště když jsou to vzorce nebo koeficienty proměnné v čase;
  - informace o časování převodů dat z DB do DS;
  - obchodní pravidla a postupy, vztahy pro výpočet ekonomických ukazatelů, používané vzorce a postupy výpočtu;
- **metadata pro data a funkce na pozadí DS**
  - podpůrné dočasné datové struktury pro transformace, pro zobrazení uživatelům,
  - funkce extrakční a transformační, funkce pro zabezpečení kvality; pořadí jejich spouštění, parametry programů,
  - popis strategie plnění DS, definice dočasných podpůrných tabulek a jejich funkce;
- **architektura DS** – v případě, že existují datové tržnice, pak
  - struktura DS a datových tržnic DM,
  - definice podmnožin DM z DS,
  - pořadí plnění DS a DM;
- **přístupová práva a zabezpečení DS**
  - informace o uživatelských rolích a jejich právech,
  - informace o jednotlivých uživatelích a jejich rolích.

Metadata pro koncové uživatele

slouží uživatelům, ať specializovaným analytikům nebo koncovým a náhodným uživatelům. Patří k nim metadata pro vytváření dotazů a pro správnou interpretaci výsledků:

- **obsah datového skladu** – datové struktury v uživatelsky přívětivé formě s možností výběru; dimenze a jejich hierarchie, fakty a jejich agregované hodnoty,
- **kvalita dat** – u všech údajů musí existovat informace o jejich kvalitě, jsou-li a která data jsou spolehlivá, upozornění na data chybná, chybějící,
- **předdefinované dotazy a sestavy** – používaných dotazů, katalog výstupních sestav a grafů, význam jednotlivých prvků v sestavách a ve výsledcích analýz, význam popisů, metod a analýz pro uživatele,
- **obchodní pravidla a postupy** – protože je možno používat různé vztahy pro výpočet ekonomických ukazatelů, musí mít uživatelé informaci o použitých vzorcích a postupech výpočtu *(například pro výpočet nákladů nebo zisku apod.)*,

- **stavové informace** – v denním provozu při denním doplňování skladu může být skla v různém stádiu vývoje a uživatel musí být informován o tom, obsahuje-li sklad jen data stará, je-li právě plněn a nová data ještě nejsou dostupná nebo už obsahuje i data aktuální,
- **pravidla pročišťování skladu** – pravidla, kdy je možné data ze skladu odstranit či archivovat, aby uživatelé věděli, do kdy budou která data ve skladu dostupná,
- **historie plnění skladu** – historie všech plnění skladu od počátečních jednorázových zdrojů až po současné periodické doplňování daty; pro všechny přenosy by měly být evidovány přenášené objemy dat, protokoly o zjištěných chybách v datech, doby nutné pro přenosy a výpočty agregací; záznam o historii má být synchronizován se stavovými informacemi; uživatelům dostupný by měl být také časový plán plnění, aby byli informováni o tom, kdy budou k dispozici čerstvá data.

Metadata optimalizační

mohou mimo jiné sloužit k optimalizaci návrhu a výkonu datového skladu. Tato metadata zahrnují

- **definice agregací a jejich umístění** – popis navigace mezi D-tabulkami a F-tabulkami rozsáhlého skladu v ROLAPu pro urychlení přístupu k požadovaným datům;
- **omezení na dotazy** – pro rychlejší plnění DS, menší kapacitu, rychlejší přístup k datům,
- **statistiky datového skladu** – sledování četnosti různých typů dotazů nad datovým skladem; tato informace je zpětnou vazbou pro správce skladu a ten může na jejich základě identifikovat jak data často sledovaná, tak data dlouhodobě nepoužívaná; výsledkem může být úprava obsahu DS.

Metadata jako základ pro automatizaci podpůrných procesů

mohou sloužit jako podklad pro pozdější automatizaci řady funkcí, dosud realizovaných na míru DS. Evidují se průběžně protokolují

- **metadata pro extrakce a transformace** – přiřazování zdrojových dat cílovým může sloužit jako podklad pro generování skriptů extrakce, integrace a transformace,
- **kvalita dat** – uživatelé mohou zadávat přípustné hodnoty pro různé atributy, to slouží k odhalení chyb s případnou následnou automatickou opravou,
- **generování dotazů** – zaznamenaná struktura dat slouží ke generování uživatelských SQL dotazů,
- **koncové nástroje** – generování nástrojů pro zobrazení struktury tabulek nebo obsahu sumačních tabulek

❑ Standardizace metadat

Sdílení a výměna metadat je jedním z důležitých problémů při návrhu datových skladů, proto existuje snaha o **standardizaci** metadat.
Tyto snahy je možno pozorovat ve třech úrovních:

- vytvářejí se obecně použitelná skladiště metadat, tzv. **repozitáře** = specializované databáze pro uchování dat o systému (*příkladem jsou Platinum Repository, Microsoft Repository, Unisys UREP, ...);* ty pak mohou být použity u kteréhokoliv DS a tedy přístup k nim by byl jednotný ze všech takových datových skladů;
- definují se **standardy pro výměny dat** od dodavatelských sdružení (*příkladem jsou CWMI - Common Warehouse Metadata Interchange od OMG - Object Management Group – IBM, Oracle, Unisys, ... a MDIS - Meta Data Interchange Specification od MDC - Meta Data Coalition – Microsoft*); i když má každý spolupracující DS vlastní metadata, stačí vytvořit konverzní

programy do a z standardizovaných metadat a opět je možno komunikovat s ostatními datovými sklady;

- **otevřené API rozhraní k produktům** – většina dodavatelů řešení datových skladů poskytuje otevřené rozhraní pro přístup produktů třetích stran a aplikací k jejich metadatům (*příkladem jsou Hyperion Integration Server, IBM Meta Data Interchange Language*).

Shrnutí pojmů 10.

Metadata a jejich dělení. Metadata datového skladu. Metadata pro analýzu DS.
Metadata pro datovou pumpu.
Metadata podporující aplikace v DSS.

Otázky 10.

1. Co jsou a k čemu jsou metadata obecně a co v datovém sklad?
2. Které druhy metadat rozlišujeme v datovém skladu?
3. Co všechno patří do metadat jednotlivých druhů?

11. TECHNOLOGIE PRO IMPLEMENTACI DS

Čas ke studiu: 2 hodiny

Cíl Po prostudování této kapitoly budete umět

- vyjmenovat nástroje, které používají SŘBD při implementaci datových skladů
- popsat indexovací technologie využívané v DS a jejich rozdíl proti použití v klasických IS
- popsat další nástroje využívané při implementaci DS

Výklad

Typy nástrojů

Vysoké objemy dat v DS a komplikovanější charakter jejich využívání oproti operativním databázím vedou i k jiným fyzickým modelům databáze i k jiným technologiím přístupu k datům.

Používají se buď technologie specifické pro datové sklady, nebo se aplikují technologie známé i jinde.

Stěžejním požadavkem je maximální zrychlení přístupu k datům. Toho se v současnosti dosahuje několika způsoby a jejich kombinacemi:

- inkrementálním plněním skladu i výpočtem agregací
- předpočítáním a uložením předpokládaných agregací
- rozdělením datového skladu na menší datová tržiště
- použitím speciálních indexovacích technologií
- využitím paralelního přístupu k datům

Předpočítání a ukládání agregovaných údajů jsme probrali v minulé kapitole, také datové tržnice jsme diskutovali. Jde o součást databázového modelování. Ostatní metody probereme v následujících odstavcích.

Inkrementální plnění datového skladu

Uváděli jsme si, že datový sklad se na počátku naplní jednorázově daty z archivů a jiných starších zdrojů, případně z aktuálních operativních databází. Potom se pomocí datové pumpy doplňuje periodicky daty novými,vzniklými v operativních databázích od posledního plnění. Perioda může být různě dlouhá, od denního cyklu *(například u obchodních dat)* až po velmi dlouhý interval *(například roční u dat určených k dlouhodobějšímu výzkumu v lékařství či jinde).*

Zvláště u dat denních je třeba dobu plnění – práci datové pumpy - optimalizovat.

Výpočet agregovaných hodnot zabírá nejdelší čas. Jak se celková velikost dat stále rozšiřuje, jsou to stále větší objemy pro sumování. Velmi mnoho času se může ušetřit na vhodném inkrementálním způsobu výpočtu agregačních funkcí:

suma nová = suma předcházející + suma přírůstku

minimum nové = min (minimum předcházející, minimum přírůstku)

obdobně se spočítá počet a maximum. Není nutné znovu sumovat všechno. Jen nové průměry je nutné spočítat podílem nových sum a počtů. Počty, sumy, extrémy přírůstků se spočítají průběžně během konverze dat, předcházející hodnoty jsou uloženy v DS.

Indexovací technologie

Indexů se v DS používá několika typů. Některé z nich známe od IS, jiné jsou nové.

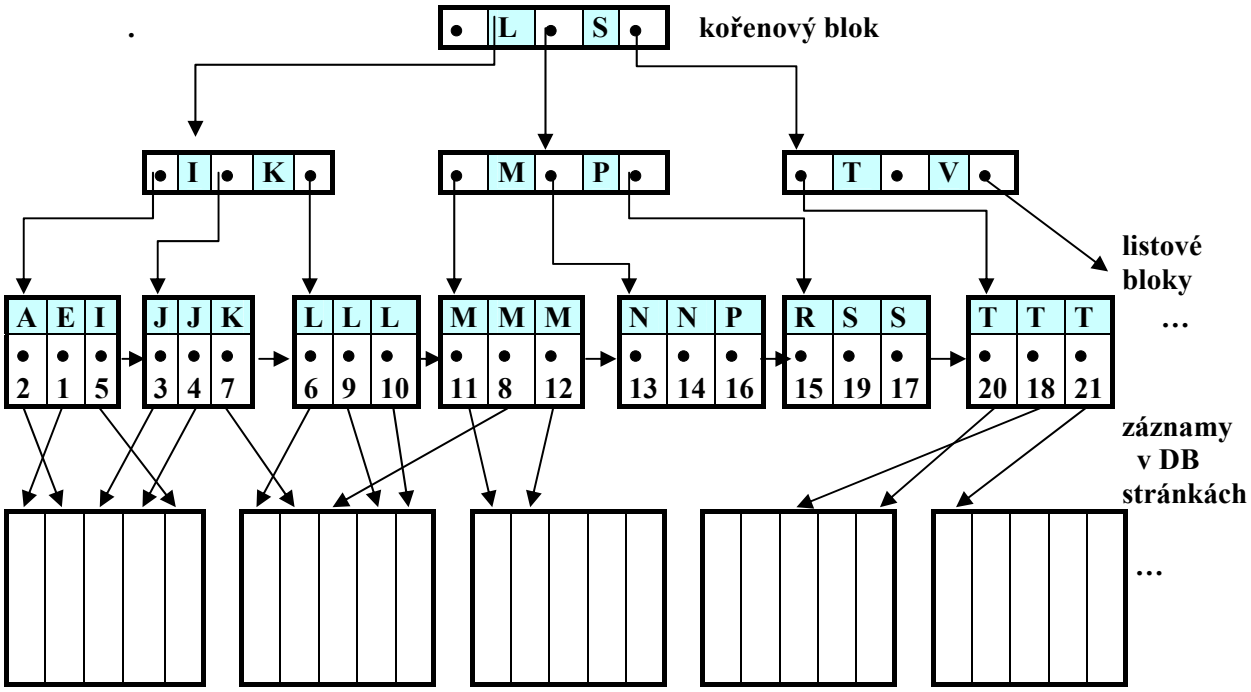
- B+ stromy (mají obecné použití v relačních SŘBD)
- Binární indexové matice (používají se v DS a OLAP)
- Join indexy (řešení je součástí multidimenzionální analýzy)
- R stromy a rastrové indexy

B+ stromy (Balanced tree)

Indexuje se jeden nebo více sloupců pomocí víceúrovňové údajové struktury, obsahující kořenový uzel s ukazateli na uzly v další úrovni.

Nejnižší úroveň obsahuje bloky listů na každý řádek indexované tabulky

Pokud jsou **listy provázány ukazateli**, umožňují sekvenční procházení tabulkou, dotazování na intervaly, třídění dle indexového klíče bez procházení celým B stromem - pak je nazýváme B+ stromy.



B+ strom

Dalšími možnostmi jsou

- ořezávání indexových hodnot v nelistových blocích u dlouhých řetězcových sloupců,
- uchování u sekundárních indexů nejen hodnot indexovaného sloupce, ale i primárního klíče pro dotazy zahrnující oba atributy (ovšem index je efektivní, pokud jsou jeho záznamy mnohem kratší proti záznamům datové tabulky!),
- kombinování více indexů na základě složených selektivních podmínek pomocí dotazů AND/OR, vytváření několika dočasných indexů jednoduchých, ty logicky spojit podle selektivní podmínky a teprve závěrem přistoupit k datovým řádkům,
- SQL optimalizátory zpracují u hvězdicového schématu nejprve omezující podmínky nad tabulkami dimenzí a až nakonec je připojí k tabulce faktů.

Tyto technologie používá většina současných výrobců SŘBD – IBM, Oracle, Informix, Sybase, atd.

❑ Binární indexové matice

Připomeňme si binární matice z předmětu Teorie zpracování dat.

Pro sekundární atributy s malou doménou (malým počtem možných hodnot) se někdy z důvodů ušetření kapacity používá indexování pomocí binárních matic. V indexovém souboru není jediný v záhlaví indexovaný atribut, ale všechny jeho možné hodnoty. U každého záznamu se zaznamenává hodnota atributu polohou jedničkového bitu v posloupnosti, která má tolik bitů, kolik je hodnot.

Příklad:

Mějme soubor zaměstnanců s osobním číslem, platem (4 hodnoty) a procentem daně (3 hodnoty). Pak příslušné indexy vypadají takto:

Zam

adr	**osob**	**...**	**plat**	**dan**
1			2000	30
2			4000	10
3			2000	20
4			4000	30
5			5000	10
6			2000	20
7			5000	10
8			4000	30
9			3000	20
10			2000	10

I_plat

2000	3000	4000	5000
1	0	0	0
0	0	**1**	0
1	0	0	0
0	0	**1**	0
0	0	0	1
1	0	0	0
0	0	0	1
0	0	**1**	0
0	1	0	0
1	0	0	0

I_dan

10	20	30	
0	0	**1**	
1	0	0	
0	1	0	
0	0	**1**	**1**
1	0	0	
0	1	0	
1	0	0	
0	0	**1**	**1**
0	1	0	
1	0	0	

Hledáme-li zaměstnance s platem 4000 a daní 30, projdou se příslušné sloupce v jednom a druhém indexu a vyhledají jedničky. K nim se určí pořadí záznamu v datové tabulce.

Ještě výhodnější je ukládat index transponovaně – záznamy ve sloupcích a binární vektory pro stejnou hodnotu atributu v řádcích. Pak se pracuje snadno s celými binárními vektory.

atribut	hodnota	pořadí záznamů 1 2 3 4 5 6 7 8 9 10 11 12 …
dan	**10**	0 1 0 1 0 1 0 0 0 0 0 1
	20	1 0 0 0 0 0 1 1 0 0 0 0
	30	0 0 1 0 1 0 0 0 1 1 1 0
plat	**2000**	0 0 0 0 1 0 0 0 0 0 0 0
	3000	1 0 0 0 0 1 0 0 1 1 0 0
	4000	0 1 1 1 0 0 0 0 0 0 1 1
	5000	0 0 0 0 1 0 1 1 0 0 0 0

dan = 30 ∧ plat = 4000	**1**		**1**

♦

Binární matice jsou zvlášť výhodné v případech, že se hodnoty sekundárních atributů nemění, když se nemění záznamy, případně se jen přidávají sériově na konec souboru. Další výhodou binárních matic je snadná realizace kombinovaných dotazů pomocí logických operátorů negace, konjunkce a disjunkce.

Pomocná indexová struktura tedy používá bit jako příznak příslušnosti hodnoty atributu k dané entitě. Počet sloupců bitmapy odpovídá kardinalitě indexovaného sloupce. Použití bitmapových indexů není výhodné při operacích INSERT a UPGRADE. Protože se však v DS pouze čte, je zde jeho využití výhodné.

Tuto technologii používá například Oracle, Sybase atd.

❑ Indexy pro podporu spojení (Join Index)

Pro velmi složité dotazy, zahrnující mnoho operací spojení (join) na velkých databázích nejsou ani používané klasické algoritmy spojení dostatečně rychlé. Přitom v datových skladech se provádí takových spojení velké množství – tabulka faktů se spojuje s řadou tabulek dimenzionálních, aby z nich získala jednak požadované hodnoty dimenzí, jednak uživatelsky pohodlnější popisy atributů, nejen jejich id.

Proto byla speciálně pro podporu operace spojení vytvořena nová údajová struktura – index pro podporu spojení. Jde o pomocnou tabulku skládající se mimo ze dvou nebo více sloupců. Ta obsahuje mimo index adresy příslušných záznamů ve 2 nebo více spojovaných tabulkách podle indexovaného spojení.

Mějme F-tabulku **F** s adresami řádků adr_F a dimenzionální D-tabulku **D** s adresami řádků adr_D, spojení se provádí pomocí atributu id_D. Pak indexový soubor má sloupce id_D, adr_F, adr_D, setříděné pro binární hledání podle id_D. Když se vyhledá hodnota id_D, příslušné adresy na vyhledaném řádku označí čísla řádků současně v obou tabulkách **F** i **D**. Navíc vytvořením B+ stromu nad tímto indexem získáme rychlý přístup při složitějších dotazech.

Příklad:

Join index pro spojení dimenzionální tabulky Oddělení a tabulky faktů Prodej *spojovací podmínkou*

Oddelení.id_odd = Prodej.id_odd

D_Oddelení

adr_D	id_odd	mesto
1	17	OS
2	13	OP
3	7	BR
4	2	FM

F_Prodej

adr_F	id_odd	datum	mnoz
32	2	3.3.02	3200
33	13	1.3.02	2500
34	7	1.3.02	9800
47	17	1.3.02	3200
55	17	2.3.02	6500
65	7	2.3.02	7400

Join index: Oddel [*]Prodej

id_odd	adr_D	adr_F
2	4	32
7	3	34
7	3	65
13	2	33
17	1	47
17	1	55

Přístup k záznamům tabulek Oddělení a Prodej *pro požadované* id_odd=13 *získáme vyhledáním hodnoty 13 v indexu (červeně vyznačené řádky), tam zjistíme adresy příslušných záznamů v obou datových tabulkách. Nebo když potřebujeme setříděný výpis celé tabulky* Prodej *i s výpisem města z Oddeleni, procházíme sekvenčně indexový soubor a podle adres čteme rovnou odpovídající si záznamy datových tabulek.*

♦

❑ Kombinovaný index

Spojením principů bitmapových indexů a indexů pro spojení dostaneme kombinovaný index.

Je obdobou předcházejícího indexu, ale tabulka faktů je spojena s dimenzionální tabulkou tak, že k popisnému sekundárnímu atributu je zkonstruován bitmapový index. Tak je možné přistupovat k faktům spojeným s dimenzí a omezit hodnoty atributu pomocí bitových operací.

Příklad:

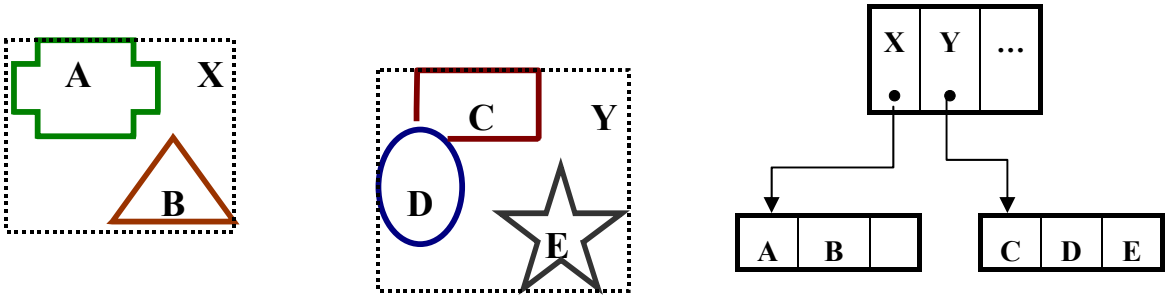
Kombinovaný join index pro spojení dimenzionální tabulky Oddělení *a tabulky faktů* Prodej *indexovaný s binárním indexem pro atribut* mesto.

adr_D	adr_F	OS	OP	BR	FM
	...				
4	32	0	0	0	1
2	33	0	1	0	0
3	34	0	0	1	0
	...				
1	47	1	0	0	0
	...				
1	55	1	0	0	0
	...				
3	65	0	0	1	0

♦

❑ R stromy

Pro podporu prostorových dat byla navržena speciální modifikace B stromu, zabezpečující efektivnější přístup k rovinným (2D) nebo prostorovým (3D) objektům v relační nebo objektově-relační databázi. Proti B stromům jsou v listech kromě id_řádků i informace o ohraničení příslušného objektu. V blocích vyšších úrovní se uchovávají informace o ohraničení sjednocení objektů nižší úrovně. Pro případ 2D (rovinné objekty) jde například o souřadnice [x, y] pro pravý dolní a levý horní roh jednoho objektu či sjednocení několika objektů.



Rovinné objekty a jejich R-strom

Používanější variantou R stromů jsou rastrové indexy (též mřížkové indexy). Mapovaný prostor je rozdělen mřížkou, její jednotlivé bloky jsou očíslovány a tedy tříditelné. Vytvořením druhé hustší mřížky se bloky první úrovně opět rozdělí a tak se vytváří varianta B stromu vhodná pro přístup k prostorovým datům.

❑ Paralelizmus

Pro efektivní provoz DS se používají dále paralelní technologie. Pro paralelní přístup se používají dvě metody:

1. rozdělení databáze do paralelně přístupných částí (partitioning)

- *horizontální dělení* - pomocí selekce tabulky se vytvoří disjunktní množiny řádků, ty se umístí do zvláštních fragmentů; hlavním kandidátem na rozdělení jsou vybrané dimenze a čas; negativním faktorem jsou „horká místa“, fragmenty nadměrně využívané proti ostatním fragmentům (například poslední časové období); výhodou je možnost rozdělení příslušných indexů, což zvyšuje rychlost odezvy;
- *vertikální dělení* – tabulka faktů je projekcí rozdělená na víc fragmentů, ty se pomocí operace spojení spojují pomocí redundandního primárního klíče; výhodou je rychlejší přístup k datům pro uživatele, kteří mají zájem jen o přidělená data (kratší záznamy, vyšší počet záznamů na stránce).

2. symetrický multiprocesing / masivně paralelní procesing (SMP/MPP) skutečné paralelní zpracování jedné aplikace na více procesorech současně rozdělením úlohy do vláken; u SMP se používají shodné procesory přistupující ke společné operační paměti, MPP může obsahovat různé procesory s vlastní operační pamětí a jsou tak spolupracujícími počítači.

U DS má smysl paralelně zpracovávat především dva oddělené typy procesů – datovou pumpu a OLAP dotazy, nebo samostatné dotazy různých uživatelů.

Σ Shrnutí pojmů 11.

Specifické nástroje pro implementaci datových skladů.
Indexovací technologie. B+ stromy. Binární indexování. Join indexy pro podporu spojení. Kombinování indexů binárních a indexů pro podporu spojení. R stromy, rastrové indexy. Paralelismus procesů.

Otázky 11.

1. Které nástroje na rozdíl od operativních databází se používají při implementaci DS?
2. Popište princip B+ stromů.
3. Popište použití binárních indexovacích matic a jejich využití.
4. Popište princip indexů pro podporu spojení.
5. Popište možnost kombinování binárních indexů a join indexů.
6. Popište princip R stromů.
7. Jak je možno využívat paralelních procesů v DS?

12. SW A VYUŽITÍ DATOVÝCH SKLADŮ

Čas ke studiu: 1 hodina popis + 2 hodiny cvičení

Cíl Po prostudování této kapitoly se seznámíte

- s použitím MS SQL Serveru pro realizaci datových skladů,
- s realizací datového skladu „na míru“ konkrétním datům.

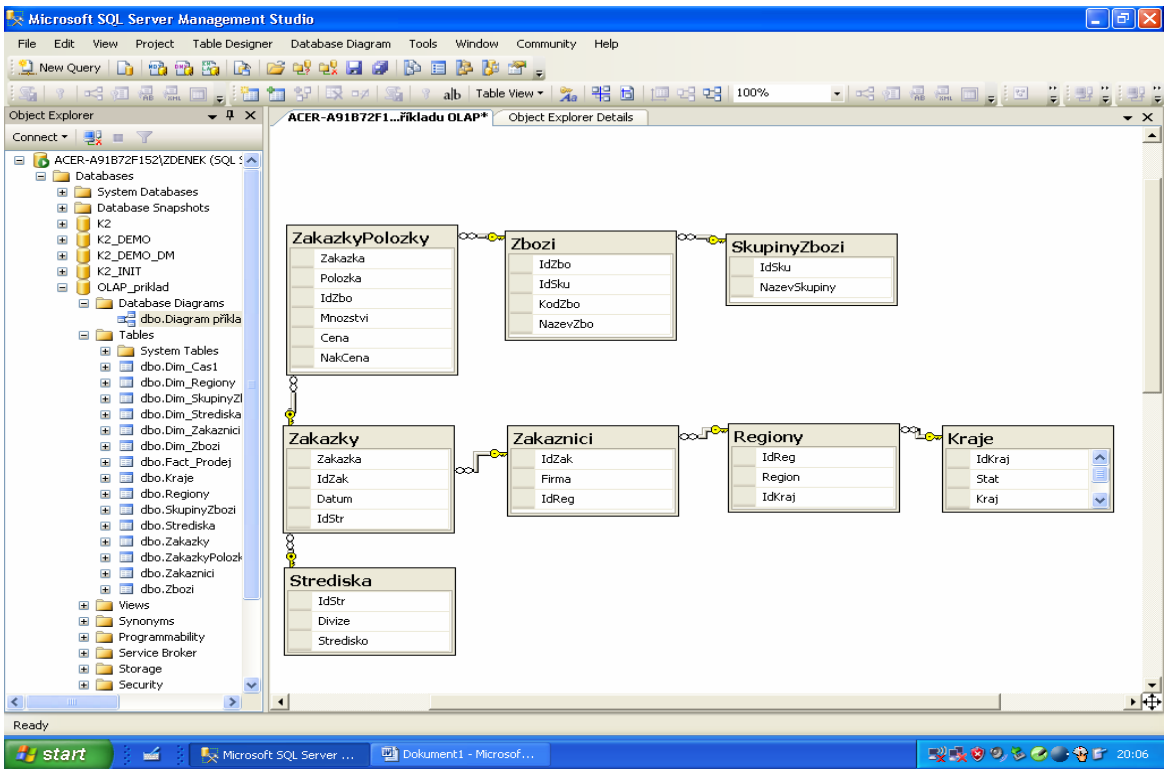
Výklad

12.2. Datový sklad v MS SQL serveru 2005

Řada velkých SŘBD podporuje mimo klasické informační systémy také datové sklady a OLAP. Ukážeme si jejich vytvoření a použití na malém příkladu realizovaném v MS SQL serveru 2005.

Příklad:

Je dána zdrojová databáze z následujícího schématu:

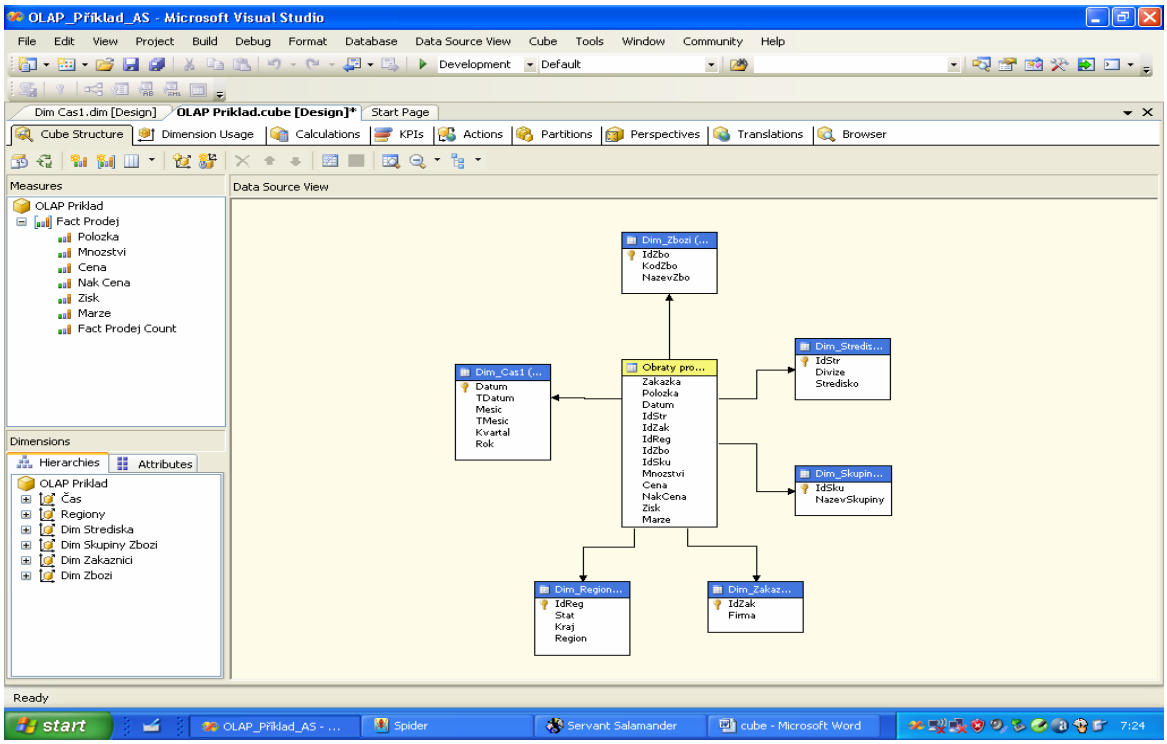


Model datového skladu odvozeného z této databáze má definované

***fakty:** počet položek, sumy množství, ceny, nákladové ceny,zisku, marže,*

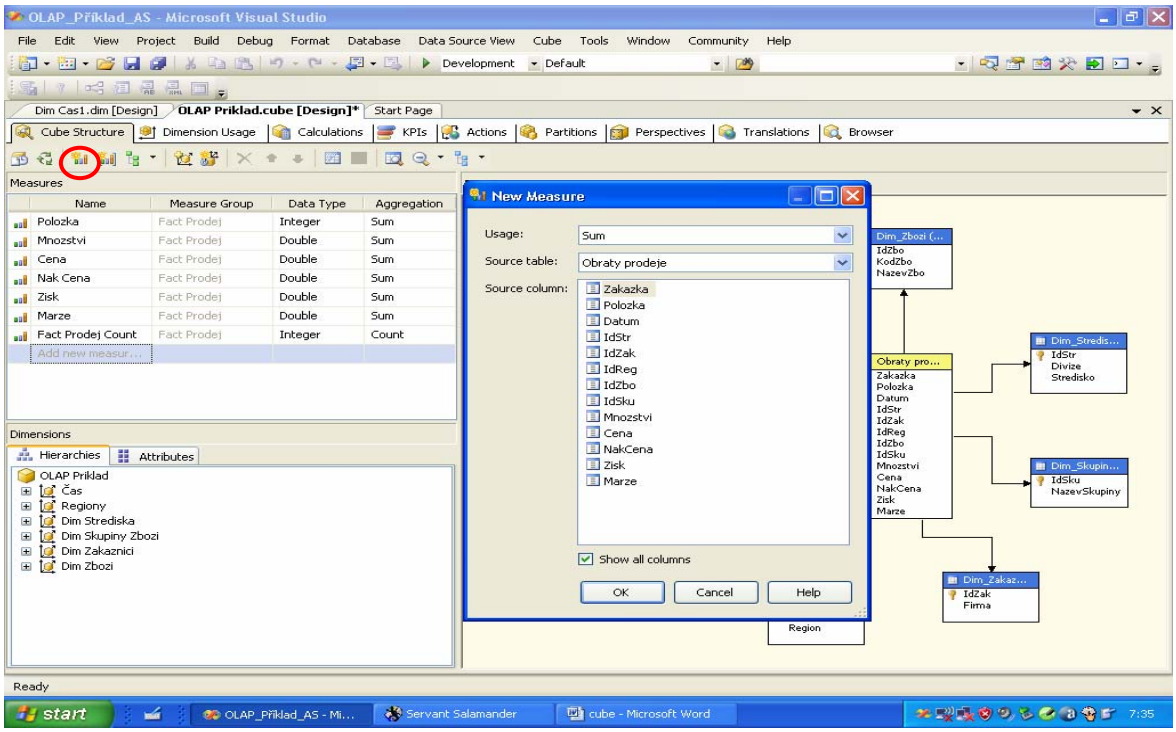
***dimenze**: čas, regiony, zboží, skupiny zboží, střediska, zákazníci.*

a tedy logický model struktury hvězdy je následující:

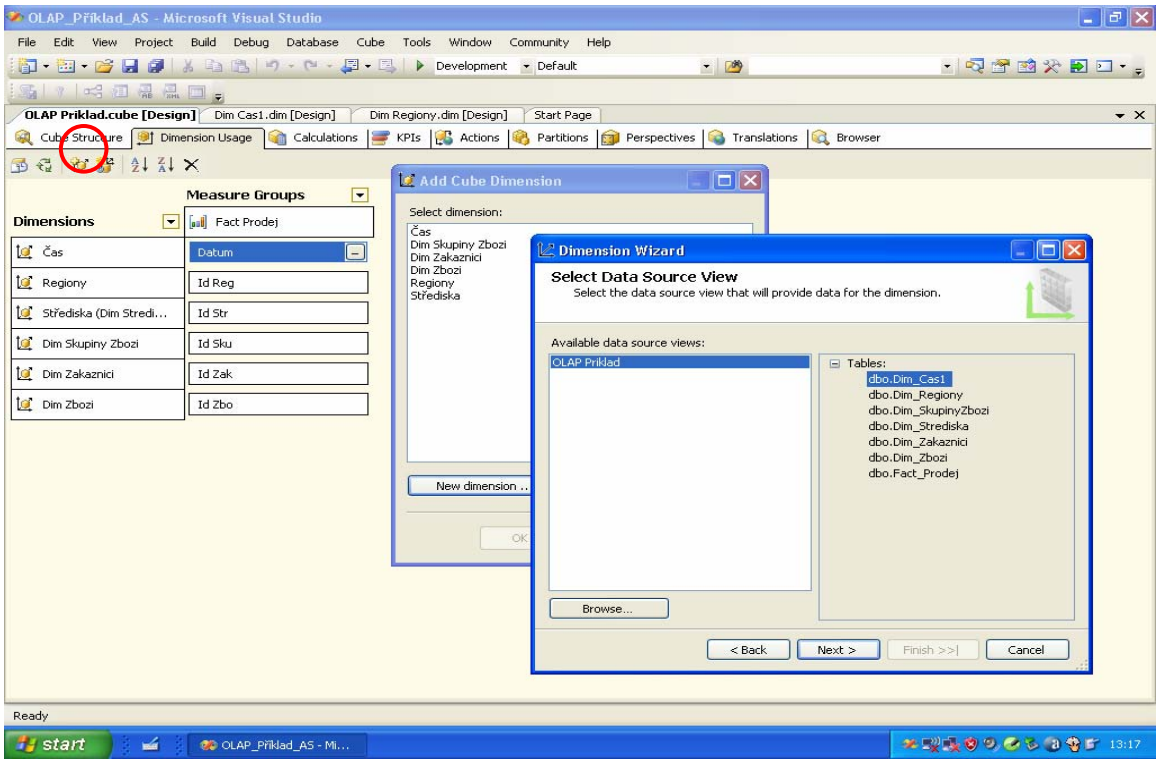


Definice tohoto datového skladu se provádí z klasicky definovaných tabulek pomocí příkazů CREATE TABLE *takto:*

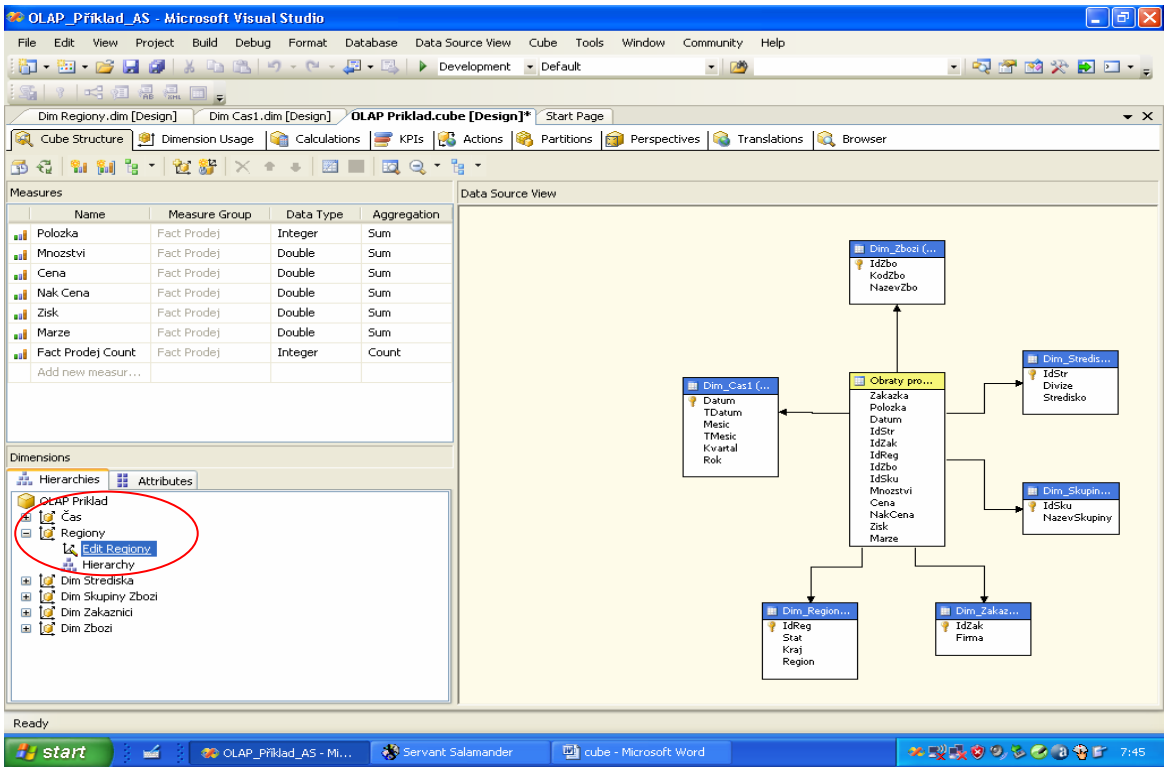
Definice faktů *ikonou* New measures *(červeně označena), výběrem agregační funkce, výběrem zdrojové tabulky a výběrem zdrojového atributu v okně* New measures:



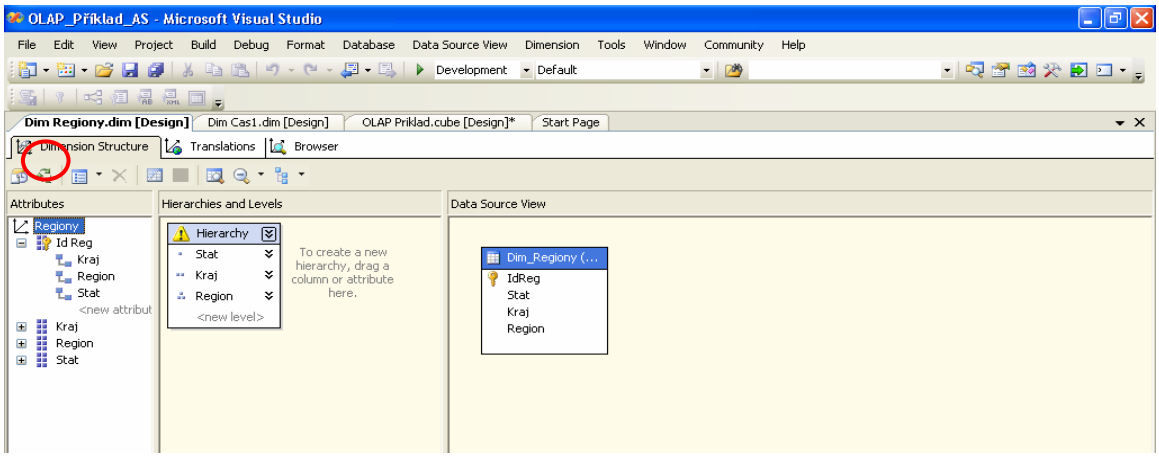
Definice dimenzí *provedeme ikonou* Add Cube Dimension,*dole volbou* New dimension *a výběrem dimensionální tabulky:*



a definice hierarchie dimenze editací zvolené dimenze



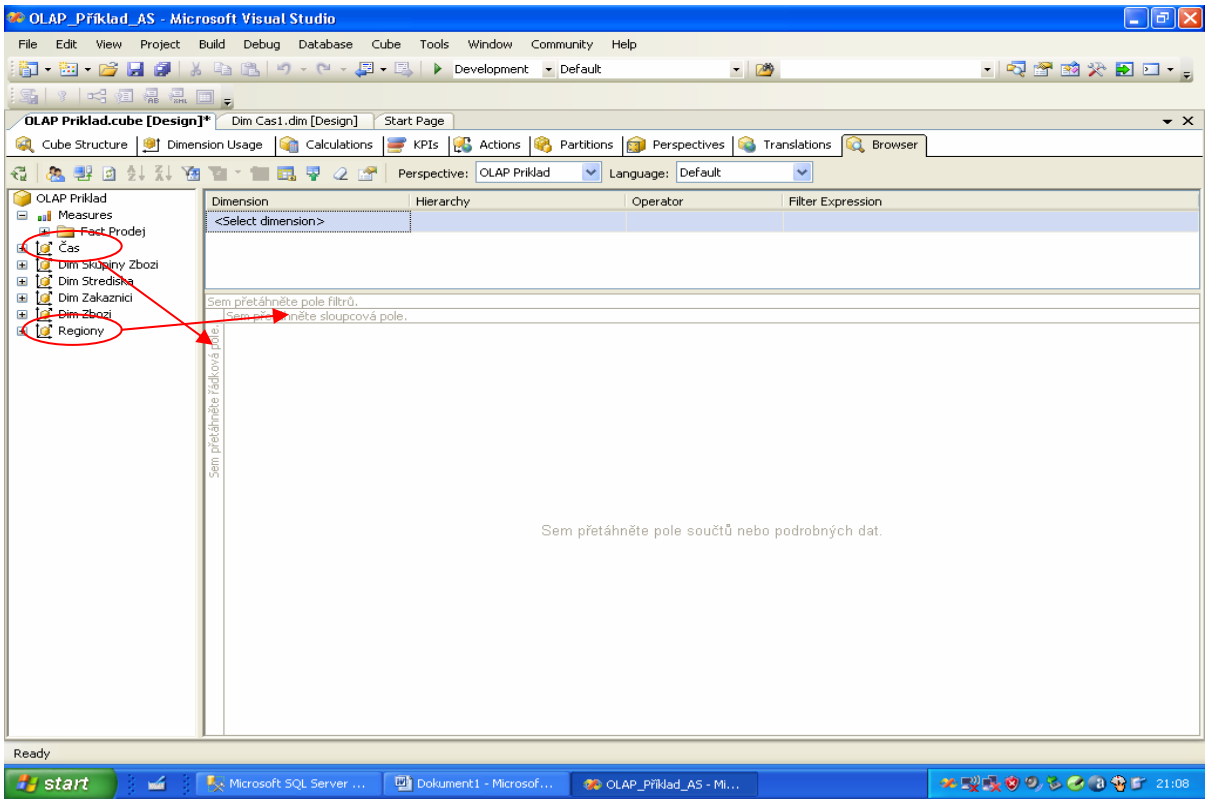
a doplnění hierarchie a případných popisných atributů:



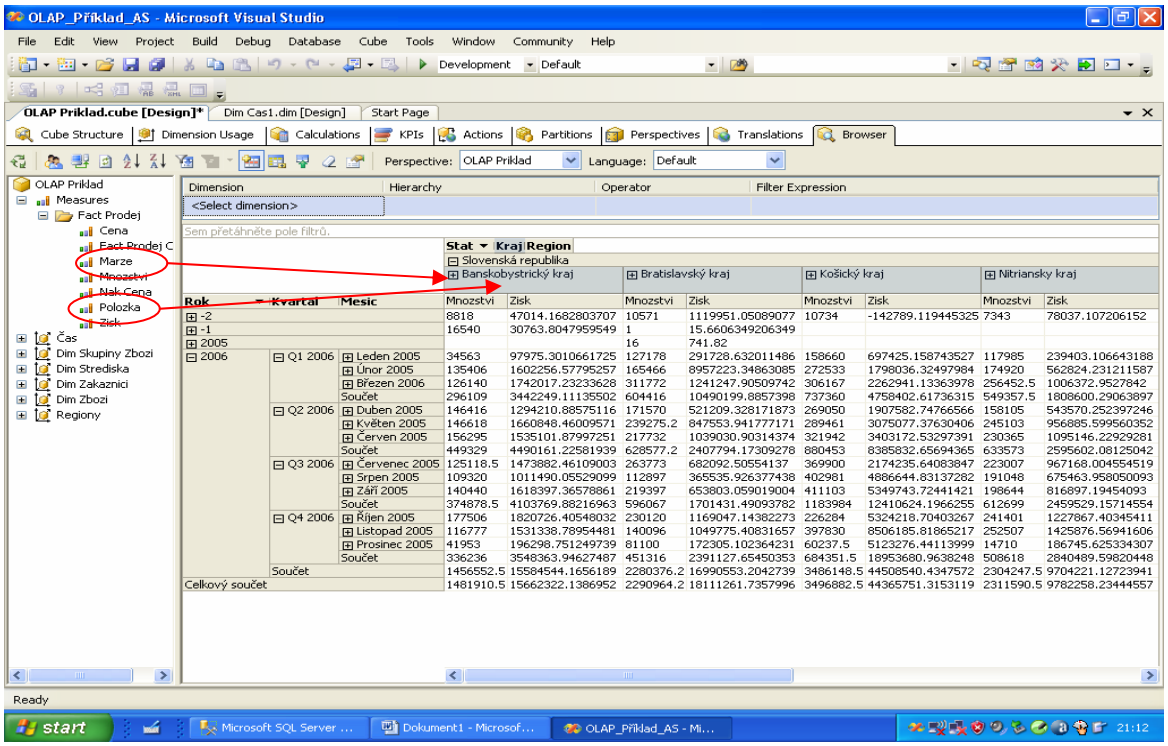
Po definování struktury kostky se pomocí ikony Process *sestaví kostka, tedy spočítají se nastavené agregace.*

Po sestavení a naplnění kostky (datového skladu) je možno prohlížet její obsah v uživatelem libovolně zvolené sestavě dimenzí a faktů.

Zvolíme záložku Browser *a dostaneme prázdnou plochu s nápovědou, kam máme přetáhnout zleva vybrané dimenze – jejich hierarchickou strukturu libovolně rozklikáme:*

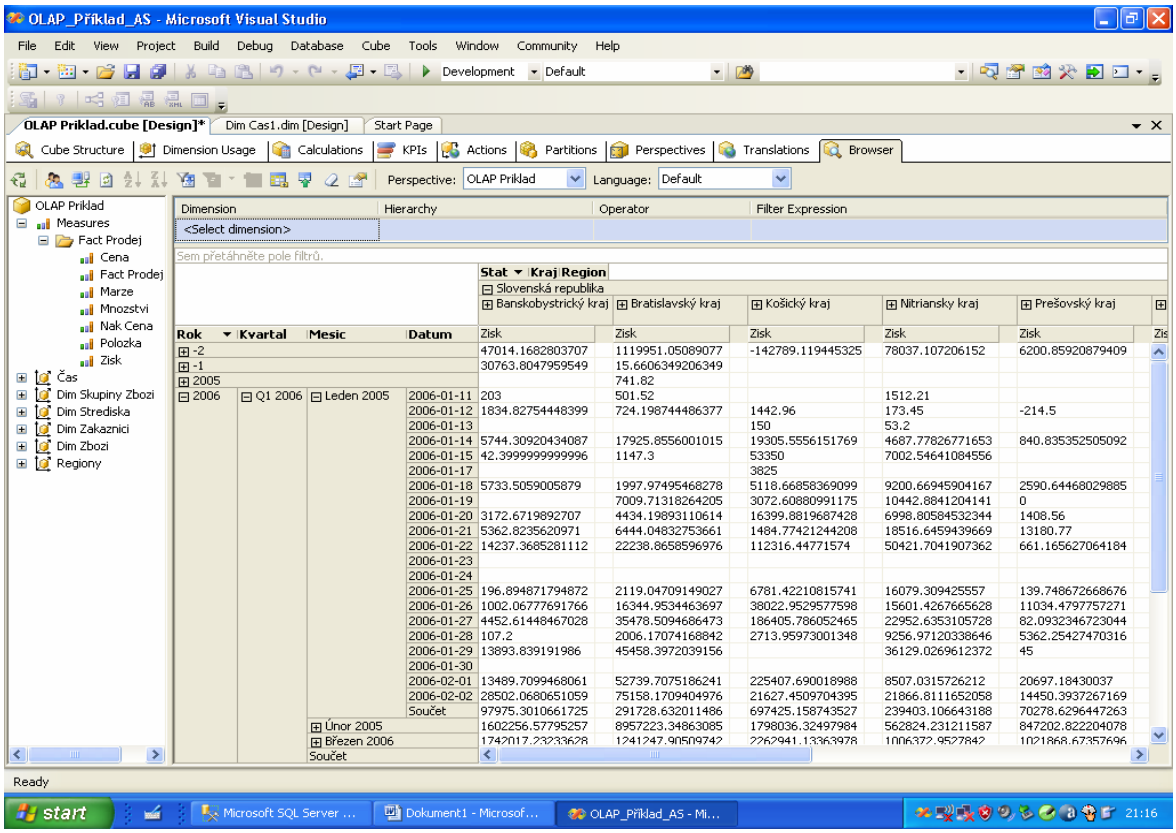


a vybrané fakty:

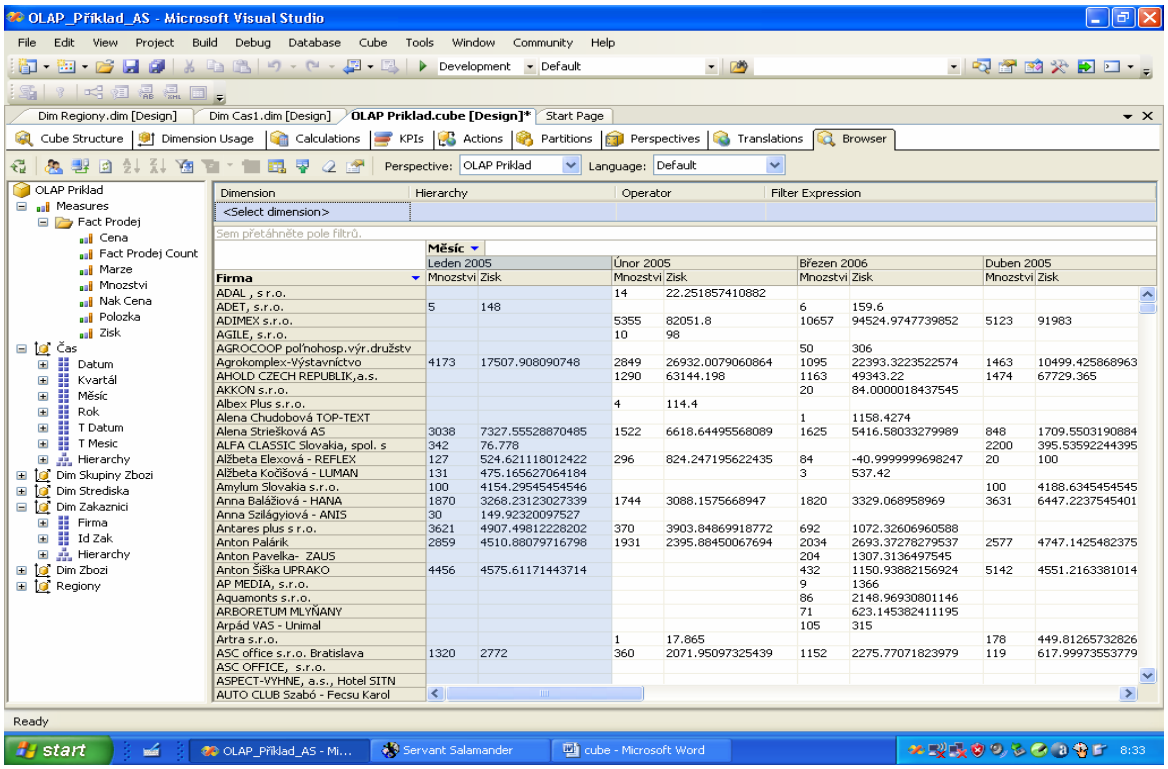


Výsledkem zde je 2-dimenzionální tabulka (cas,region) s 2-dimenzionálními fakty (mnozstvi, zisk).

Jednoduchým rozkliknutím vyšší úrovně dimenze dostaneme detailnější úroveň (leden ⇒ dny):



Obdobně můžeme volit libovolnou jinou kombinaci a umístění dimenzí i faktů:



♦

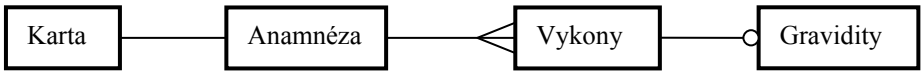
12.2. Datový sklad vytvořený na míru datům

Druhý příklad prezentuje datový sklad vytvořený na míru zadání. Důvodem byly požadované netradiční agregované hodnoty, tzv. ukazatele, potřeba velmi jednoduchého uživatelského rozhraní a konečně integrovaná možnost nastavitelných grafických výstupů – mimo klasické tabulky. Uváděný datový sklad je výsledkem diplomové práce na katedře informatiky FEI VŠB-TUO.

Příklad:

Je dána více než pětadvacetiletá databáze Fakultní porodnice o případech asistované reprodukce (AR = umělého oplodnění). Obsahuje více než 120 atributů a přes 10 000 záznamů. Po dvaceti letech se začala databáze využívat nejen k prosté evidenci, ale i k analýzám pro vědecký výzkum. Mimo jiné byl vytvořen datový sklad a na míru implementovaný systém pro jeho užívání.

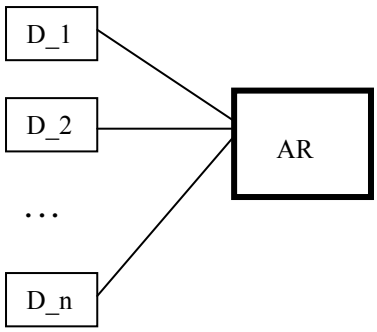
Zdrojová data mají strukturu:



kde Karta a Anamnéza *jsou základní atributy pacientek,* Vykony *popisují charakteristiky vlastních provedených pokusů o AR,* Gravidity *pak zaznamenávají úspěšné pokusy a další atributy o pokračování gravidity.*

Návrh struktury datového skladu nebudeme popisovat podrobně, jde o velmi rozsáhlý seznam dimenzí i faktů. Výsledkem je jediná tabulka faktů AR *a několik desítek dimenzí. Mimo několik klasických agregačních funkcí COUNT a SUM se používají ještě další složitější, které požadoval*

zadavatel a které jsme nazvali Ukazatele. Jsou to hlavně poměry některých faktů (například počet oplozených vajíček na jeden výkon) a procentuální vyjádření některých poměrů faktů (procento počtu oplozených vajíček ku počtu oplozovaných vajíček, procento počtu gravidit ku počtu provedených transferů atd.). Každý ukazatel dostane svůj název. Struktura datového skladu je tedy:



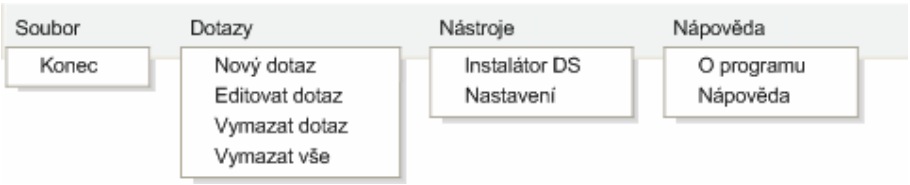
Mimo dělení na dimenze a fakty je v těchto datech další „složitost", a sice rozdělení všech atributů do tzv. „etap" podle stavu výkonu. Celý výkon se provádí v několika časových etapách a každá etapa má své atributy.

Uživatel znalý dat a jejich významu (lékař) tedy může volit postupně kteroukoliv kombinaci podle následujícího schématu. Každý dotaz může pojmenovat a příště jej jen použít bez nového definování.

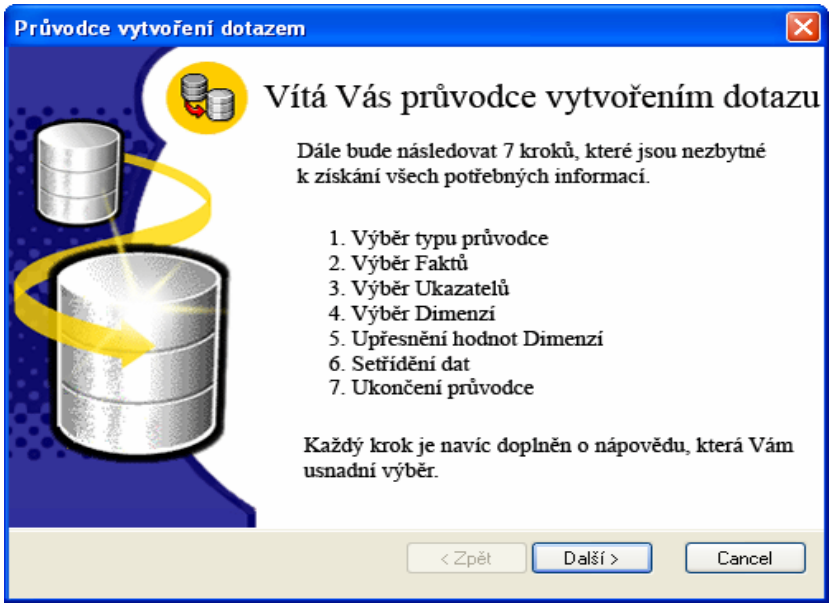
Logický postup formulace dotazu:

1. *uživatel zvolí etapu nebo výběr ze všech atributů*
2. *uživatel zvolí fakty a jejich agregační funkce*
3. *uživatel zvolí ukazatele*
4. *uživatel zvolí 1-2 dimenze*
5. *uživatel může upřesnit (filtrovat) hodnoty dimenzí*
6. *uživatel může definovat setřídění dat.*

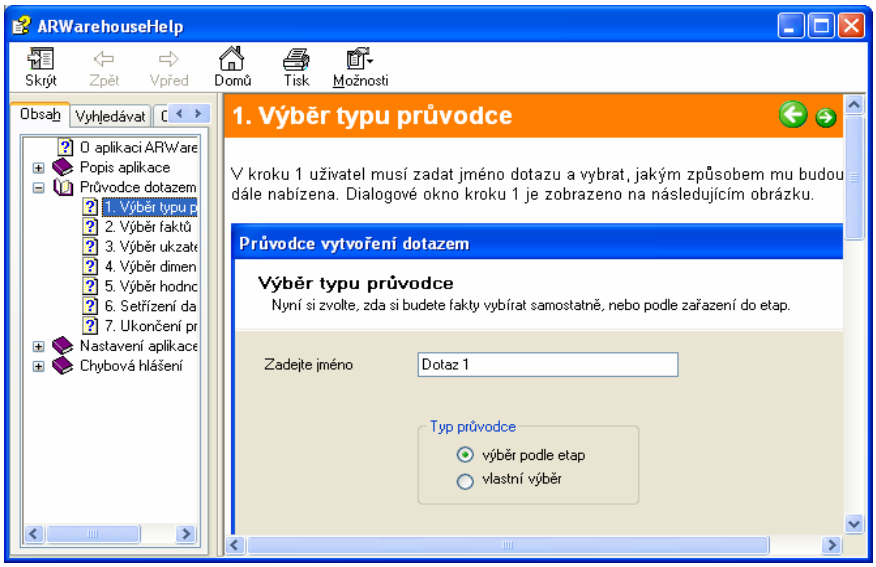
Základní menu programu:



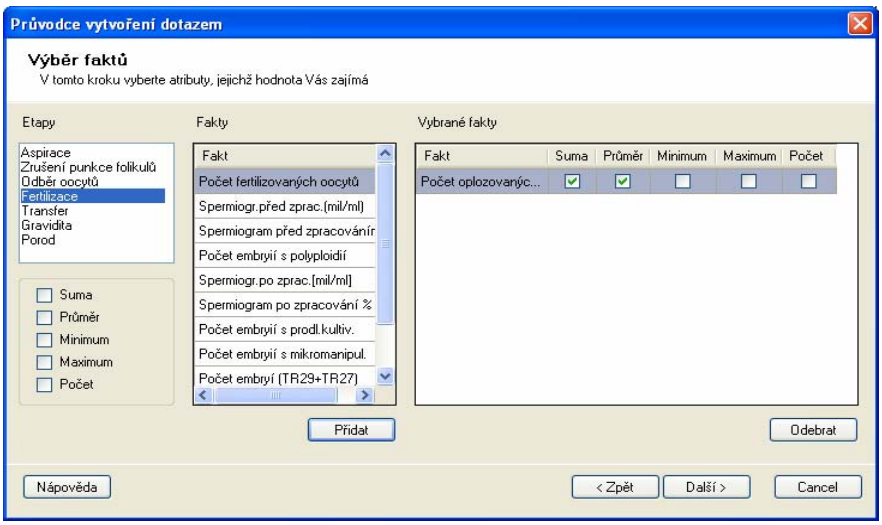
Průvodce dotazem:



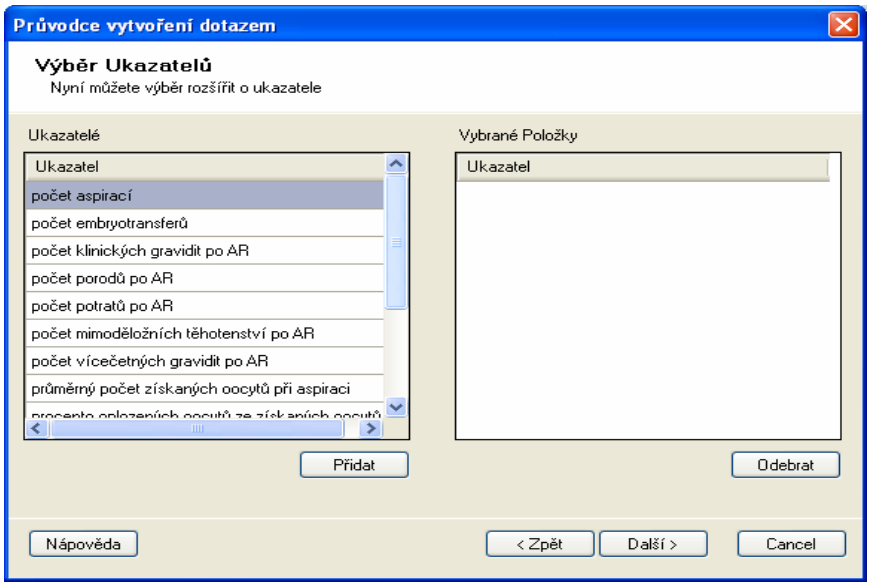
Definice dotazu:



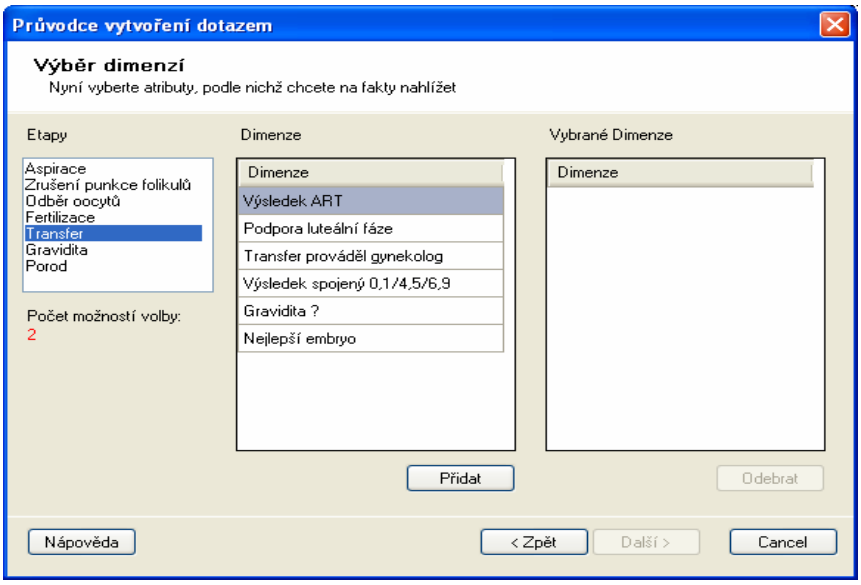
Výběr faktů a jejich agregací:



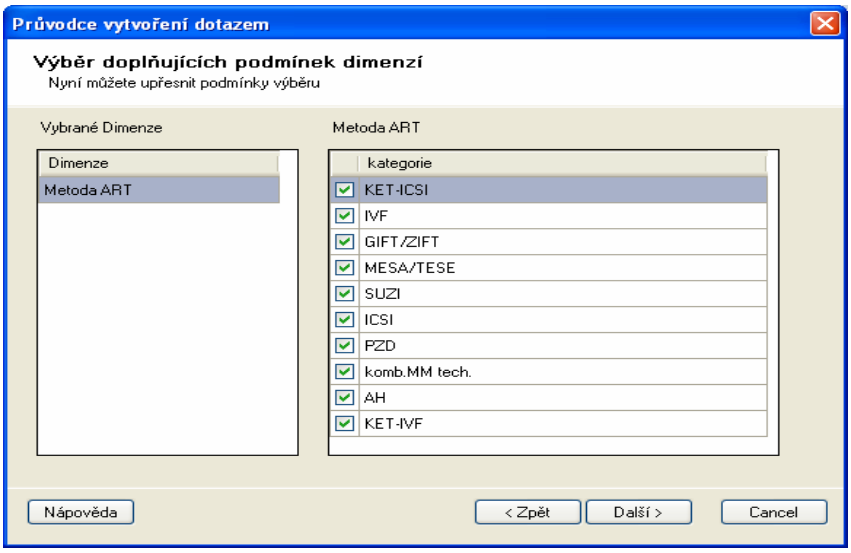
Výběr ukazatelů:



Výběr dimenzí:

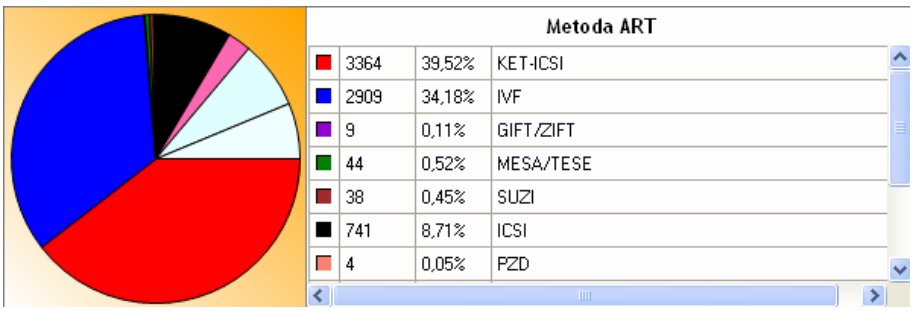


Doplnění filtru na hodnoty dimenzí:

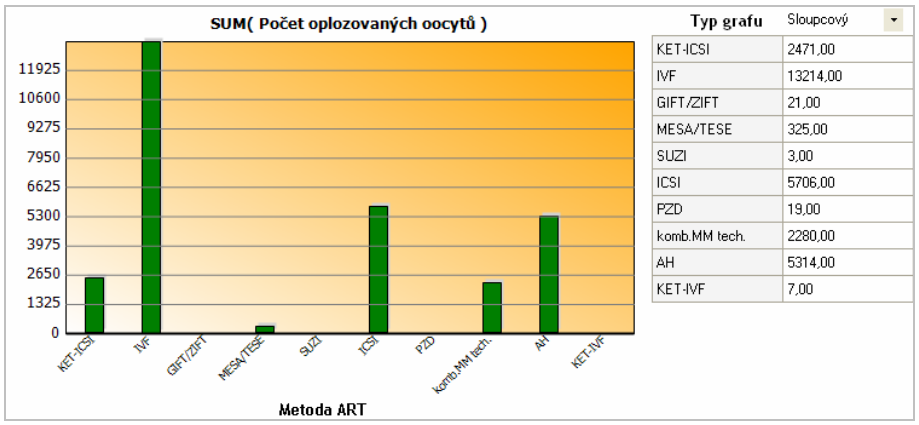


Po případném zadání třídění a po vyvolání definovaného dotazu se výsledek dotazu zobrazí formou tabulky a zvoleného typu grafu:

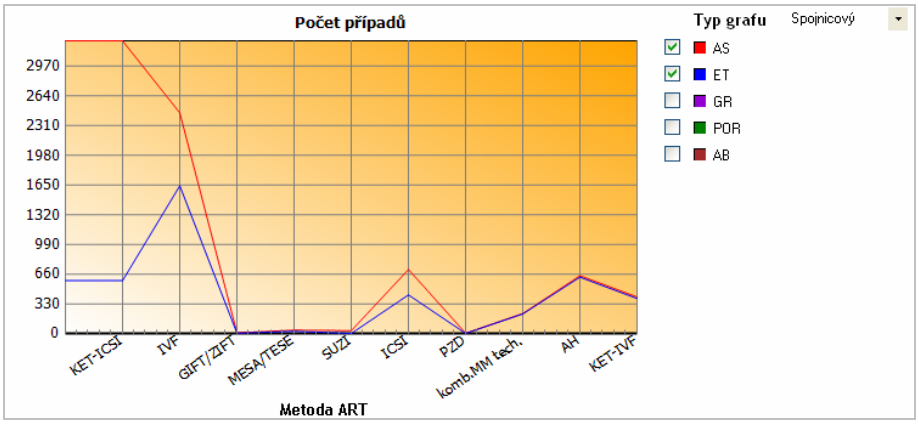
jednoduchý 1-dimenzionální graf koláčový



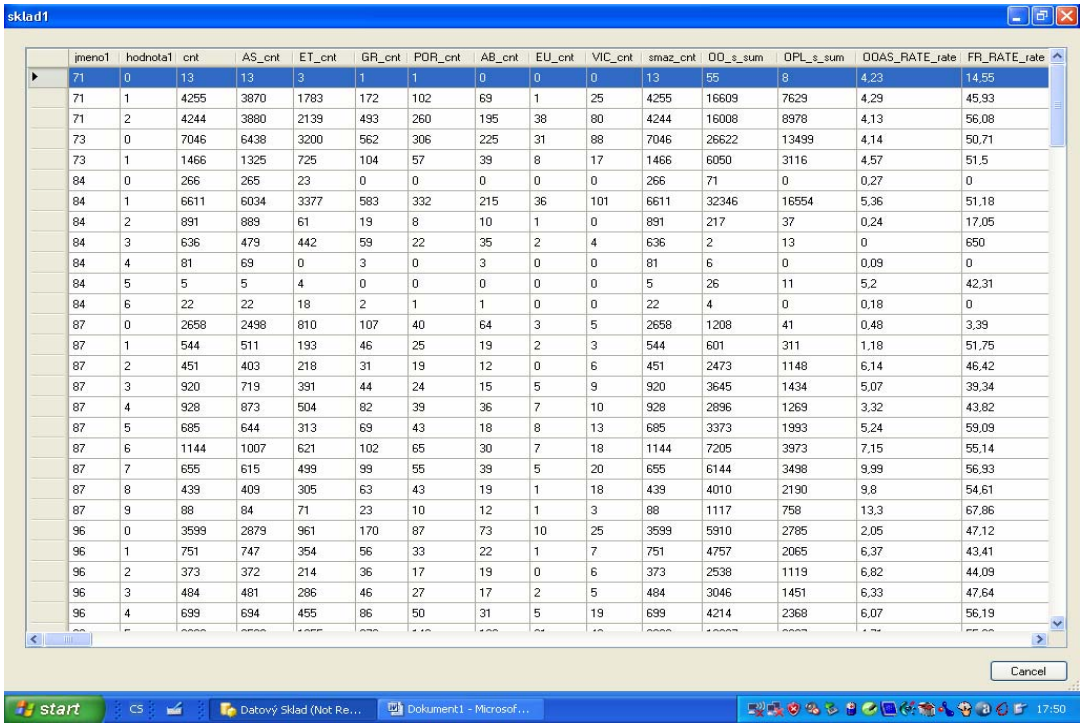
1-dimenzionální graf sloupcový



1-dimenzionální graf spojnicový



1- dimenzionální úplná tabulka faktů:



jmeno1	hodnota1	cnt	AS_cnt	ET_cnt	GR_cnt	POR_cnt	AB_cnt	EU_cnt	VIC_cnt	smaz_cnt	OO_s_sum	OPL_s_sum	OOAS_RATE_rate	FR_RATE_rate
71	0	13	13	3	1	1	0	0	0	13	55	8	4,23	14,55
71	1	4255	3870	1783	172	102	69	1	25	4255	16609	7629	4,29	45,93
71	2	4244	3880	2139	493	260	195	38	80	4244	16008	8978	4,13	56,08
73	0	7046	6438	3200	562	306	225	31	88	7046	26622	13499	4,14	50,71
73	1	1466	1325	725	104	57	39	8	17	1466	6050	3116	4,57	51,5
84	0	266	265	23	0	0	0	0	0	266	71	0	0,27	0
84	1	6611	6034	3377	583	332	215	36	101	6611	32346	16554	5,36	51,18
84	2	891	889	61	19	8	10	1	0	891	217	37	0,24	17,05
84	3	636	479	442	59	22	35	2	4	636	2	13	0	650
84	4	81	69	0	3	0	3	0	0	81	6	0	0,09	0
84	5	5	5	4	0	0	0	0	0	5	26	11	5,2	42,31
84	6	22	22	18	2	1	1	0	0	22	4	0	0,18	0
87	0	2658	2498	810	107	40	64	3	5	2658	1208	41	0,48	3,39
87	1	544	511	193	46	25	19	2	3	544	601	311	1,18	51,75
87	2	451	403	218	31	19	12	0	6	451	2473	1148	6,14	46,42
87	3	920	719	391	44	24	15	5	9	920	3645	1434	5,07	39,34
87	4	928	873	504	82	39	36	7	10	928	2896	1269	3,32	43,82
87	5	685	644	313	69	43	18	8	13	685	3373	1993	5,24	59,09
87	6	1144	1007	621	102	65	30	7	18	1144	7205	3973	7,15	55,14
87	7	655	615	499	99	55	39	5	20	655	6144	3498	9,99	56,93
87	8	439	409	305	63	43	19	1	18	439	4010	2190	9,8	54,61
87	9	88	84	71	23	10	12	1	3	88	1117	758	13,3	67,86
96	0	3599	2879	961	170	87	73	10	25	3599	5910	2785	2,05	47,12
96	1	751	747	354	56	33	22	1	7	751	4757	2065	6,37	43,41
96	2	373	372	214	36	17	19	0	6	373	2538	1119	6,82	44,09
96	3	484	481	286	46	27	17	2	5	484	3046	1451	6,33	47,64
96	4	699	694	455	86	50	31	5	19	699	4214	2368	6,07	56,19

12.3. Využití datových skladů

Popsali jsme podrobně proces analýzy a realizace datových skladů s obecným prohlášením, že jsou užitečné především managementu firem pro jejich kompetentnější rozhodování. Uveďme si nyní podrobněji, k čemu mohou být výstupy z DS užitečné.

❑ Využití informací z DS

S datovým skladem pracuje především datový analytik, který obvykle provádí managementem požadované analýzy, případně si je manager provádí sám.

1. Základním využitím DS je zobrazování dosavadního průběhu sledovaných dat – faktů v dimenzionálních (příp. časových) řadách s možností libovolně přepínat na detailnější nebo naopak globálnější přehledy. Výstupy nejsou závislé na klasických naprogamovaných sestavách z OLTP a vytvářených (sumovaných) po každém dotazu.
2. DS může automaticky nebo na vyžádání analytika podávat důležité informace se sestavami ohlašujícími:
   - výjimky z běžného rozsahu dat, tzv. **analýzy výjimek**;
   - důležité informace pro rozvoj firmy, vyhledávání dosud nevyužitých **příležitostí** firmy – například nízké nebo nulové hodnoty v datech;
   - vyžádané **analýzy predikční** typu „co když“ – extrapolace dat za zadaných podmínek;
   - **varovné informace** při přiblížení se kritickým hodnotám.
3. Informace užitečné pro **prodej a marketing**; na zákazníky je možno pohlížet ne jednotlivě, ale vyhledávat komunity (shluky) podobných zákazníků, obracet se k nim cíleně:
   - **křížový prodej** (nabízet současným zákazníkům i další produkty a služby);
   - **cílený marketing** (použitím vlastních dat + dat ze sčítání lidu + zeměpisných údajů + profilů zákazníků jako věk, pohlaví, stav, druh příjmu, historie prodeje) nabízet cíleně zboží a služby i novým zákazníkům;
   - **speciální nabídky a produktové balíčky** (slučování produktů různých výrobců).
4. Finanční analýzy a řízení
   - **analýzy rizik** (profily zákazníků - banky při přidělování úvěrů, trvale neziskové firmy, nerentabilní sektory, …)
   - analýzy ziskovosti (podle zákazníků, obchodních agentů, časových období, zeměpisných lokalizací, organizačních jednotek, sektorů, …)
5. Řízení **vztahu se zákazníky,** evidence kontaktů, neobtěžovat opakovaně stejnými dotazy z různých oddělení, nezapomínat na cílené nabídky, návody, blahopřání, …
6. **Call centra - počítačová telefonie –** automatické poskytování informací z integrovaných dat DS.
7. **Integrované služby kreditních společností**
   - integrované služby bankám, pojišťovnám, telefonním a dalším společnostem poskytujícím veřejné služby, poskytující negativní informace o zákaznících z mnoha různých podniků;
   - obdobně podávání informací o zaměstnancích – elektronické pracovní knížky.

12.4. Chyby při budování datových skladů

Závěrem DS si uvedeme souhrn publikovaných negativních zkušeností a chyb ze zavádění DS, kterých by se měli vyvarovat manažeři pro DS.

1. **Chybný řetěz sponzorství** - správný řetěz zahrnuje dvě klíčové osoby nad manažerem, který odpovídá za DS: sponzora (dodává peníze do projektu) a tahouna z aplikačního prostředí. Ten by měl mít tři vlastnosti: již dříve získaný respekt, vlastní zdravý skepticismus nad technologií a být rozhodný a pružný.
2. **Stanovení nevhodných očekávání** - ne vše bude vyhovovat uživateli. DS se většinou naplňují agregovanými daty, chce-li uživatel zdrojová data, odpověď je frustrující; frustrace se hodí na hlavu DS manažerovi.
3. **Politicky naivní chování** - často se tvrdí, že DS umožní dělat manažerovi lepší rozhodnutí. Správný manažer se stane nedůvěřivý, DS dělá lepší rozhodnutí než já? Je to jako s jazyky 4GL na konci 70. let a šílenstvím kolem EIS koncem 80. let. Obojí slibovaly lepší přístup k informacím, 4GL vydržely (byly kupovány jako prostředky pro získávání dat), EIS měly rychlý pád (byly ohlašovány jako prostředky pro zlepšení manažerských rozhodnutí).
4. **Předimenzování DS** - nelze přijít s dotazníkem, co by mělo být v DS. Lze tak obdržet příliš mnoho požadavků a příliš málo skutečně potřebného.
5. **Záměna návrhu databáze DS za návrh transakčního systému** - jde o dva zcela různé cíle. U DS se ptáme více na agregované data, trendy, sumy ap. Dotazy jsou často jen jednou, databáze je často nenormalizovaná (ukládání agregací narušuje 3NF) pro jednodušší navigaci potencionálnímu uživateli, DS obsahuje i odvozená data, např. spočítané časové řady ap.
6. **Volba manažera pro DS** - spíše technicky než uživatelsky orientovaného.
7. **Interní data starého stylu (záznamy)** - a ne externí data typu video, obrázky, zvuk (např. uživatel chce vidět obrazovou kopii původního papírového dokumentu a ten není k dispozici).
8. **Překrývání a omyly v definicích dat** - jeden z nejprekérnějších problémů, který se vymstí řídícímu pracovníkovi, který nedodá a neodsouhlasí korektní definice dat.
9. **Víra ve sliby týkající se výkonu** - skutečnost později ukáže, že prostředky nebyly dobře odhadnuty a jsou potřeba další investice. Zvláště se podceňují náklady na síť.
10. **Předsevzít si krátký termín** na vývoj DS - domněnka, že jakmile je DS hotov, všechny problémy skončily. DS je cesta, ne vzdálenost. Uživatelé chtějí stále nová data.
11. **Zaměření se na ad hoc dolování dat a periodické sestavy** - ani to nemusí vést k pokroku. Manažeři často nemají čas vše číst. Lépe je vyvíjet systémy reagující na změny toku dat do DS.

Proč končí řada DS projektů

Pozorování z konferencí 90. let vedlo k závěru, že 50% DS je považováno za chybné či neúspěšné po 18 měsících. Chyba je obvykle založena na některém z následujících faktorů:

- projekt nebyl předán nebo dokončen
- projekt byl dokončen, DS se však stal „mauzoleem“ dat, nikdo ho nemohl použít
- projekt byl nevhodně vyměřen a navržen a postaven v souvislosti s činnostmi podniku
- od projektu bylo upuštěno a začalo se po nové vývojové cestě.

Shrnutí pojmů 12.

Využití datových skladů. Analýzy výjimek, analýzy predikční, varovný systém, analýzy příležitostí, analýzy rizik.

Chyby a chybné představy při budování datového skladu.

Otázky 12.

1. Uveďte alespoň 3 konkrétní příklady využití informací z různých datových skladů.
2. Vyjmenujte obecné možnosti využití datových skladů.
3. Vyjmenujte alespoň část chyb, kterých je možno se dopustit při budování datového skladu.

14. TEXTOVÉ DATABÁZE

Čas ke studiu: 4 hodiny

Cíl Po prostudování této kapitoly budete

- vědět, co jsou textové databáze a v čem je jejich základní rozdíl proti klasickým operativním databázím,
- umět popsat metody předzpracování dat,
- umět popsat metody vyhledávání vzorků v textech.

Výklad

13.1. Textové vyhledávací systémy

❑ Základní vlastnosti textových databází

Textovou databází obecně nazýváme databázi, v níž jsou všechny údaje nebo jejich podstatná část ve formě **volného textu**, téměř bez vnitřní strukturalizace. Existuje několik úrovní textových databází, přecházejících od klasických relačních databází s dlouhými textovými položkami pevného formátu přes rozšíření relačních bází o datové typy OLE, BLOB či MEMO s proměnnou délkou dlouhých textových položek a přes další rozšíření až po tzv. full-textové systémy či hypertexty.

Příklad:

Typickou aplikací jsou knihovní systémy s částečně strukturovanou informací (název knihy nebo časopisu, autor, ISBN, cena, ...) a částečně textovou (abstrakt, celý článek, celá kniha).

♦

❑ Vyhledávání v textech

Hlavní rozdíl mezi textovým a relačním SŘBD je v možnostech textového vyhledávání, v možnostech formulace podmínky selekce. Textové položky jsou rozsáhlé a nestrukturované. Mimo klasické dotazy formulované jako vyhledání shody celého údaje nebo podřetězce je potřeba formulovat i další dotazy, odpovídající požadavku „najdi záznamy, pojednávající o tom a o tom ...“. Obvykle se pak **podmínka selekce formuluje pomocí zadaných slov a jejich kombinací**, které mají být v prohledávaném textu obsaženy.

Elementárním prvkem dotazovacího jazyka v textovém SŘBD je slovo. Slovem můžeme rozumět nejjednodušeji jakékoliv podřetězce, nebo slovní kmeny či začátky slov (pokud neznáme pád, ve kterém se hledané slovo v textu vyskytuje), dále - zvláště v angličtině, kde výslovnost se liší od psaného textu existují i složitější transformace např. SOUNDEX pro vyhledávání „podobně čtených“ ale různě psaných slov, či další „lingvistické procesory“ pro vyhledávání všech gramatických tvarů hledaného slova apod.

V textových databázích se mimo známé logické operátory používají navíc tzv. **distanční či kontextové operátory**, označující, že nějaká slova mají být nalezena těsně za sebou, nebo ne víc než n slov za sebou, v jedné větě, v jednom odstavci, v téže položce apod.

K vyhledávání na základě uvedených rozšířených požadavků musí mít SŘBD nějakou organizaci datových struktur, pracující nad vlastní databází. Nejjednodušší součástí těchto pomocných struktur bývá **slovník** všech slov (slovník klíčových slov), podle kterých se dá v databázi vyhledávat. Protože uživatel nemusí přesně vědět, jakými slovy je ve slovníku popsáno to, co v databázi hledá, je jednou z užitečných možností mu nabídnout při formulaci dotazu tento slovník k prohlížení a vybírání termínů, uživatel je jen spojí vhodnými operátory.

Dalším vývojovým stupněm u výkonnějších textových SŘBD je uspořádání slovníku do složitějších struktur. Jednodušším typem je **tezaurus**, tj. slovník, v němž jsou zachyceny hierarchické vztahy a rovnosti mezi slovy (synonyma).

Komplexnější a komplikovanější je tzv. pojmový strom, **topic tree**, který slouží k mnohem obecnějšímu pojmovému vyhledávání. *(Např. databáze americké firmy Verity na základě slov „řidič“, „křižovatka“, „automobil“ usoudí, že text pojednává o automobilové dopravě.)*

Naznačené pojmové struktury posunují textové vyhledávání na kvalitativně vyšší úroveň, protože osvobozují uživatele od bloudění ve slovech a umožňují přiblížit se vyhledávání skutečně podle toho, o čem texty jsou.

❑ Fulltextové systémy a hypertexty

Chápeme-li textové databáze jako částečně nestrukturované, pak dalším typem jsou fulltextové systémy. Jsou určeny k uchování velkého množství informací v textové podobě a k vyhledávání informací z nich *(typickou aplikací je systém právních informací, zákonů a norem)*.

Rozdíl mezi textovými SŘBD a fulltextovými databázemi není definován přesně. Obecně hypertextové systémy používají efektivnější algoritmy a pomocné datové struktury pro textové vyhledávání. Jsou určeny k uchování plných textů informačních dokumentů, nejen stručnějších výtahů z primárních textů, jako obvykle u textových databází. Na druhé straně se u nich nepředpokládá manipulace s mnoha netextovými údaji. Obecně se u nich nepoužívá pojmů atribut a entita. Bází, v níž se vyhledává je zde prostě text, ne soubor záznamů. Některé hypertextové systémy podporují i víceúrovňové členění textu (dokumenty, kapitoly, odstavce apod.). Obecně platí, že uživatel vyhledává úseky textu kolem hledaných slov.

Další úrovní je hypertext - datová struktura z textových (případně i obrazových a zvukových) částí, nazývaných hypertextové stránky. Stránky jsou provázány vzájemnými odkazy, a to hierarchicky i horizontálně. Hypertextové odkazy představují odkazy typu „viz“. Hypertexty jsou výborným prostředkem pro realizaci příruček, skript, učebnic, encyklopedií.

K vytváření a prohledávání hypertextů slouží hypertextové systémy. Jsou to částečně editory a částečně SŘBD. Tvorbu hypertextu musí předcházet logická analýza zdrojových informací, nesouvisející s databázovým přístupem. Souvislost se projeví až při jeho využívání. Hypertextové prvky musí být spojeny s textovým vyhledáváním (podobně jako vyhledávání v příručce pomocí rejstříku). Obvykle se nejprve vyhledává základní množina dokumentů, která odpovídá dotazu a pak se hypertextovým prohledáváním s používáním odkazů dohledá úplná informace.

Hypertexty jsou doplněny odkazy - „tlačítky“ pro přechody mezi stránkami, pamatují si navíc i cestu, kterou se uživatel dostal k aktuální informaci a umí se po ní vracet.

13.2. Vyhledávací metody v textu

Vyhledávání v textu je operace, při které se zjišťuje, **zda zadaný text obsahuje hledaná slova – vzorky**. Pokud ano, pak nás zajímá i to, kde se v prohledávaném textu vzorky vyskytují.

Vyhledávání se dělí podle několika kritérií.

- Podle počtu vzorků – jeden, několik, nekonečná množina textů zadaná regulárním výrazem.
- Podle počtu vyhledávaných výskytů – první výskyt, všechny.
- Podle vyhledávání jen disjunktních nebo i překrývajících se výskytů vzorků.
- Podle směru prohledávání textu – sousměrné, protisměrné.
- Podle metody vyhledávání – prohledává se vlastní text nebo jeho náhražka (předzpracovaný text), případně se předzpracovává i vzorek.

Poslední kritérium dělí metody vyhledávání v textu na 4 typy podle následující tabulky.

Text \ Vzorek	-	Předzpracování
-	Elementární	Vyhledávací stroj
Předzpracování	Indexování	Signatura

13.3. Elementární algoritmus vyhledávání

Elementární algoritmus postupně přikládá a porovnává jediný vzorek ve všech pozicích textu, ani text ani vzorek nijak předem neupravuje.

Proměnná i ukazuje v případě nalezení vzorku na první znak výskytu vzorku v textu. V nejhorším případě je počet porovnání znaků textu a vzorku V * (T-V+1). U přirozených jazyků však dochází k neshodě znaků většinou velmi brzy, u prvního nebo druhého porovnávaného znaku, počet porovnání se tedy výrazně sníží na C*T, kde C je empiricky zjištěná konstanta. Pro angličtinu je to 1.07, pro češtinu zjištěna není. Praktická časová složitost je tedy lineární. Implementace algoritmu je velmi jednoduchá.

Elementární algoritmus umožňuje sousměrné vyhledávání jednoho vzorku. V případě konečné množiny vzorků je nutno použít algoritmus opakovaně pro každý vzorek, pro vyhledávání nekonečné množiny vzorků je nepoužitelný.

13.4. Indexové metody

Index ve vyhledávacích systémech je obvykle uspořádaná množina slov obsažených v textu. U každého slova v indexu jsou uvedeny odkazy do textu, kde se příslušné slovo vyskytuje (obdobně, jako při konstrukci rejstříku). Indexový soubor tedy tvoří slova tvořící index s identifikací jejich výskytů v textu. Vlastní text je v samostatném souboru.

Celý soubor textový je obvykle rozdělen na části (text - kapitola, odstavec, věta, slovo), které jsou průběžně očíslovány a nazveme je **dokumenty**. Implementace indexů je možná několika způsoby.

❑ Použití invertovaného souboru

Nejnázornější je forma invertovaného seznamu. Základní princip je následující. Z textu se vytvoří tabulka výskytů slov v dokumentech:

	slovo1	slovo2	slovo3	slovo4
dokum1	1	1	0	1
dokum2	0	1	1	1
dokum3	1	0	1	1

Z ní se pak vytvoří invertovaný soubor, v němž jsou navíc slova uspořádána (pro binární hledání).

	dokum1	dokum2	dokum3
slovo1	1	0	1
slovo2	1	1	0
slovo3	0	1	1
slovo4	1	1	1

Pokud je zadán jediný vzorek, provádí se vyhledání binárním půlením.

Pokud je vzorků víc, vlastní vyhledávání se provádí dvojím způsobem. Je-li počet vzorků výrazně menší, než délka indexového souboru, opakuje se pro každý vzorek binární půlení. V opačném případě se používá metody dvojitého slovníku. Množina vzorků se setřídí a vyhledání se provádí metodou dvou uspořádaných množin.

Přiřazení bitového vektoru každému klíčovému slovu, pozice jedniček v tomto vektoru udávají čísla dokumentů, v nichž se klíčové slovo vyskytuje. Dokumenty jsou očíslovány, index je setříděn podle klíčových slov. Při této organizaci indexů je problémem přidávání dokumentu do souboru. Je přitom nutno upravit invertovaný soubor přidáním sloupce a případně přidáním nových slov se setříděním indexu.

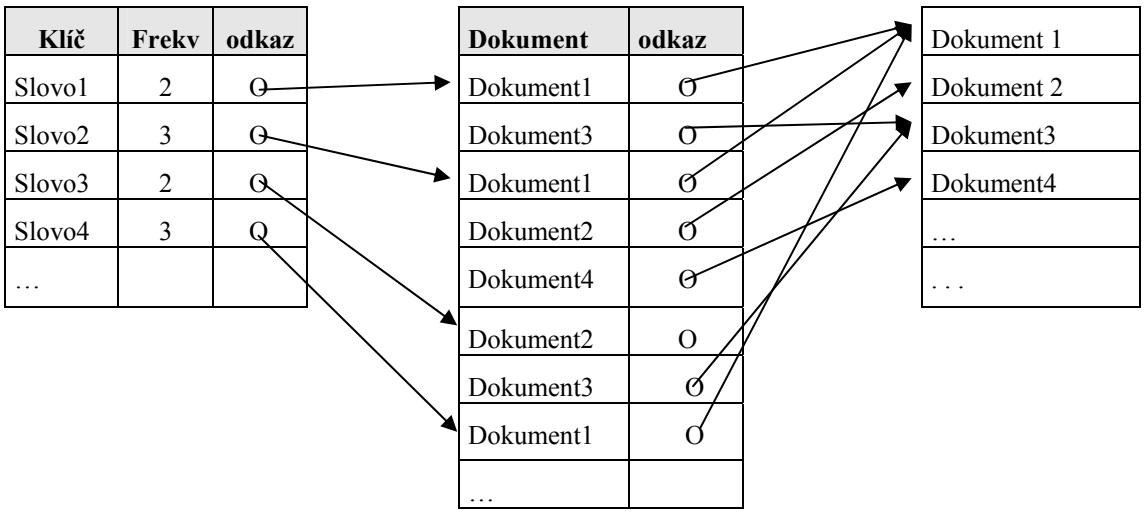
❑ Použití seznamu dokumentů

Místo přiřazení bitového vektoru se každému klíčovému slovu přiřadí seznam čísel dokumentů, ve kterých se slovo vyskytuje.

slovo1	1,3
slovo2	1,2,4
slovo3	2,3
slovo4	1,2,3

❑ Souřadnicový systém indexů s ukazateli

V tomto případě se nepracuje s čísly dokumentů, ale indexový soubor je rozdělen na dvě části: slovník s ukazateli do seznamu čísel dokumentů a seznam čísel dokumentů s ukazateli na dokumenty.



Metody procesu indexování

Při použití indexování je nejsložitějším úkolem nalezení vhodných slov do slovníku indexů. Slova musí co nejpřesněji reprezentovat obsah jednotlivých dokumentů. Zřejmě slovo, které se vyskytuje ve všech dokumentech, nemá z hlediska vyhledávání žádný význam, podobně slovo, které se vyskytne například jen v jediném dokumentu.

Postupu při nacházení vhodných slov pro index nazýváme indexováním. Způsoby indexování můžeme rozdělit na

- **ruční,**
- **automatické.**

Ruční indexování provádí zkušení experti, zejména při zpracování bibliografických dokumentů. Je to práce náročná a proto je snaha tento úkol algoritmizovat. Uvedeme dále jednoduchou metodu automatického indexování, založenou na frekvenci výskytu slov v jednotlivých dokumentech.

Při všech metodách se obvykle definuje určitá množina slov, které se při indexování používat nebudou. Jsou to slova, která mají ve větách jen gramatický význam, jako předložky, spojky, zájmena, pomocná a způsobová slovesa. Seznam těchto slov (tzv. stop seznam, stop list) se buď vytváří ručně, je však možno využít výsledků **analýzy textu**, protože slova ve stop seznamu jsou obvykle vysoce frekventovaná a krátká.

Při indexování se používá slovník slov, která se použijí do indexu. Výběr slov do indexu by měl respektovat tyto požadavky:

1. musí dobře charakterizovat dokumenty z hlediska oblasti zájmu uživatele,
2. užívat slova s dobře definovaným významem, aby dobře charakterizovala jednotlivé dokumenty,
3. musí dávat do souvislostí dokumenty týkající se podobných a souvisejících oblastí,

Dalším kritériem pro klasifikaci indexovacích metod je to, zda je indexování

- **neřízené**, kdy tento slovník není nijak vnitřně strukturován (**pass list**)
- **řízené**, kdy je ve slovníku definována hierarchie slov, příbuznost slov a synonyma (**tezaurus**).

Celkem tedy může být indexování ruční neřízené (pak ke každému slovu je nutno zadávat i všechna jeho synonyma a použít je při indexování), ruční řízené (synonyma jsou uvedena v použitém tezauru a není nutno je zadávat), nebo automatické. O některých technikách pro automatické indexování se zmíníme v dalších odstavcích.

Analýza textu pro konstrukci slovníku

Úkolem indexování je nalezení slov charakterizujících dokument a určení váhy slov vzhledem k jejich důležitosti při identifikaci dokumentu. Slova charakterizující dokument budeme zřejmě hledat především ve vlastním textu dokumentu. Proto se nejprve budeme zabývat analýzou textu, vedoucí k výběru slov do indexu.

Většina automatických indexovacích metod vychází ze skutečnosti, že frekvence výskytu jednotlivých slov má přímou souvislost s jejich významností při identifikaci dokumentu.

Pokud by se jednotlivá slova vyskytovala ve všech dokumentech se stejnou frekvencí, nebylo by možno vybrat žádné z nich jako vhodné pro identifikaci dokumentů. Skutečností je, že se jednotlivá slova vyskytují v textu s různou frekvencí. Vytvoříme-li seznam slov a setřídíme jej podle klesající frekvence jejich výskytů, dostaneme frekvenční slovník. Začátek takového frekvenčního slovníku pro angličtinu je uveden v následující tabulce. Tento slovník byl vytvořen ze souboru dokumentů obsahujících 1 000 000 slov.

pořadí	slovo	frekvence	Pořadí*frekvence
1	the	69971	0.070
2	of	36411	0.073
3	and	28852	0.086
4	to	26149	0.194
5	a	23237	0.116
6	in	21341	0.128
7	that	10595	0.074
8	is	10099	0.081
9	was	9816	0.088
10	he	9543	0.095

Frekvenční slovník lze charakterizovat empirickým Zipfovým zákonem

$$Pořadí * frekvence \cong konstanta$$

Tento zákon je vysvětlován pomocí obecného „principu nejmenšího odporu“, tedy že autor textu raději některá slova častěji opakuje místo používání nových a různých slov.

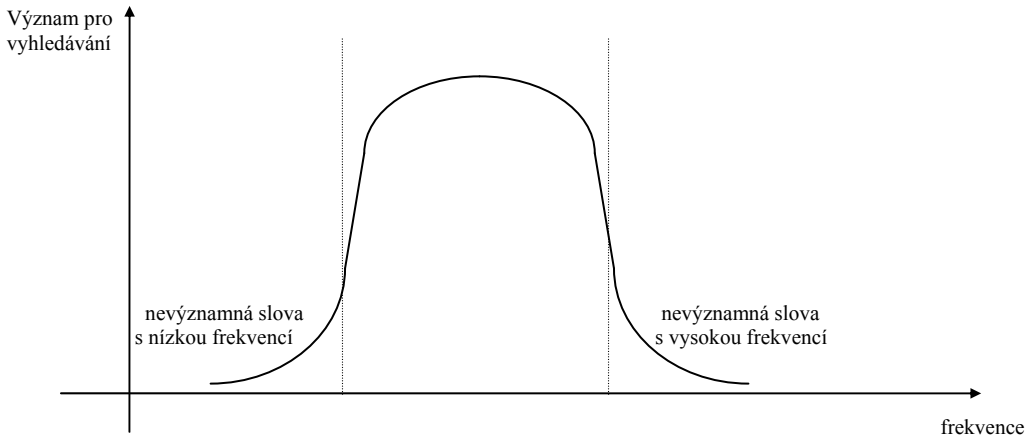
Jednoduchá metoda automatického neřízeného indexování

Frekvenční slovník, Zipfův zákon a jeho důsledky můžeme použít jako východisko pro jednoduchou metodu odvození významnosti slov pro identifikaci dokumentů.

Mějme soubor n dokumentů. Provedeme tyto operace:

1. Vypočteme frekvenci pro každý dokument a každé slovo použité v celém souboru dokumentů.
2. Vypočteme frekvenci pro každé slovo ve všech dokumentech dohromady.
3. Vypočteme frekvenční slovník pro všechna slova. Stanovíme práh pro vyloučení velmi frekventovaných slov ze začátku frekvenčního slovníku. Tím vyloučíme slova velmi frekventovaná, která mají žádný nebo velmi malý význam pro identifikaci dokumentů.
4. Podobným způsobem vyloučíme slova s velmi nízkou frekvencí.
5. Zbývající slova se střední frekvencí jsou vhodná pro zařazení do slovníku.

Tato metoda se opírá o empiricky zjištěnou skutečnost, že slova s nízkou a vysokou mají malý význam pro vyhledávání. Následující graf ukazuje význam slova pro vyhledávání v závislosti na frekvenci jeho výskytu.

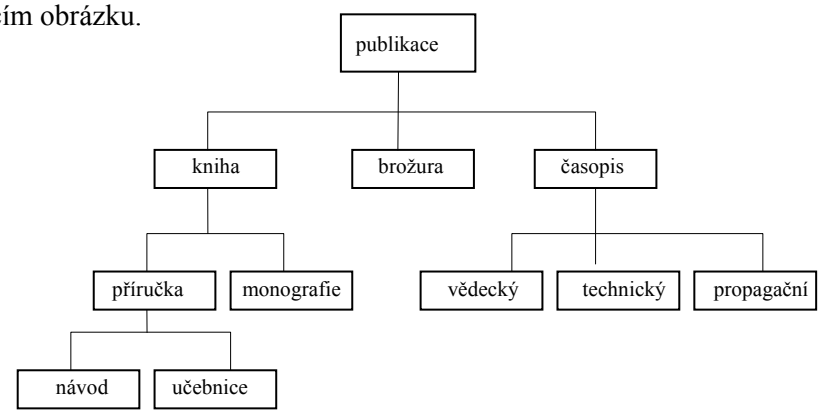


Řízené indexování, tezaurus

Základním prostředkem pro řízení procesu indexování je stanovení indexovacího jazyka, který obsahuje omezený počet slov pro indexování. Takový slovník nazýváme tezaurus a mimo seznam slov standardních obsahuje i hierarchické a asociativní vztahy a vztahy ekvivalence mezi jednotlivými termíny. Tyto vztahy můžeme rozdělit do čtyř skupin:

1. Vazba slova na standardní termín (viz ...). Pro každou množinu ekvivalentních termínů (synonym) je stanoven standardní (preferovaný) termín. Při indexování je vždy vybrán standardní termín bez ohledu na to, jaký termín je použit v dokumentu nebo v dotazu.
2. Vazba standardního slova na příbuzné termíny (viz též ..., related term, RT). Mezi příbuznými slovy jsou uvedeny vazby, příbuzná slova pak tvoří skupiny.
3. Vazba standardního slova na obecnější (širší) termíny (broader terms, BT).
4. Vazba standardního slova na užší termíny (narrower terms, NT).

Příklad grafického znázornění části hierarchické struktury tezauru pro vyhledávání v oblasti publikací je na následujícím obrázku.



Konstrukce tezauru

Tezaurus opět může být konstruován ručně nebo automatizovaně. V každém případě je nutno provést

- rozhodnutí o výběru slov do tezauru (jako u neřízeného slovníku na základě frekvencí slov);
- vytvoření skupin podobných termínů (mohou se použít i termíny s nižší frekvencí a před zařazením do slovníku je sdružit do skupin tak, aby součet frekvencí jednotlivých skupin byl dostatečně velký).

Tezaurus se obvykle vytváří pro řízené indexování v určitém oboru, odtud pak vyplývají další experimentálně potvrzená pravidla pro jeho tvorbu:

1. má obsahovat jen termíny daného oboru,
2. pokud se jako odborné termíny používají slova z jiného oboru, je potřeba jim přiřadit sémantiku obvyklou v daném oboru *(termíny POLE, STROM v informatice),*
3. skupiny příbuzných slov mají tvořit termíny s podobnou frekvencí, součet frekvencí termínů by měl být pro všechny skupiny přibližně stejný,
4. slova s vysokou frekvencí je potřeba vyloučit, pro rozlišení dokumentů mají malý nebo nulový význam.

Vyhledávání pomocí tezauru

Tezaurus se používá nejen pro vytváření a řízení indexovaného slovníku, ale i při vlastním vyhledávání. Při zpracování dotazu se nejprve uživatelem zadané termíny upraví na standardní termíny a podle něj se vyhledají dokumenty. Pak je dále možno na základě požadavků uživatele přidat pro vyhledávání příbuzné, širší nebo užší termíny. Tezaurus je možno graficky znázornit jako strom.

13.5. Metody s předzpracováním vzorků

Dokumenty se při této metodě neupravují, ale vzorek je před vlastním vyhledáváním předzpracován. Výsledkem předzpracování je tzv. **vyhledávací stroj**, který je potom použit pro rychlejší vyhledávání.

Metody se dále dělí na metody se sousměrným a metody s protisměrným vyhledáváním.

❑ Algoritmus sousměrného vyhledávání autorů Knuth-Morris-Pratt

Princip metody s předzpracováním jednoho vzorku a sousměrným vyhledáváním spočívá v tom, že když při porovnání několika znaků textu se vzorkem dojde k neshodě, nemusí se vždy porovnávat znovu tytéž znaky textu se vzorkem posunutým doprava o 1 znak (jako u elementárního algoritmu), ale (protože část znaků textu je již prohlédnuta a známa a je tedy známo, kolik znaků z nich neodpovídá začátku vzorku) obecně o více znaků. O kolik, to udává hodnota tzv. zpětné přechodové funkce h, která je výsledkem předzpracování vzorku.

Nejprve tedy zkonstruuje pro zadaný vzorek zpětnou přechodovou funkci *h*, pak pomocí ní provádíme vyhledávání vzorku v textu.

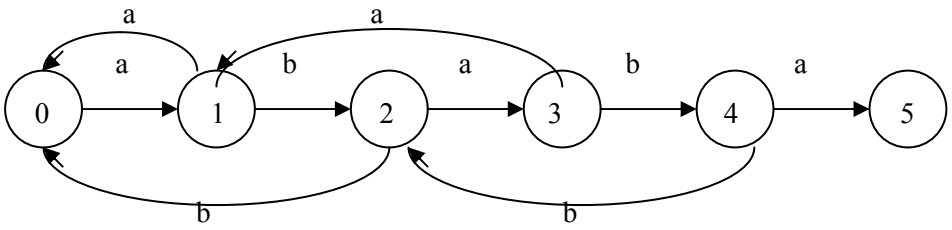
Příklad:

Vzorek nad abecedou T = {a,b} je *ababa.*

Dopředná a zpětná přechodová funkce mají hodnoty dle tabulky:

Stav q	Dopředná funkce g pro vstup a	vstup b	Zpětná funkce h
0	1	0	Nedefinováno
1	Fail	2	0
2	3	Fail	0
3	Fail	4	1
4	5	Fail	2
5			

Přechodový diagram tohoto vyhledávacího stroje je na obrázku.



♦

Existují další varianty tohoto algoritmu pro vyhledávání sousměrné a protisměrné, vyhledávání všech výskytů, nekonečné množiny vzorků.

13.6. Signaturové metody

Signaturou S_x nazýváme bitový řetězec délky m, který je přiřazen textovému řetězci x zobrazením $h(x) \rightarrow S_x$, $S_x \in \{0,1\}^m$. Každý bit signatury S_x vyjadřuje nějakou vlastnost textového řetězce x (existenci některého slova nebo podřetězce v textovém řetězci). Zvolíme-li vhodně zobrazení h, je možno porovnáním signatury textu S_T a signatury vzorku S_V zjistit, zda text může či nemůže obsahovat vzorek. Pokud ano, pak se nějakým algoritmem ověří, zda jej skutečně obsahuje. Tak se

předem z detailního prohledávání vyloučí řada textů, o kterých se porovnáním signatur zjistí, že vzorek neobsahují.

❑ Konstrukce signatur jednotlivých termínů

Signatury jednotlivých termínů můžeme vytvořit těmito způsoby:

1. pomocí vektor výskytu slov, každému termínu přiřadíme jeden bit v signatuře,
2. signaturu termínu vytvoříme pomocí nějaké hašovací funkce.

Příklad:

Signatury pro 6 slov o délce 6 bitů

termín	signatura
data	000001
disk	000010
soubor	000100
informace	001000
systém	010000
páska	100000

♦

Nevýhodou tohoto způsobu je velká délka signatury, protože její délka je dána počtem možných termínů. Navíc vyžaduje pevný počet termínů, protože při změně jejich počtu by se měnila délka signatury. Tyto nevýhody odstraňuje použití hašovací funkce pro výpočet signatur jednotlivých termínů. Pak délka signatury může být konstantní.

Příklad:

Jednoduchý příklad je použití hašovaných k-signatur. Všechny podřetězce délky k (k-gramy) *se použijí jako argumenty hašovací funkce hash. Její hodnotou je číslo z intervalu <1,m>, které udává pořadí příznaku v signatuře, nastaveného na jedničku. Obecně různým podřetězcům odpovídají stejné hodnoty, proto je výsledná signatura jakási komprimovaná verze s částečnou ztrátou informace. Konkrétnímu termínu odpovídající bit odpovídá i jiným termínům, může nastat falešný předvýběr a proto je nutný dodatečný ověřovací přesný výběr některou jinou metodou.*

termín	signatura
data	0001
disk	0101
soubor	1001
informace	0110
systém	0011
páska	1010

♦

❑ Konstrukce signatur dokumentu

Ze signatur jednotlivých termínů vytvoříme signaturu dokumentu

1. řetězeným kódováním
2. vrstveným kódováním

Řetězené kódování

Při řetězeném kódování jsou signatury jednotlivých termínů zřetězeny. Tento postup se hodí jen pro případy, kdy je text nějakým způsobem strukturován a jednotlivé složky jsou uspořádány.

Pro záznam tvaru $z = (a1, a2, \ldots, an)$ *je s*ignatura záznamu $h(z) = h(a1) \,.\, h(a2) \,.\, .\, .\, h(an)$

Při vyhledávání je třeba určit hodnotu příslušných prvků záznamu.

Příklad:

Záznam Kniha = (ISBN, autor, titul), instance záznamu z = (72345, Karel Čapek, Krakatit), signatury jsou voleny například takto:

hodnota	Signatura
72345 Karel Čapek Krakatit	10001001 01011000 10010001
Signatura záznamu St	10001001 01011000 10010001

Při vyhledání knih Karla Čapka bude mít signatura dotazu tvar:

Sv = ???????? 010110000 ????????

Otazníky v dotazu znamenají, že v tomto místě může mít signatura libovolnou hodnotu.

♦

Vrstvené kódování

Při vrstveném kódování jsou signatury jednotlivých termínů logicky sčítány bit po bitu, tedy signatura záznamu $z = (a1, a2, \ldots, an)$ je

$$h(z) = h(a1) \text{ or } h(a2) \text{ or } .\, .\, .\, \text{or } h(an)$$

Příklad:

Záznam o knihách z minulého příkladu

hodnota	signatura
72345 Karel Čapek Krakatit	10001001 01011000 10010001
Signatura dokumentu St	11011001

Při vyhledání knih Karla Čapka bude mít signatura dotazu tvar:

Sv = 010110000

♦

Vrstvené kódování se používá i u nestrukturovaných dokumentů.

Kombinací signatur termínů pomocí vektoru výskytu slov a signatur dokumentu vrstveným kódováním vytvoříme pro každý dokument binární řetězec, kde každý bit odpovídá jednomu z možných termínů. (*Dokument obsahující termíny* data soubor systém *z výše uvedeného příkladu bude mít signaturu 010101, každá jednička odpovídá přesně jednomu termínu).* Tato metoda vede na přesné vyhledávání, pro každý vzorek jsou vybrány právě ty dokumenty, které jej obsahují.

Kombinací hašování pro jednotlivé termíny a vrstveného kódování dostáváme předvýběr se ztrátou informace. Dále při vrstveném kódování se počet jedniček v signatuře záznamu zvětšuje s počtem termínů použitých v dokumentu. Pokud by počet jedniček vzrostl natolik, že téměř všechny bity signatury jsou jedničky, mělo by to negativní vliv na vyhledávání, protože by mu vyhovovaly téměř všechny dotazy a bylo by nutno detailně prohledávat mnoho dokumentů. Experimentálně je zjištěno, že z hlediska vyhledávání je nejvhodnější signatura s 50% jedničkami.

Z toho důvodu je třeba rozdělit rozsáhlé dokumenty na bloky tak, aby počet jedniček v jejich signaturách nebyl velký. Jsou dvě možnosti: rozdělit dokument na bloky

1. stejné délky (FSB = Fixed Size Blocks),
2. stejné váhy (FWB = Fixed Weight Blocks), kdy se signatura vytváří postupným přidáváním jednotlivých termínů a blok se ukončí, když počet jedniček dosáhne 50%.

❑ Vyhledávání pomocí signatur

Pomocí techniky signatur se tedy u nestrukturovaných dokumentů používá většinou hašovací funkce pro určení signatur termínů a vrstvené kódování pro určení signatury dokumentu.

Vyhledání dokumentu obsahujícího zadaný vzorek pak znamená

1. určit signaturu vzorku:
   Signatura vzorku se určí stejným postupem, jako u dokumentu. Máme nyní Sv a St.
2. vyhledat dokumenty, které by mohly obsahovat vzorek (tj. vyloučit, které jej neobsahují),

   Dokument může obsahovat vzorek, když

   $$St \text{ and } Sv \equiv Sv$$

   Prakticky místo tohoto testu je výhodnější použít pro implementaci jednodušší vztah

   $$(St \text{ and } Sv \neq Sv) \equiv (\text{not } St \text{ and } Sv)$$
3. některým přesným vyhledávacím algoritmem určit dokumenty, které vzorek obsahují.

   Dohledání se provede například elementárním algoritmem, vyhledávacím strojem nebo pomocí indexu, který určí polohu vzorku v dokumentu.

Příklad:

Signatura vzorku Sv = 1010
signatura textu St = 1010 (*shodná s* Sv) St = 1000 (*užší než* Sv) St = 1110 (*širší než* Sv)

1. Sv	1010	1010	1010
St	1010	1000	1110
Sv ∧ St	1010	1011	1010
Sv	1010	1010	1010
Sv∧St⇔Sv	1111 ... *vyhovuje*	1000 ... *nevyhovuje*	1111 ...*vyhovuje(test na* 1111)
2. Sv	1010	1010	1010
¬ St	0101	0111	0001
¬ St ∧Sv	0000 ... *vyhovuje*	0010 ... *nevyhovuje*	0000 ...*vyhovuje(test na* 0000)

♦

❑ Kombinace signatur a indexu

Je tedy možné zkombinovat signaturu s indexováním a ke každé signatuře připojit seznam ukazatelů do textového řetězce. Pak v signatuře není jediný bit, ale také ukazatel na začátek seznamu ukazatelů. Nevýhodou této metody je značná paměťová náročnost a časová složitost přitom není menší.

13.7. Jazyky pro vyhledávání v textu

Zřídka se vyhledávají dokumenty obsahující jediné klíčové slovo, obvykle jen pro nová nebo zvlášť specializovaná slova (např. tezaurus). Často může být na běžné klíčové slovo vyhledána téměř celá databáze (např. výpočet) a taková odpověď je fakticky stejná, jako žádná. Proto je nutné dotazy vylaďovat, aby jich nebylo příliš, ale pokud možno všechny potřebné. Tak je dotaz složitější a kromě klíčových slov obsahuje **operátory,** ty dělíme na **logické a poziční**.

1. **Logické** operátory jsou **and**, **or, xor** a **not**, ovšem jejich interpretace je poněkud jiná, než v logice.

Výraz	význam
A and B	dokumenty, ve kterých se vyskytuje jak slovo A, tak slovo B
A or B	dokumenty, ve kterých se vyskytuje slovo A nebo slovo B nebo obě
A xor B	dokumenty, ve kterých se vyskytuje slovo A nebo slovo B, ale ne obě
A not B	dokumenty, ve kterých se vyskytuje slovo A a nevyskytuje slovo B

   Odtud vidíme, že operátory **and**, **or** a **not** budou implementovány pomocí množinových operací průnik, sjednocení a množinový rozdíl a že operátor **not** je binární.

   Pokud nejsou použity závorky, priorita operací bývá definována různě, obvykle:

   zleva doprava,
   and, (or, xor), not

   Operátor not není komutativní, obvykle se pro stejné operátory provádí zleva doprava.

2. **Poziční** operátory předepisují, v jakém pořadí musí být klíčová slova v textu umístěna. Základní poziční operátory jsou **adj** (adjacent), **(n) words**, **with**, **same**, **syn**.

Výraz	význam
A adj B	dokumenty, ve kterých se vyskytuje jak slovo A následované slovem B
A (n) words B	dokumenty, ve kterých se vyskytuje slovo A a nejdále o n slov za ním slovo B
A with B	dokumenty, ve kterých se vyskytují slova A a B ve stejné větě
A same B	dokumenty, ve kterých se vyskytují slova A a B ve stejném odstavci
A syn B	definuje, že slovo A a B jsou synonyma

   Priority opět bývají definovány různě, většinou však poziční operátory mají vyšší prioritu, než logické. Mezi sebou pak nejčastěji v tomto pořadí: **adj**, **(n) words**, **syn**, **with**, **same**.

3. Dalším rysem vyhledávacích jazyků bývá možnost využití **hvězdičkové konvence** pro zadání klíčových slov pomocí symbolů ***** (= libovolný řetězec) a **?** (= libovolný znak). Mohou být použity kdekoliv ve slově (na začátku, uprostřed i na konci). To je důležité zejména u ohebných jazyků při skloňování a časování, jako v češtině.

4. Pokud systém pracuje s tezaurem, je možno vyhledávat s použitím konceptu (tématu, topic). K tomu se užívají operace

výraz	význam
BT(A) = broader term	vybere z tezauru širší termín k termínu A
NT(A) = narrowed terms	vybere z tezauru užší termíny k termínu A
PT(A) = preffered term	vybere z tezauru preferovaný termín k termínu A
SYN(A) = synonyms	vybere z tezauru všechna synonyma k termínu A
RT(A) = related terms	vybere z tezauru všechny příbuzné termíny k termínu A
TT(A) = top term	vybere z tezauru nejširší termín

U operací NT a BT je možno ještě specifikovat, o kolik úrovní výše nebo níže se vybírají termíny z tezauru. Parametry operátorů jsou shrnuty v tabulce:

operand	zápis v textovém výrazu
termín širší o jednu úroveň	BT(‘řetěz’)
termíny širší o n úrovní	BT(‘řetěz’, n)
termíny širší všech úrovní	BT(‘řetěz’, *)
termín užší o jednu úroveň	NT(‘řetěz’)
termíny užší o n úrovní	NT(‘řetěz’, n)
termíny užší všech úrovní	NT(‘řetěz’, *)

Jako zjednodušený příklad použití uvedených vlastností operací jazyka pro vyhledávání v textových systémech uvedeme rozšíření jazyka SQL, jak je definován v systému ORACLE (SQL*TextRetrieval).

SELECT specifikace položek
FROM specifikace tabulek
WHERE položka CONTAINS textový výraz

Příklad:

SELECT title
FROM book
WHERE abstract CONTAINS ‘cluster’

♦

Příklad:

Výběr jmen zaměstnanců s vysokoškolským vzděláním, kteří ovládají angličtinu a němčinu a napsali knihu, příručku, monografii, návod nebo učebnici (s použitím výše uvedeného tezauru a pokud k termínu univerzita jsou uvedeny příbuzné termíny s označením vysokých škol).

SELECT jméno
FROM zaměstnanec
WHERE vzdělání CONTAINS RT(univerzita) AND
jazyky CONTAINS ‘angličtina’ AND ‘němčina’ AND
publikace CONTAINS ‘kniha’ OR NT(‘kniha’)

♦

Shrnutí pojmů 13.

Text, vzorek. Předzpracování textu, předzpracování vzorku.
Vyhledávací stroj, indexování textů, signatury.
Slovník vzorků, tezaurus. Automatická tvorba slovníku.
Operátory logické, poziční. Funkce pro vyhledávání v textu.

Otázky 13.

1. Které typy metod pro vyhledávání informace v textech rozeznáváme?
2. Jak se předzpracovává informace pomocí indexování?
3. Jak se vyhledává informace pomocí indexů?
4. Jak se předzpracovává informace pomocí signatur?
5. Jak se vyhledává informace pomocí signatur?
6. Co je tezaurus a k čemu slouží při vyhledávání informací v textu?
7. Jak se sestavuje tezaurus?
8. Jak se formulují SQL dotazy pro prohledávání textu?
9. Které funkce pro práci s textem se používají v SQL?
10. Které operátory pro práci s textem se používají v SQL?
11. Jaké typy operátorů pro vyhledávání v textu se používají?

LITERATURA

[DK07] DATAKON'2000 - DATAKON'2007. *Sborníky seminářů*. Brno, 2000 – 2006

[DK88] DEMNER J., KRÁL J.: *Softwarové inženýrství*. Skripta MFF UK, Praha 1988.

[DR97] DRBAL P.: *Objektově orientované metodiky a technologie*. Skripta VŠE FIS, Praha 1997.

[DS99] DATASEM'92 - DATASEM'99. *Sborníky seminářů*. CS-Compex, Brno, 1992-1999.

[HM02] Humphries M., Hawkins M.W., Dy M.C.: Data warehousing. Computer Press Brno 2002. ISBN 80-7226-560-1

[HO74] HOŘEJŠ J.: *Strukturované programování.* In Sb Sofsem 74. VVS OSN Bratislava, 1974.

[LA03] LACKO Ľ.: Datové sklady, OLAP a dolování dat. Computer Press Brno 2002. ISBN 80-7226-969-0

[VO02] VONDRÁK I.: *Úvod do softwarového inženýrství*. Skripta VŠB-TU, Ostrava 2002.

[MA02] MATIAŠKO K.: Databázové systémy. EDIS – vydavateľstvo Žilinské univerzity. Žilina 2002. ISBN 80-7100-968-7

[MD06] MODERNÍ DATABÁZE. *Sborníky seminářů*, Dům techniky, Ústí nad Labem, 1995-2006.

[ME94] MELICHAR B.: Textové informační systémy. Skripta ČVUT FEL, Praha 1994. ISBN 80-01-01206-9

[PE03] PENDER T.: *UML bible*. Indianapolis. Wiley, 2003. ISBN 0-7645-2604-9

[PO92] POKORNÝ, J.: *Databázové systémy a jejich použití v informačních systémech.* Academia, Praha, 1992

[PO94] POKORNÝ, J.: *Dotazovací jazyky*. Science, Praha, 1994.

[PO97-1] POKORNÝ, J.: *Základy implementace souborů a databází.* Skripta MFF UK, Praha 1997.

[PO97-2] POKORNÝ, J.: *Datové sklady očima databázisty*. In Sb Moderní databáze. Dům techniky Ústí s.r.o. 1997, str. 1-12.

[PO98] POKORNÝ, J.: *Modelování datových skladů očima databázisty*. In Sb Moderní databáze. České manažerské centrum Čelákovice 1998, str. 1-12.

[RS96] RICHTA K., SOCHOR J.: *Softwarové inženýrství I.* Skripta ČVUT FEL, Praha 1996

[RS93] RICHTA K., SOCHOR J.: *Projektování programových systémů.* Skripta ČVUT FEL, Praha 1993.

[SK07] SKÁLA Z.: *Datový sklad a OLAP pro asistovanou reprodukci*. Diplomová práce VŠB-TU FEI, Ostrava 2007.

[ST05] STANEK W. R.: Microsoft SQL Server 2005. Microsoft Press. Redmond Washington 2005. I