# DATABÁZOVÉ SYSTÉMY
## prednášky

**Martin Drlík - Ján Skalka**

**Nitra 2007**

## Rozdiel medzi údajom a informáciou

Skúsme sa zamyslieť nad otázkou, čo má na svete najvyššiu hodnotu. Sú to peniaze? Zlato? Potraviny? Voda? Žgrloš bažiaci po majetku by zrejme súhlasil s prvými dvoma možnosťami. Hladujúci žobrák by sa priklonil k potravinám a Berberi na saharskom slnku by najskôr ocenili vodu. Každý si najviac váži to, čoho má nedostatok.

Ale skúsme sa zamyslieť nad spôsobom, akým sa dostane boháč k peniazom, žobrák k potravinám alebo Berber k vode. Skôr, ako sa vyberú za svojím cieľom, potrebujú vedieť, kam sa majú vybrať – musia dostať (alebo získať) **informáciu**. A práve informácia je nevyhnutnosť, ktorú potrebuje na dosiahnutie cieľa každý človek, rovnako ako aj zviera alebo stroj (napr. počítač). Pokiaľ je informácia nesprávna, výsledok zvyčajne nie je uspokojivý.

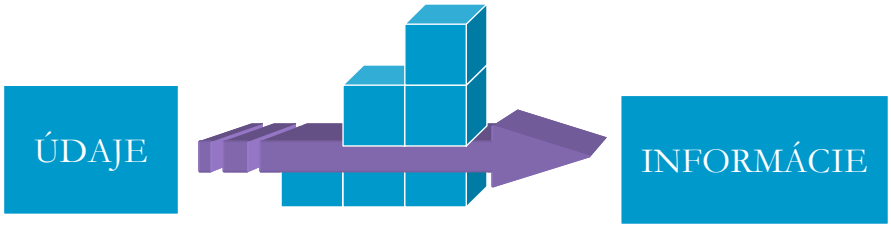
Predstavte si, že niekto vám oznámi, že v nasledujúci deň bude škola zatvorená. Pokiaľ je informácia pravdivá, môžete pokojne zostať doma, ale ak si z vás informátor vystrelil alebo vás úmyselne oklamal, budete mať problémy. Informácie si teda musíte pred ich využitím najprv **overiť**.

Informácie majú niekedy (takmer vždy) vysokú hodnotu. Stačí sa vrátiť o niekoľko desaťročí nazad, keď počas druhej svetovej vojny pracovalo na oboch stranách množstvo špiónov, ktorých cieľom bolo práve získavanie informácií (či už o pohybe vojenských jednotiek, o vybavenosti nepriateľa, jeho plánoch alebo úskokoch). Bez nich by sa velenie vojsk ťažko rozhodovalo o ďalších krokoch, presunoch alebo stratégiách. Bez včasných a spoľahlivých informácií by možno povojnové rozdelenie sveta vyzeralo úplne inak.

Nie je všetko zlato, čo sa blyští a rovnako nie je všetko, čo počujete alebo vidíte informácia. Pokiaľ chceme presne definovať informáciu, musíme rozlišovať pojmy informácia a údaj. Za údaj možno považovať všetko čo vidíte, počujete alebo cítite. **Informácia** je len tá **časť údajov, ktorá je dôležitá pre prijímateľa, a ktorej rozumie - znižuje jeho nevedomosť a neistotu pri rozhodovaní**. Informácia preto musí obsahovať niečo, čomu príjemca rozumie. Dochádza preto k spracovaniu údajov (Obr. 1). Ostatné údaje (ktorým adresát nerozumie) možno považovať len za šum, ktorým je informácia obalená. Napr. veta *„I am hungry“* informuje adresáta o stave žalúdka odosielateľa len v prípade, ak adresát vie po anglicky. Inak pre neho informáciou nie je - nerozumie jej.

To isté platí aj pre zápis $2^2 = 4$. Tento text je informáciou len pre žiaka, ktorý už čosi o mocninách počul. Pre prváčika, ktorý ledva dokáže prečítať prvých desať čísel hovorí len to, že dva, dva o čosi vyššie, rovná sa štyri.

V praxi pojmy údaj a informácia často nadobúdajú aj iné významy (závisí od oblasti alebo vedy, v ktorej sa používajú) alebo naopak, veľmi často sa oba pojmy stotožňujú.




Obr. 1 Po spracovaní sa z údajov stanú informácie

Význam informácií v ostatných desaťročiach výrazne narástol. Nemalou mierou k tomu prispel prienik počítačov do bežného života. Veda, ktorá sa zaoberá zbieraním, uchovávaním, spracúvaním a využívaním údajov (informácií) sa nazýva **informatika**. Informatika je jedna z najmladších vied, ktorá sa veľmi rýchlo vyvíja. Počas niekoľkých desaťročí prenikla do všetkých ostatných vied a veľmi často urýchlila ich napredovanie.

## Čo je to databáza, databázový systém a je vôbec medzi nimi rozdiel?

Skôr ako si vysvetlíme základné pojmy a zákonitosti, ktoré sú zaužívané v oblasti vedy zaoberajúcej sa spracovaním údajov, je potrebné vedieť odpovedať na hore uvedenú, napohľad jednoduchú, otázku.

**Databázová terminológia** je nestála, podobne ako v ostatných oblastiach informatiky, jej rýchly rozvoj a využitie v praxi nedovolili najprv definovať pojmy a pravidlá, ktoré by boli pre všetkých

záväzné, ale vznikli neskôr ako výsledok určitého kompromisu. Preto sa možno stretnúť s rôznymi pojmami, ktoré popisujú jednu a tú istú „vec“.

K týmto pojmom patrí aj pojem **databáza**, ktorým možno na jednej strane nazvať telefónny zoznam, na strane druhej napríklad SQL server, GIS a pod. Pojem databáza môžeme v užšom slova zmysle chápať ako tabuľku - zoznam, v ktorom sú údaje uložené v stĺpcoch a usporiadané podľa zvoleného kritéria. Toto je však veľmi laické vysvetlenie, s ktorým nemôžeme byť spokojní. V širšom zmysle pojem databáza zahŕňa nielen tabuľku, ale aj celý rad operácií, tlačových zostáv, nastavení a výberov.

Jedna z presnejších definícii znie:

**Databáza je súbor informácií ako sú znaky, čísla, diagramy, ktorého systematická štruktúra umožňuje, aby sa tieto informácie dali vyhľadávať pomocou počítača.**

Hoci tejto definícii vyhovujú aj množiny dátových súborov, známe ako **hromadné spracovanie údajov**, ich nevýhodou je, že sa zložito udržujú, hrozí riziko vzniku redundancie a nekonzistencie údajov, ako aj problémov s integritou, ochranou údajov a viacpoužívateľským prístupom. Navyše, na vyhľadávanie v takýchto systémoch je potrebné naprogramovať vždy na mieru „ušité“ metódy zadávania požiadaviek.

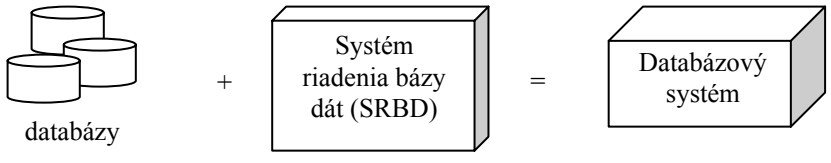
Na druhej strane, pri použití databázového prístupu sú definície dát a ich údržba **oddelené** od používateľských programov, t.j. v dobre naprogramovanej aplikácii môžeme po malom zásahu do zdrojového kódu, v profesionálnejších riešeniach jednoduchou úpravou niekoľkých parametrov, úplne zmeniť miesto uloženia našich údajov. Údaje sú v tomto prípade uložené v komplikovanejšej, centrálne spravovanej štruktúre, ktorú nazývame **databáza**. Databáza obsahuje štyri základné komponenty:

- **dátové prvky**, ktoré zachytávajú elementárne hodnoty, t.j. vlastnosti v databáze popisovaného objektu (meno a priezvisko, aprobácia, rodné číslo študenta).
- **vzťahy medzi dátovými prvkami**, ktoré sú definované zložitejšími dátovými štruktúrami (napr. študent Miro Mokrý má kombináciu INF-Tv možno zachytiť pomocou dátovej štruktúry záznam s prvkami (Miro Mokrý, INF-Tv))
- **integritné obmedzenia,** môžeme si ich predstaviť ako explicitne zadané podmienky, ktoré musia spĺňať údaje uložené v našej databáze (vek v rozmedzí 0-150, výška > 0, a pod.)
- **schému databázy**, čiže popis dát na úrovni, ktorej rozumie používateľ databázy.

Už vieme, že počítač neslúži na počítanie (alebo len na počítanie), ale skôr ide o stroj určený na spracúvanie informácií. Pod slovom spracúvanie sa skrýva **evidovanie** (zaznamenávanie, uchovávanie), **selektovanie** (výber vhodných údajov), **výstup** (najčastejšie na tlačiareň) **údajov** a rôzne **matematické operácie.**

V počítačovej oblasti je na takúto prácu určená zvláštna skupina programov, ktorým sa hovorí **databázové systémy.** Informácie, ktoré obsahujú, sú uchovávané v zoznamoch (zoznam si možno v jednoduchom prípade predstaviť ako jeden súbor) **- databázach**.

Centrálna správa databáz je organizovaná prostredníctvom špeciálneho programového vybavenia, známeho pod pojmom **systém riadenia bázy dát (SRBD)**. SRBD spolu s databázou tvoria




Obr. 2 Vzťah medzi základnými pojmami databáza, databázový systém a SRBD

**databázový systém (DBS)** (Obr. 2)**.**

## Charakteristiky databázových systémov

Každý DBS nasadený do praxe by mal spĺňať nasledovné požiadavky:

- **nezávislosť údajov** – údaje sú nezávislé na aplikácii, dokáže ich prečítať (viacero) DBS aj bez konkrétnej aplikácie. Vďaka nej možno pomerne jednoducho a bez následkov meniť aj organizáciu celkovej štruktúry. Zmena rozsahu celého čísla z 2 na 4 byty neurobí nič, kým pascal....
- **zdieľanie údajov aplikáciami aj užívateľmi** – údaje sú spoločné a ak sú už v báze dát (alebo len v počítači), môžu ich využiť všetky aplikácie, ktoré to potrebujú. Na tomto mieste musíme spomenúť aj **redundanciu** (nadbytočné, viacnásobné zadávanie rovnakých údajov), ktorú vďaka tomu môžeme minimalizovať. Redundancia je žiaduca len v prípade zálohovania a niektorých formách zabezpečenia, vtedy sa nazýva **riadená redundancia**. Vďaka zdieľaniu je možné vytvárať centrálne databázy, ktorých výhody sú nesporné a jedinou nevýhodou je vytvorenie nového pracovného miesta (resp. funkcie) – administrátora = správcu bázy údajov
- **utajenie** – nie všetky údaje sú vhodné pre všetkých. DBS preto obsahujú funkcie utajenia, pomocou ktorých je možné obsluhovať aj protokolovať prístup k údajom. Používatelia dostávajú práva na čítanie a zápis k jednotlivým objektom databázy, napr. k položkám, vetám, poliam, tabuľkám alebo pohľadom.
- **ochrana integrity** – je, ak si pomyslíme, čo by sa mohlo stať vymazaním jedného bajtu v súbore s hlavičkou tabuľky, veľmi dôležitá. **Integrita** je stav, v ktorom sú údaje v plnom rozsahu prístupné a využiteľné v aplikačných programoch a medzi hodnotami platia vzťahy, ktoré boli stanovené, aby prispeli k zaručeniu sémantickej korektnosti bázy dát
- **odolnosť proti chybám - možnostiam narušenia** – DBS musí mať definované postupy, ako sa správať v prípade hardvérovej chyby, chyby aplikačných programov, vírusov, ...
- **zálohovanie údajov** – databázové systémy poskytujú kopírovanie bázy dát alebo jej častí na záložné médium, a opačne – v prípade zlyhania spätné kopírovanie. Kopírovanie celej bázy dát každý deň je zrejme nepredstaviteľné, preto sa využíva **žurnálové zálohovanie** – len položky, ktoré boli zmenené. Navyše, existujú algoritmy zabraňujúce nevhodnému aktualizovaniu položiek – prepísaniu starými údajmi – pomocou **príznakových bitov.** Podrobnejšie o zálohovaní a odolnosti voči chybám pojednáva samostatná kapitola.
- **náhodný prístup** – DBS umožňujú využívať nielen naprogramované funkcie, ale realizovať aj náhodné požiadavky pomocou používateľských jazykov (SQL), ktoré sú jednoduché a neprocedurálne, dokážu ich používať aj neprogramátori

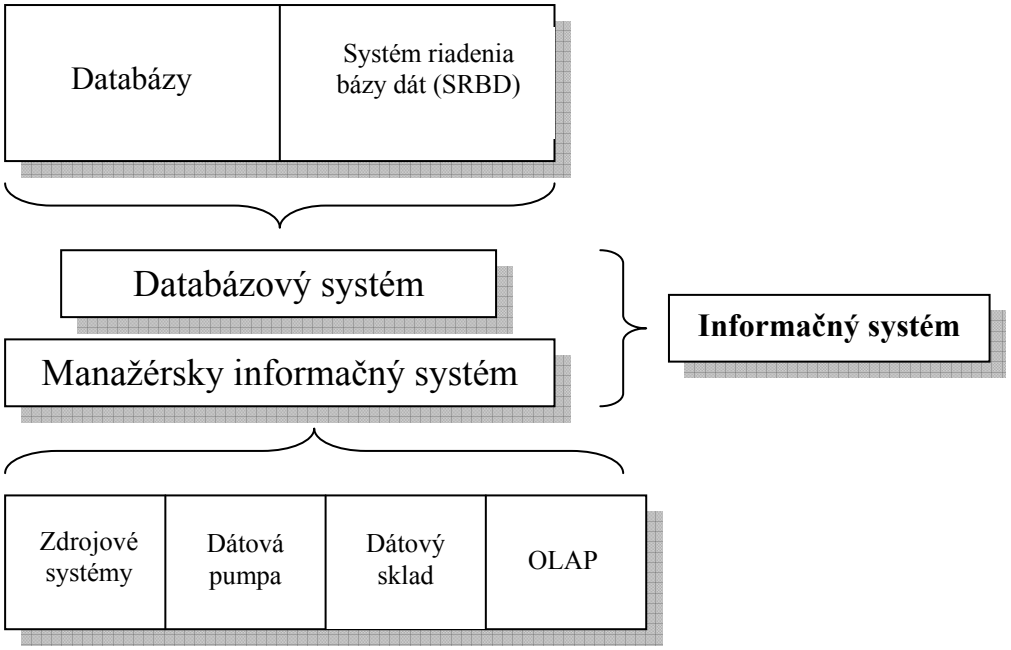
Používateľov DBS môžeme potom rozdeliť do niekoľkých kategórií:

- **správcovia** – majú na starosti vytvorenie organizácie údajov, sledovanie prevádzky a správu databáz, ochranu údajov, používajú **definičný jazyk**
- **aplikační programátori** – vytvárajú programy, ktoré majú prístup k údajom z databáz, používajú **manipulačný – hostiteľský jazyk**
- **neprogramátori** – využívajú bázu dát, používajú **špecializované jazyky** alebo programátormi vytvorené **parametrické jazyky.**

## A kam zaradím pojem informačné systémy?

Aký je rozdiel medzi databázovým systémom a informačným systémom (IS)? Ich vzájomný vzťah zobrazuje Obr. 3. IS je spojený s viacerými databázami obsahujúcimi údaje, ktoré nás zaujímajú. Keďže dnes je najdôležitejšie mať čo možno najpresnejšie informácie v čo najkratšom čase, údaje sú uložené práve v databázach, ktoré umožňujú ich jednoduché vyhľadávanie, transformácie na požadovaný tvar, agregácie (spájanie) podľa zvolených kritérií.

Často sa k týmto úlohám pridáva potreba realizovať rôzne analýzy, na základe ktorých dochádza k strategickým rozhodnutiam. Preto sa k databázovému systému pridáva tzv. manažérsky informačný systém, ktorého súčasťou sú zdrojové systémy, dátová pumpa (data pump), dátový sklad (data warehouse) a prostriedky na analýzu (OLAP) a vizualizáciu dát.




Obr 3 Vzťahy medzi pojmami databáza, databázový systém a informačný systém

## Nevyhnutná terminológia

Po krátkom zastavení sa pri objasnení niektorých bežne používaných pojmoch prejdime späť k databázovým systémom.

Najjednoduchším príkladom využitia databázy môže byť už spomenutý telefónny zoznam. Obsahuje údaje o mene, adrese a telefónnom čísle. V prípade, že hľadáme telefónne číslo konkrétnej osoby, nalistujeme ju v abecednom zozname a jednoducho zistíme telefónne číslo.

Horšie je, keď poznáme len adresu niektorého zo svojich obchodných partnerov, a zabudli sme jeho meno (prípadne sme si ho zapísali absolútne nečitateľne). Kým by sme v zozname našli príslušné údaje, potrebovali by sme pozorne prejsť jeho značnú časť. Silu databázového systému oceníme, keď nám v takomto prípade behom niekoľkých sekúnd vyhľadá všetky osoby žijúce na príslušnej adrese a vypíše ich abecedný zoznam.

Ďalším príkladom môže byť knižnica. V každej knižnici sú tri katalógy – knižky zoradené podľa autorov, podľa názvu a systémový podľa oblastí. Zbytočne veľa regálov a zbytočne veľa papiera. Každá kniha je totiž evidovaná trikrát - v každom katalógu raz. A ak sa náhodou stane, že nedisciplinovaný čitateľ poprehadzuje kartotečné lístky, pravdepodobnosť nájsť hľadanú publikáciu poriadne klesne. Navyše, neustála aktualizácia knižnej ponuky si vyžaduje nemalé úsilie niektorého z pracovníkov knižnice.

Ak má knižnica k dispozícii počítač, knihovníčka naplní databázu údajmi (o každej knihe len raz), a to je všetko. Počítač (resp. databázový systém, ktorý je skrytý pod prehľadným používateľským prostredím) údaje usporiada podľa želania, vyhľadá požadované knihy, prípadne knihy obsahujúce zvolené kľúčové slová a konečný výstup môže i vytlačiť. Ak sa knižnica nebráni využitiu počítača úplne, eviduje v ňom aj čitateľov, výpožičky a upomienky. V knižnici, ktorú má na starosti počítač, nemusí knihovníčka každé ráno prezerať zoznam výpožičiek a hľadať „darebákov“, ktorým treba poslať upomienku.

Ak má používaný systém zabudovanú čo i len trošku inteligencie, na požiadanie vyhľadá čitateľov, ktorí k aktuálnemu dátumu mali knihy vrátiť, vytlačí im upomienky a jedinou starosťou knihovníčky je zaniesť ich na poštu. Môžeme predpokladať, že vo veľmi blízkej budúcnosti bude už upomienky posielať po sieti samotný počítač.

Čím väčší podnik, inštitúcia alebo obchod, tým väčšiu databázu potrebuje. V obchodných domoch a veľkoskladoch už prišli na to, že bez počítača sa možno zaobísť len veľmi ťažko. V oblastiach, kde je

nutné rýchlo a pružne reagovať na požiadavky zákazníka, človek nestíha. Nie je možné, aby si v hlave udržal informácie o cenách, kvalite a množstve tovaru. Vďaka počítaču netreba - má k dispozícii databázu (ktorá potrebné údaje obsahuje) a databázový systém, ktorý s nimi dokáže pracovať

Vo všetkých uvedených príkladoch sú stredobodom nášho záujmu údaje. Aké charakteristiky by mali mať údaje uložené v databáze? Možno uviesť napríklad tieto:

- **perzistencia** – pretrvávanie dát nezávisle na programoch (optický disk),
- **spoľahlivosť** – charakterizovaná pojmami integrita a bezpečnosť,
- **zdieľanie** údajov viacerými používateľmi súčasne,
- **neredundancia** – odstránenie nadbytočnosti údajov. Našou snahou je, aby sa v databáze údaje zbytočne neopakovali,
- **nezávislosť** – programy pristupujúce k údajom sú nezávislé na tom, kde a ako sú údaje uložené.

V súvislosti s údajmi musíme dávať pozor na to, že je diametrálny rozdiel medzi pojmami **neexistujúca a neznáma hodnota**. Ak niektorú hodnotu nepoznáme, nemôžeme jednoznačne povedať, že určite neexistuje, v databázový systém na to používa hodnotu **NULL**. Dôležité je zapamätať si, že hodnota NULL:

- nie je 0 (nula),
- , ‘, , ‘ ani „ „ alebo „ “,
- ‚‘ alebo „“ (reťazec nulovej dĺžky),
- Hodnota NULL ovplyvňuje matematické operácie, preto (21 * NULL) + 32 = NULL, (3 * 23) + NULL = NULL, a dokonca aj [jednotkova_cena] * [pocet_kusov] = NULL, ak jedno z polí obsahuje NULL.

Hodnotou NULL sa budeme zaoberať podrobnejšie v kapitole o relačnej algebre, ktorá „stojí“ v pozadí práce s dotazmi.

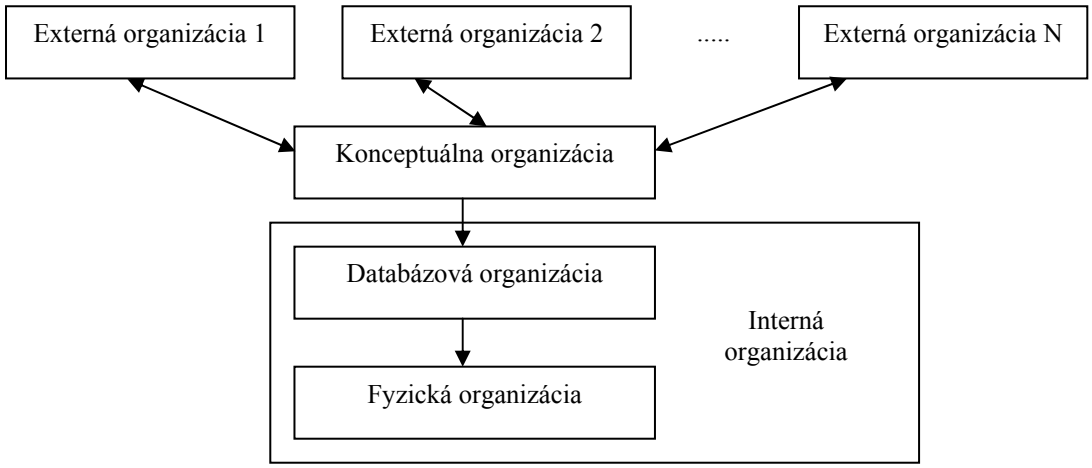
Riadením údajov sa zaoberá **databázová technológia**. Pod týmto pojmom rozumieme unifikovaný (jednotný) súbor pojmov, prostriedkov a techník slúžiacich na vytváranie informačných systémov.

Často sa databázová technológia popisuje ako hierarchické usporiadanie abstrakcií. Tento prístup je definovaný štandardom ANSI/SPARC a používa sa v súvislosti s databázovými a informačnými systémami od 70. rokov minulého storočia. Uvedený štandard rozlišuje tri úrovne, vrstvy organizácií údajov (Obr. 4):

- Externé organizácie
- Konceptuálna organizácia
- Interná organizácia

Niekedy sa k týmto trom základným vrstvám pridáva štvrtá – fyzická organizácia. Na Obr. 4 je zobrazená trojúrovňová architektúra, ktorá bola zavedená skupinou ANSI/SPARC a je štandardom pre modely databázových systémov.

Rovnako ako každá aplikácia je aj každý databázový systém naprogramovaný v niektorom z programovacích jazykov. Zároveň každý DBS pracuje v operačnom systéme, ktorý mu ponúka funkcie pre prácu so súbormi na najnižšej úrovni. Na základe týchto úvah možno rozdeliť organizáciu údajov na internú a externú.




Obr. 4 ANSI/SPARC model trojúrovňovej architektúry väčšiny DBS

Údaje ako také sú uložené v báze údajov, pre ktorú existuje jediná – **interná organizácia**, spoločná pre všetky oblasti využitia pracujúce nad databázovým systémom. Ide o vnútorné uloženie údajov v súbore alebo súboroch (niektoré DBS ukladajú všetko do jedného súboru, iné používajú viacero súborov), z ktorých je možné získavať údaje určitým, niekedy jednoduchým inokedy komplikovanejším, spôsobom).

Vezmime si údaje ukladané do súboru v niektorom programovacom jazyku. Pokiaľ sme ukladali údaje do súboru obsahujúceho údaje napríklad typu record, bolo možné k údajom pristupovať z iného programu len po definovaní rovnakého recordu. Museli byť zladené jednotlivé bajty v dátovom type. Táto požiadavka v databázových systémoch odpadá.

Nad internou organizáciou môže existovať viac **externých organizácií**. Externá organizácia už poskytuje možnosti manipulácie s údajmi nie ako s bajtami, ale ako so záznamami.

Pre porovnanie uveďme, že jedna externá organizácia môže byť navrhnutá tak, že v rámci recordu bude mať meno, priezvisko ako dve položky napr. po 20 znakov, v inom bude definovať meno + priezvisko ako 1x40 znakov – sú to dve rôzne externé organizácie údajov nad jednou bázou údajov.

Prostredníctvom externej organizácie môže príslušný používateľ údaje na základe poskytnutých práv a priorít vidieť resp. môže s nimi pracovať. Spôsob čítania týchto údajov je zakomponovaný v obslužnom systéme, v systéme riadenia bázy dát (SRBD), ktorý si popíšeme neskôr.

Báza údajov je spoločná pre všetky aplikačné oblasti, interná organizácia využíva nástroje operačného systému – I/O operácie, zápis a čítanie do súboru. Prostredníctvom nej sa vytvárajú aplikácie na prácu s externými organizáciami údajov (konkrétnymi typmi súborov alebo výberom podľa práv).

Z hľadiska návrhu a vytvorenia celkového používateľského pohľadu na bázu údajov sa postupne presadila zaujímavá myšlienka – vložiť medzi internú a externú organizáciu údajov ešte jednu - **konceptuálnu organizáciu dát**. Ide o pohľad na údaje v používateľsky prijateľnej forme – napr. organizácia polí a typov v rámci tabuľky. Je implementovaná nad internou organizáciou a operácie v nej sú použité pre implementáciu funkcií externých organizácií.

Na čo táto organizácia údajov slúži? Je podstatne jednoduchšie odstrániť stĺpec z tabuľky tak, že povieme „stĺpec preč“. Pokiaľ by takáto funkcia v rámci externej organizácie neexistovala, museli by sme použiť cyklus odstraňujúci položku stĺpca z každého záznamu osobitne, teda prehľadávať a mazať súbor ručne. Takúto funkciu nám našťastie poskytuje konceptuálna organizácia údajov.

Hlavnou prednosťou DBS je, že nie je nutné pre každú novú aplikáciu vytvárať celé prostredie a navrhovať štruktúru súborov od piky. Ide o prostriedok na spracovanie veľkej triedy problémov.
Prakticky všetky činnosti sa v DBS realizujú pomocou **definičného jazyka** (či už ako príkazy menu alebo príkazy SQL jazyka). Môžeme to chápať tak, že na základe vytvorenej internej organizácie navrhujeme konceptuálnu a externé organizácie údajov.

Výhodou je, že údaje a základné „príkazy“ (ide skôr o nastavenia, popis databázy) definičného jazyka je možné používať aj vo viacerých, navzájom odlišných, DBS. Tieto základné príkazy sú uložené v katalógoch (informačných schémach) databázových systémov (zvyčajne sú to návrhy, resp.

definície databázových objektov), ktorých spoločný obsah tvorí prepojenie medzi jednotlivými systémami.

Na tomto princípe sú založené **univerzálne databázové systémy**. Funkcie takýchto DBS zahrňujú:

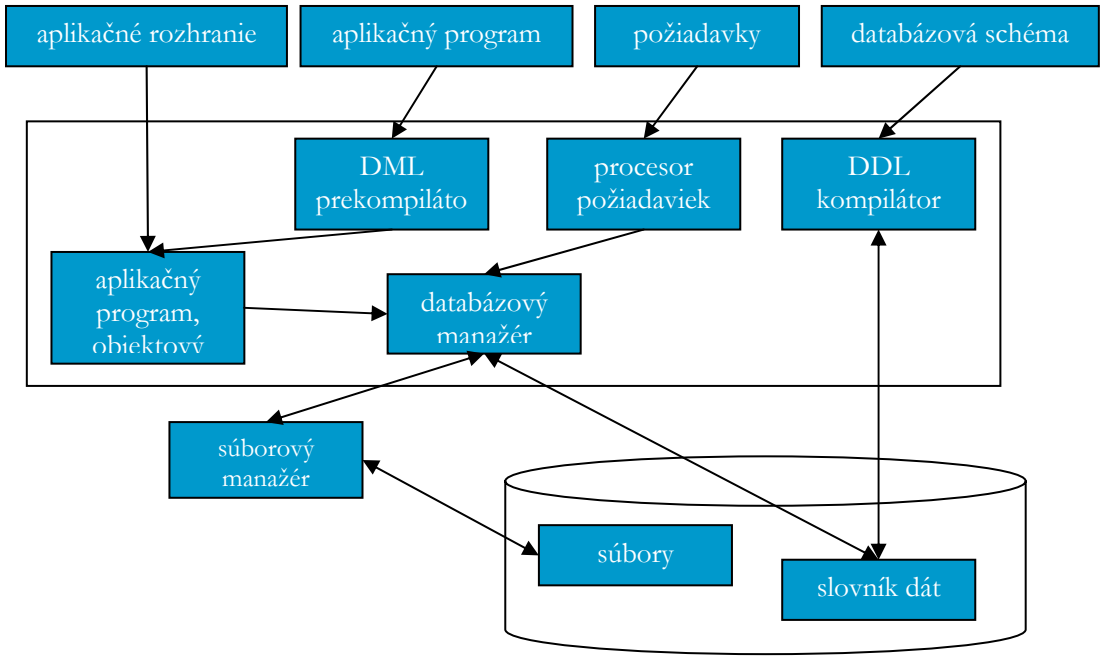
- definovanie bázy dát,
- vytváranie bázy dát (napĺňanie),
- aktualizáciu bázy dát (mazanie, prepočty, pridávanie,...),
- výber bázy dát (selekcia, filtrovanie, nastavenie práv,...).

Okrem univerzálnych databázových systémov existujú ešte .federatívne, heterogénne, homogénne. Podrobnejšie sa však s nimi nebudeme zaoberať.

Zo základných prvkov sa odvíja vytváranie konkrétnych databázových aplikácií vo vývojových prostrediach, používajúcich vyšší programovací jazyk, informačnú schému a bázu dát.

Okrem univerzálnych DBS existujú aj **špecializované**. Používajú sa na miestach, kde aplikovanie univerzálnych DBS nie je vhodné, buď kvôli rýchlosti, rozsahu alebo iným obmedzeniam – Palm Pilot, banky, armáda.

Prostredníctvom DBS možno dosiahnuť podstatne lepšiu účinnosť implementačných programov ako môžu vytvoriť bežní programátori v nižších programovacích jazykoch s limitovanými skúsenosťami a časovo-kapacitnými možnosťami programátora. Na nasledujúcom Obr. 5 vidíme klasickú architektúru databázového systému. Nemusíte sa obávať toho, že väčšina pojmov je pre vás zatiaľ neznáma. Po preštudovaní tohto študijného materiálu budete väčšinu pojmov vedieť popísať a vysvetliť.




Obr. 5 Architektúra databázového systému

Naším cieľom je naučiť sa vytvárať databázy, pracovať s databázovým systémom a uvedomiť si základné súvislosti. Čiastočne sa budeme snažiť prejsť celý **životný cyklus tvorby databázy**, ktorý pozostáva približne z nasledujúcich krokov:

1. Navrhneme vhodné organizácie údajov (konceptuálna, interná, externé), ktoré vyplynú z analýzy funkcií systému.

2. Ak pracujeme s už definovaným DBS orientujeme návrh a analýzu štruktúry databázy tak, aby sme využili výhody a nástroje konkrétneho DBS
3. Do databázy vložíme alebo vygenerujeme testovacie údaje.
4. Vytvoríme aplikačné programy a používateľské rozhrania pre neprogramátorov (vkladanie údajov, aktualizácia, výpočty, výstupy).
5. Naplníme DBS reálnymi údajmi.
6. Otestujeme funkčnosť, rýchlosť a v prípade potreby aj správnosť výpočtových výsledkov.
7. Pripravíme servisné programy a doladíme vzhľad

Výsledná práca s databázovým systémom už nie je zložitá, ak máme dostatočne široké teoretické poznatky. Bežné operácie, ktoré budeme v databázovom systéme realizovať, možno zhrnúť do niekoľkých oblastí:

- vkladanie údajov a prezeranie databázy,
- úpravy (napr. mazanie, prepisovanie),
- triedenie (usporiadanie podľa abecedy, podľa číselných hodnôt a pod.),
- výber, prehľadávanie (napr. údajov spĺňajúcich zadanú podmienku),
- realizácia matematických a štatistických operácií (súčty, priemery, ...),
- výstupy (napr. na tlačiareň, na obrazovku),
- procedúry, spúšte, makrá (automatické vykonanie viacerých, najčastejšie opakujúcich sa operácií).

Pri práci s databázami nezáleží na tom, s akým systémom pracujeme, ide len o to, aby sme mali prístup k informáciám. Najdôležitejšie sú pre nás údaje, nie spôsob (program - databázový systém), akým sa k nim dostaneme.

Existuje mnoho databázových systémov, ktoré sú schopné vykonávať všetky tieto operácie. Môžeme ich hrubo (bez podrobnejšieho vysvetľovania) rozdeliť na desktopové DBS a DBS typu client/server (budeme ich označovať C/S). Medzi najznámejšie a najpoužívanejšie (nielen u nás) desktopové DBS patria dBase, FoxPro, Paradox a alebo MS Access. MS Access má špecifické postavenie v tom, že v skutočnosti využíva služby MS Jet, ktorý skôr môžeme pomenovať pojmom SRBD.

Pri práci s obrovským množstvom údajov (rádovo GB a TB) sa používajú robustné databázové systémy postavené na architektúre C/S, ako sú napríklad DB2, Oracle, MS SQL, Firebird a pod. Webové aplikácie sa často spoliehajú na služby databázových systémov MySQL alebo PosgreSQL.

V bežnej praxi sa však nestretneme s databázovým systémom v jeho pôvodnej - surovej podobe. V každom zo spomínaných systémov totiž možno vytvárať aj aplikácie - programy, ktoré sú už prispôsobené konkrétnym oblastiam použitia. Uľahčujú a zrýchľujú prácu, dávajú k dispozícii len operácie, ktoré tá ktorá oblasť využitia potrebuje.

Po rozhliadnutí sa vôkol seba zistíme, že databázy majú veľmi široké využitie. Úlohou databáz je modelovať určitú časť reálneho sveta, ktorá sa nazýva **priestor problému**. Báza údajov musí obsahovať také údaje o objekte (**realite**), aby v nej boli zachytené všetky skutočnosti potrebné pre používateľov databázy. Pritom údaje musia odrážať skutočnosti verne a v časovom súlade s **vývojom reality**.

Tomu musíme pri návrhu databázy venovať značnú pozornosť – nie je vhodné používať napr. položku vek, pretože táto sa neustále mení, a pokiaľ nemáme zabezpečenú automatickú alebo nebodaj ručnú aktualizáciu údajov, hrozí nebezpečenstvo straty správnosti údajov.

Priestor problému je zo svojej podstaty komplikovaný a chaotický, preto sa na jeho popis používa model, v ktorom sa definuje presne stanovená množina objektov a ich vzájomných vzťahov. Čo to vlastne znamená? Predstavme si takúto situáciu:

Vráťme sa k už spomínanej knižnici. Váš známy, ktorý tam pracuje, už ďalej nestíha neustále upravovať papierové zoznamy autorov, knižných titulov a upomienok, a tiež by chcel mať vytvorenú aplikáciu, ktorá by toto všetko uľahčovala.

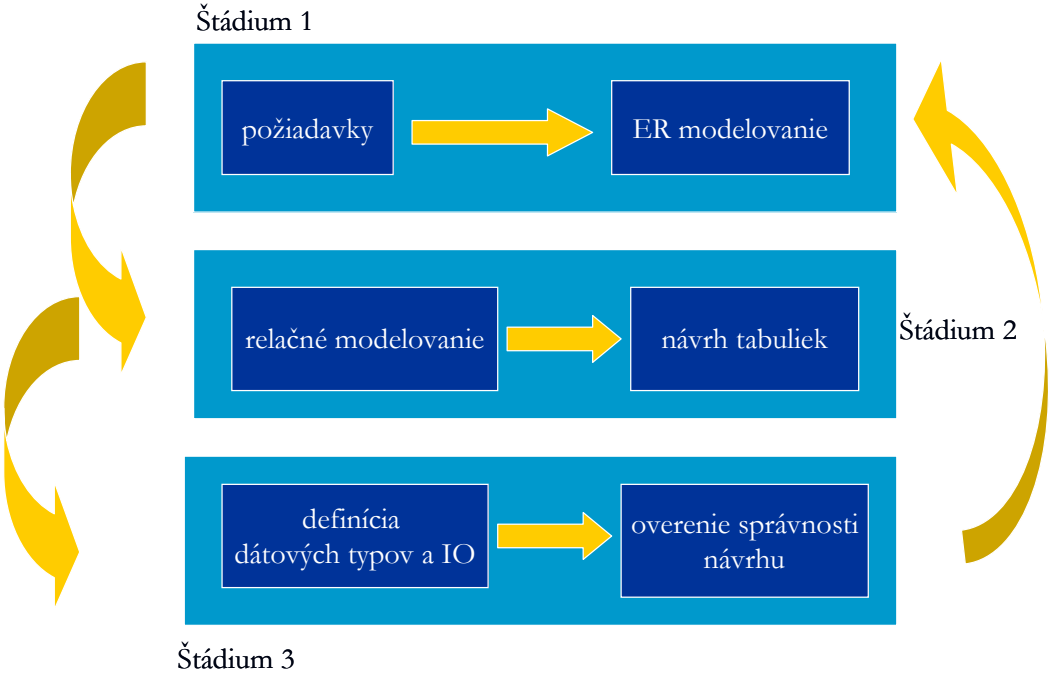
Keďže ste informatici, osloví vás s ponukou na spoluprácu. Vaša úloha sa na prvý pohľad zdá Sizyfovská – ako možno „preniesť“ police plné kníh, katalógy autorov a skriňu plnú upomienok, ktoré vlastne tvoria priestor problému, do počítača tak, aby sme ? Pri dodržaní určitého overeného postupu, to možno dosiahnuť. Tento postup môžeme nazvať **návrh databázy**.

## Prečo sa vôbec zaoberať návrhom databázy?

Možno ste si tiež položili otázku v nadpise podkapitoly. Človek (študent) je tvor lenivý a rád by chcel mať všetky veci hneď hotové. Dostane nápad, a hneď ho ide zrealizovať, bez toho, aby sa na chvíľu zastavil a porozmýšľal (samozrejme, tak ako všade, aj tu existujú výnimky).

Položme si však otázku: Začali by ste stavať svoj vysnívaný dom bez toho, aby ste mali presný plán, spočítané náklady a vybavené povolenia? Asi nie. A rovnako je to v návrhu databáz. Chyby, ktoré by sme spravili v návrhu databáz, môžu mať rovnako katastrofálne následky ako chyby pri stavbe domu. Okrem toho, mnoho databáz môže mať v konečnom dôsledku oveľa vyššiu cenu ako dom. Chyba v stavbe domu sa bude týkať vás, zatiaľ čo chyba v návrhu databázy ohrozí mnoho ľudí, priamych i nepriamych používateľov. Preto musíme byť pri návrhu databázy nanajvýš obozretní. Existuje jeden trefný výrok dlhoročného návrhára databáz, ktorý povedal:

*Nikdy nie je dosť času navrhnúť databázu správne, ale vždy sa nájde dostatok času na jej kompletné prerobenie.*




Obr. 6 Kroky v návrhu databázy a ich vzťah

Čo chceme dosiahnuť návrhom? Chceme, aby sa naša výsledná databáza správala podľa týchto zásad:

- Databáza bude podporovať plánované i neplánované získavanie informácií.
- Databáza bude obsahovať správne a efektívne navrhnuté tabuľky.
- Integrita údajov bude zabezpečená na úrovni tabuliek a vzťahov.
- Databáza bude podporovať tzv. Bussines rules.
- Databázu bude možno udržovať a bezproblémovo rozširovať.
- Údaje bude možné pohodlne upravovať.
- Redundancia (nadbytočnosť) údajov bude na minime.
- Údaje bude možné rýchlo a pohodlne získavať, aj keď sa bude jednať o ad-hoc (náhodné) požiadavky.
- Docielime oddelenie aplikačného rozhrania od databázového.

Vráťme sa k nášmu príkladu s knižnicou a začnime realizovať práce na jej návrhu. Kým vytvoríme plnohodnotnú databázu a nad ňou databázovú aplikáciu, musíme prejsť všetkými štádiami, ktoré sú zobrazené na Obr. 6.

Prvým krokom v návrhu je dlhý rozhovor s vašim známym, v ktorom sa vám svojimi slovami pokúsi popísať, čo všetko eviduje, kde to má uložené, ako sa v tom vyzná, aké tlačivá používa, čo všetko tlačí a pod. O vedení týchto rozhovorov existuje mnoho publikácií a nemožno ich v žiadnom prípade podceňovať. Uvedomte si, že ste v úlohe tlmočníka, ktorý má preložiť reč bežného smrteľníka do reči programátora, pričom nesmie stratiť pointu. Vašou úlohou v tomto kroku je pochopiť jednotlivé procesy a úkony, s ktorými sa knihovník každodenne stretáva, aby ste boli schopní preniesť uvedené postupy do aplikácie. Nesmie dôjsť k žiadnej strate potrebných údajov.

Na základe poznámok z tohto rozhovoru by však počítač (alebo skôr programátor) nevedel, čo má spraviť. Musíme sa preto sústrediť na to, aby sme mu vedeli popísanú realitu nejakým štandardizovaným spôsobom namodelovať. Preto ďalším krokom v tvorbe aplikácie je vytvorenie dátového modelu.

## Dátové modely

Pojem **dátový model** možno definovať ako súbor konceptuálnych prostriedkov pre popis údajov, vzťahov medzi údajmi a ich sémantiky. Systémy riadenia bázy dát (SRBD) ukladajú údaje tak, aby k nim bol možný prístup a manipulácia pomocou aplikačných programov. Štruktúra údajov a techniky ich sprístupnenia poskytované rôznymi SRBD sa označujú pojmom dátový model. Pod pojmom model preto nebudeme rozumieť návrh konkrétnej databázy, ale popis, ktorý hovorí, aké postupy pre spracovanie údajov máme v danej databázovej platforme k dispozícii, akým spôsobom sa na údaje pozeráme a aké pravidlá by mali uložené údaje spĺňať.

Dátové modelovanie vzniklo ako odpoveď na základnú požiadavku, ktorá je na databázy kladená – databázy musia byť obrazom skutočného, reálneho sveta. Iba v tomto prípade je z nich možné získať správne informácie. Úlohou modelovania je identifikovať dôležité prvky reality a uložiť ich v databáze.

Dátové modely možno rozdeliť podľa viacerých kritérií. Na základe toho, s akým elementárnym prvkom model pracuje, možno modely rozdeliť do troch skupín:

1. **Logické modely založené na objektoch**, ktoré popisujú údaje na konceptuálnej úrovni. Ich hlavnou výhodou je flexibilita a možnosť priameho definovania obmedzení. Do tejto skupiny patrí viac ako 30 modelov, pričom k najznámejším patrí entitno – relačný model, objektovo – orientovaný model, binárny model alebo sémantický model.

2. **Logické modely založené na záznamoch**, ktoré tiež popisujú údaje na konceptuálnej úrovni. Modely tejto skupiny popisujú logickú štruktúru databázy a priamo nepopisujú ako sú údaje v databázach uložené. Každý dátový typ definuje presný počet polí alebo atribútov. Pole má vo väčšine prípadov fixnú veľkosť. K najznámejším, a zároveň k najrozšírenejším, modelom patria hierarchický, sieťový a relačný model.
3. **Fyzické dátové modely** popisujú údaje na najnižšej úrovni. Používajú sa zriedkavo.

V literatúre možno nájsť informáciu o štyroch generáciách modelov:

1. primitívne dátové modely
2. klasické dátové modely
3. sémantické dátové modely
4. aplikačne orientované dátové modely

### Systémy riadenia súborov (File management system)

Systém riadenia súborov možno zaradiť do prvej generácie modelov. Pred začiatkom používania databázových systémov sa všetky údaje ukladali do súborov. Systém riadenia súborov sledoval názvy a umiestnenie týchto súborov, neposkytoval však žiadne informácie o ich obsahu, preto textový

a binárny súbor boli z pohľadu systému identické. Informácie o obsahu súborov bolo možné získať až na úrovni aplikácií.

Najjednoduchšou formou databázových systémov je súbor s jedinou tabuľkou, pričom medzi jednotlivými záznamami neexistujú žiadne vzťahy. K záznamom možno pristupovať iba sekvenčne v aplikačnom programe, čo spôsobuje výraznú časovú náročnosť vyhľadávania konkrétnych informácií. Súborový model dát je taktiež charakteristický redundanciou údajov, problémami s integritou a celkovou údržbou.

Do druhej generácie modelov možno zaradiť modely, ktoré sú založené na záznamoch. Vzťahy z reality, ktorá sa má v modeli zachytiť, sa realizujú vzťahmi medzi záznamami. Do tejto generácie modelov sa zaraďujú hierarchické, sieťové a relačné dátové modely. Všetky tri sú súborovo orientované, čo niekedy spôsobuje problémy so zachytením reality priestoru problému.

## Hierarchický model

Typický problém, ktorý bolo potrebné vyriešiť na počítači, bola evidencia jednotlivých komponentov, z ktorých sa skladali rôzne výrobky. Pre tieto komponenty je typické hierarchické usporiadanie, a ich zoznam, ktorý zachytáva ich hierarchickú štruktúru, sa nazýva kusovník. Pre modelovanie kusovníka bol vytvorený hierarchický dátový model.

Hierarchický dátový model je predchodcom relačného modelu, v súčasnosti sa používa iba v špecifických prípadoch. Tento model pozostáva zo série súborov usporiadaných do stromovej štruktúry. Údaje sú reprezentované sériou vzťahov typu **rodič – potomok**. Jeden záznam je zapísaný v podobe stromovej štruktúry zloženej z jednej alebo viacerých skupín polí, ktoré sa nazývajú **segmenty**. Segmenty tvoria jednotlivé uzly stromu. Každý potomok môže byť spojený iba s jediným rodičom a k potomkovi sa možno dostať iba cez jeho rodiča.

Zložitosť modelovaného problému vedie k redundancií údajov, ktorú možno odstrániť tak, že údaje budú fyzicky uložené iba na jedinom mieste. V modeli sa na tieto údaje bude ukazovať pomocou smerníkov, ktoré sú priamo zadefinované v databázovej štruktúre a nemožno ich meniť. Výsledkom je celková malá flexibilita (prispôsobivosť) hierarchického modelu na zmeny v modelovanej realite.

Vyhľadávanie konkrétnych údajov si vyžaduje navigáciu medzi záznamami po jednom zázname v smere hore, dole a do strany. Popis cesty od koreňa až ku konkrétnemu záznamu sa nazýva hierarchická cesta.

Hlavnou výhodou hierarchického modelu je jednoduchá štruktúra, usporiadanie rodič – potomok a výkon. Systémy postavené na tomto modeli sú vhodné pre spracovávanie veľkého množstva transakcií, napríklad overovanie kreditných kariet, spracovanie bankových transakcií ATM a pod. Typickým predstaviteľom hierarchického riadiaceho systému je IMS (Information Management System) od IBM.

## Sieťový model

Na základe hierarchického modelu vznikol sieťový model dát. Sieťový model je založený na súboroch a vzťahoch medzi záznamami súborov, jedná sa preto o súborovo orientovaný model. V tomto modeli možno vytvárať viacnásobné vzťahy medzi segmentmi, čím sa čiastočne zlepšila flexibilita sieťového modelu oproti hierarchickému modelu. Záznamy možno zapojiť do viacerých vzťahov typu rodič – potomok. Tieto vzťahy sa nazývajú sady, na úrovni databázovej schémy **CS–typy** alebo **C-množiny**. Najčastejším typom vzťahu je binárny vzťah, v ktorom vystupujú dva objekty. Sieťový model pracuje iba so vzťahmi typu 1:1 a 1:N. Sieťový model má vlastný štandard, známy ako model CODASYL.

Vyhľadávanie a celá navigácia medzi záznamami sa uskutočňuje záznam po zázname, podobne ako v hierarchickom modeli, ale okrem smeru sa určuje aj relácia, na ktorú sa treba presunúť.

Výhodou sieťových databáz je flexibilita, existencia štandardu a výkon, ktorý sa blíži k výkonu hierarchických SRBD. Najznámejšími sieťovými SRBD sú Adabas alebo IDMS.

## Relačný model

Relačný dátový model vznikol s cieľom zjednodušiť celkovú štruktúru ukladaných údajov. Základným rozdielom oproti predchádzajúcim modelom je skutočnosť, že všetky explicitne

definované štruktúry rodič – potomok boli z modelu odstránené a všetky údaje v databáze sú reprezentované v podobe databázovej relácie –tabuľke podobnej štruktúre.

Možno ste sa už stretli s pojmom relačná databáza, relačný model. Čo vyjadruje prívlastok relačný? Ako prvé by nám mohlo prísť na um, že je to pomenovanie databáz, v ktorých existujú konkrétne vzťahy, múdro povedané relácie, medzi jednotlivými entitami. Skutočnosť je však iná. Môže za to istý pán E. F. Codd, ktorý v roku 1970 úspešne zaviedol základné matematické princípy odvodené z teórie množín a predikátovej logiky do dátového modelovania. Vytvoril tzv. **relačný model** a definoval 12 pravidiel, ktoré by mali relačné SRBD spĺňať. Medzi základné charakteristiky tohto modelu zaraďujeme:

- všetky údaje sa dajú reprezentovať pravidelne usporiadanými štruktúrami, pozostávajúcimi z riadkov a stĺpcov, ktoré nazývame **relácie**. Relácia nemá žiadne vstavané poradie. Jedná sa vlastne o množinu, teda môže byť aj prázdna. Prvky množiny však musia byť jednoznačne identifikované. Z toho potom vyplýva, že tabuľka je reláciou jedine vtedy, ak je každý jej prvok jednoznačne identifikovateľný a tabuľka neobsahuje opakované záznamy
- všetky hodnoty v reláciách sú **skalárne**, t.j. že v konkrétnom priesečníku riadku a stĺpca sa nachádza práve jedna hodnota
- operácie v databáze sa uskutočňujú nad celou reláciou a ich výsledkom je znova celá relácia, tomuto mechanizmu sa hovorí **uzáver**
- riadku v relácii sa hovorí **vektor hodnôt**, počet všetkých riadkov sa označuje pojmom **kardinalita**
- relácia sa skladá z neusporiadanej množiny nula a viac vektorov hodnôt. Poradové číslo riadku nemá žiadny význam
- stĺpec v relácii dostal pomenovanie atribút, a ich počet sa nazýva **stupeň**
- každý atribút má svoje meno (názov stĺpca) a **obor hodnôt**, čiže **doménu**.

Ďalšou charakteristickou črtou relačného dátového modelu je nezávislosť logickej a fyzickej úrovne návrhu modelu. Údaje sa stali nezávislými na implementácii. Zmeny v štruktúre databázy si preto nevyžadujú veľké zmeny v aplikačných programoch.

Relačný model je založené na predikátovom kalkule 1. rádu a relačnej algebre. Na prístup k údajom sa používajú dotazy, ktoré sú založené na poznaní logickej štruktúry vzťahov medzi reláciami a nie na poznaní presnej cesty k fyzicky uloženým údajom. Prístup k údajom je symetrický, čo znamená, že manipulácia s údajmi si nevyžaduje znalosť prístupových mechanizmov k údajom.

Popis reality v relačnom modeli pozostáva z definície entít, ich atribútov a integritných obmedzení. **Entitu** si môžeme predstaviť ako čokoľvek, o čom v systéme potrebujeme uchovávať nejaké informácie. V našom príklade s knižnicou sú entitami Čitateľ, Knihy, Autori, Výpožičky a pod.. Vo firmách sú typickými entitami napríklad Zákazník, Objednávky, Tovar. Každá entita má vlastnosti, ktoré nás zaujímajú z pohľadu funkčnosti systému, nazývame ich **atribúty**, napr. Meno, Adresa sú atribútmi entity Čitateľ, Dátum výpožičky je atribútom entity Výpožička, a pod.

Relačný dátový model taktiež obsahuje popis vzťahov medzi entitami a všetky obmedzenia, platné pre tieto vzťahy. Medzi čitateľom a výpožičkou platí nasledujúci vzťah: Čitateľ môže mať viacero výpožičiek, ale jedna výpožička patrí vždy iba jedinému čitateľovi. Obmedzením môže byť podmienka, že Dátum vypožičania nesmie byť novší ako je aktuálny dátum, dátum vrátenia zase skôr ako Dátum vypožičania.

Existuje názorová nejednotnosť, ktoré z 12 pravidiel Dr. Codda sú potrebné na to, aby bolo možné prehlásiť SRBD za relačný.

## 12 Coddových pravidiel

Doktor E. F. Codd definoval v roku 1970 v článku „A Relational Model of Data for Large Shared Data Banks“ relačný databázový model. Neskôr uverejnil zoznam 12 pravidiel, ktoré musí systém riadenia bázy dát (SRBD = DBMS) spĺňať, aby ho bolo možné pokladať za systém správy relačnej databázy. Pravidlá predstavujú ideálny stav, nie vlastnú definíciu relačnej databázy. Žiaden súčasný relačný systém všetky uvedené pravidlá nespĺňa.

### 1. Pravidlo informácie

Všetky informácie v relačnej databáze sú reprezentované explicitne na logickej úrovni jediným spôsobom, hodnotami v tabuľkách. Tabuľka predstavuje zoskupenie logicky súvisiacich údajov. Každý riadok obsahuje informáciu o jednej položke tabuľky. Stĺpec popisuje jednotlivé charakteristiky každej položky. Hodnota položky sa nachádza v priesečníku riadka a stĺpca, je atomárna, t.j. priesečníkom riadka a stĺpca je jediná hodnota. V rámci tabuľky neexistuje žiadne ďalšie rozdelenie položiek, napríklad hierarchické.

### 2. Pravidlo zaručeného prístupu

Každý údaj v relačnej databáze musí byť logicky prístupný uvedením názvu tabuľky, názvu stĺpca a primárneho kľúča.

### 3. Systematické ošetrenie prázdnych hodnôt

Prázdne hodnoty NULL sú systematicky plne podporované relačným databázovým systémom pre reprezentáciu chýbajúcich a neplatných údajov nezávisle od dátového typu. Prázdne hodnoty predstavujú neznáme hodnoty, nie neexistujúce. Neznáme hodnoty nie sú totožné s hodnotami prázdny reťazec „“ alebo nula a ovplyvňujú výsledky aritmetických a logických operácií.

### 4. Dynamický online katalóg založený na relačnom modeli

Popis databázy sa na logickej úrovni reprezentuje rovnako ako ostatné údaje. Používateľ môže získavať tieto údaje pomocou rovnakého dotazovacieho jazyka, ako požíva pri kladení požiadaviek v tomto jazyku na normálne údaje. Štvrté pravidlo hovorí o nutnosti existencie metadát, teda každá databáza musí obsahovať systémové tabuľky, ktorých stĺpce popisujú štruktúru samotnej databázy. Kolekcia systémových tabuliek sa nazýva systémový katalóg alebo slovník dát.

### 5. Pravidlo komplexného dátového podjazyka

V relačnom systéme môže existovať podpora pre viacero jazykov, ale zároveň musí existovať aspoň jeden jazyk s definovanou syntaxou, pomocou ktorého možno definovať údaje, definovať pohľady, manipulovať s údajmi, riešiť otázky zachovania integrity, zabezpečovať autorizáciu a transakcie. Možno povedať, že musí existovať aspoň jeden jazyk na komunikáciu so SRBD. Štandardom relačných databázových jazykov sa stal jazyk SQL. SQL je deklaratívny a neprocedurálny jazyk.

### 6. Pravidlo aktualizácie pohľadu

Relačná databáza nemôže byť limitovaná v zobrazovaní údajov používateľovi, nemôže poskytovať pohľady iba na zdrojové tabuľky, ale musí vedieť vytvoriť aj virtuálne tabuľky – pohľady. Pohľady umožňujú zobraziť údaje z jednej alebo viacerých tabuliek, pri ich vytvorení však nedochádza k duplicite uložených údajov. S pohľadmi možno pracovať rovnako ako so zdrojovými tabuľkami. Všetky pohľady, ktoré je možné aktualizovať, je možné aktualizovať aj systémovo. Pohľady umožňujú rôznym používateľom rôzny pohľad na uložené údaje.

### 7. Vysokoúrovňové vkladanie, aktualizácie a odstraňovanie

Predstavuje schopnosť SRBD spracovávať základné a odvodené tabuľky (relácie) ako jednu množinu, v ktorej sa uskutočňuje vyhľadávanie, vkladanie, aktualizácia a odstraňovanie údajov. Riadky sa spracúvajú ako množiny operácií vloženia, aktualizácie a odstránenia. Siedme pravidlo zabezpečuje, že relačné databázy musia podporovať množinové operácie (prienik, rozdiel, zjednotenie, delenie) a základné operácie relačnej algebry (projekcia, reštrikcia, spájanie).

### 8. Fyzická údajová nezávislosť

Aplikačné programy a terminály nie sú logicky ovplyvnené zmenami v spôsobe uloženia údajov alebo zmenami prístupových metód.

### 9. Logická údajová nezávislosť

Aplikačné programy a terminály nie sú logicky ovplyvnené, ak nastanú zmeny v uchovávaní údajov v základných tabuľkách a vzťahoch medzi nimi.

### 10. Nezávislosť integrity

Integritné obmedzenia sa musia dať definovať pomocou relačného dátového jazyka a musia sa dať uložiť v údajovom katalógu. Integritné obmedzenia musia byť súčasťou SRBD, nie aplikačných programov.

### 11. Distribučná nezávislosť

Relačný databázový riadiaci systém má distribučnú nezávislosť. Databázový jazyk musí byť schopný manipulovať s distribuovanými údajmi na iných počítačových systémoch, keďže údaje môžu byť uložené centrálne alebo distribuovane. Používateľ musí mať možnosť získavať údaje z tabuliek uložených na viacerých serveroch pomocou distribuovaných dotazov, ako aj z iných relačných SRBD pomocou heterogénnych dotazov pri zachovaní integritných obmedzení.

### 12. Pravidlo nenarušenia

Ak je súčasťou relačného systému nizkoúrovňový jazyk, nesmie tento jazyk porušovať pravidlá integrity, ktoré sú definované pomocou vysokoúrovňového relačného jazyka.

Relačným modelom sa budeme zaoberať podrobnejšie v ďalšom texte, pre úplnosť si však najskôr uvedieme základné charakteristiky objektovo – orientovaného modelu.

### Objektovo – orientovaný model

Na najnižšej úrovni si možno údaje uložené v databáze predstaviť ako sekvenciu bitov, či už je reprezentovaný číselným alebo textovým dátovým typom. Tok (sekvencia, postupnosť) týchto bitov je atomický, nemožno ho deliť na menšie časti. Relačné SRBD sú orientované práve na krátke bitové sekvencie. Jednou z limitácií relačného dátového modelu je preto práca s veľkými binárnymi objektmi (BLOB), ako sú obrázky, multimediálne súbory, dokumenty a i..

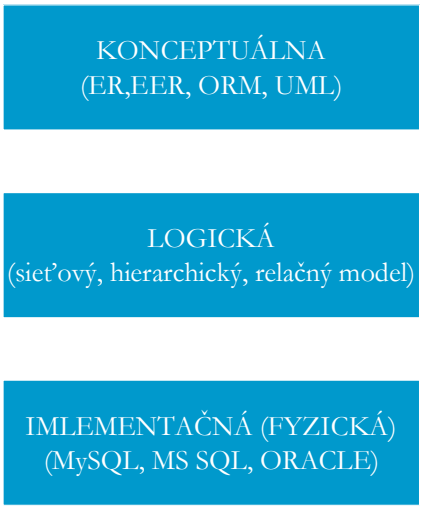
Potreba práce aj s takýmito objektmi viedla k vzniku objektovo – orientovaného modelu dát, ktorý má niekoľko podobných vlastností ako hierarchický model.

V relačnom modeli možno síce pracovať s údajmi typu BLOB, ale tieto sú uložené mimo databázy a SRBD k nim pristupuje pomocou smerníkov. V údajoch typu BLOB možno aj vyhľadávať, ale ostatná manipulácia s týmito údajmi je možná iba klasickými I/O metódami.

Na druhej strane, objektovo – orientovaný dátový model natívne podporuje rozsiahle binárne objekty. V tomto modeli je podobne ako v OO jazykoch všetko reprezentované ako objekt. V modeli sa zavádzajú pojmy ako trieda, inštancia alebo metóda, ktoré nahrádzajú klasické dátové typy relačného modelu. Nedostatkom objektovo – orientovaného modelu je absencia jednotného štandardu – pravidiel, ktoré by definovali podmienky podobne ako Coddove pravidlá v relačnom dátovom modeli.

Treťou generáciou dátových modelov sú sémantické (konceptuálne) modely, ktoré presnejšie vyjadrujú vzťahy v modelovanom priestore problému – v reálnom svete. Hlavným predstaviteľom tejto generácie zostáva entitno – relačný model.

Ako poslednú generáciu možno nazvať modely, ktoré sú špecializované na spracovanie určitého typu údajov, napríklad výsledkov experimentálnych meraní, grafiky a dokumentov.




Obr. 7 Tri úrovne návrhu

## Dátové modelovanie

Ako sme už spomenuli, pod pojmom **dátový model** budeme rozumieť myšlienkový, čiže konceptuálny popis priestoru

problému. Dátový model predstavuje najabstraktnejšiu časť celého návrhu databázy.

Účelom modelovania databázy je návrh databázy odrážajúcej skúmanú realitu. Dátový model je akýmsi plánom pre vytvorenie databázy. Aby bol efektívny, musí byť dostatočne jednoduchý, aby ho mohli posudzovať a kontrolovať aj koncoví používatelia, a zároveň musí byť dostatočne podrobný, aby bolo možné na jeho základe vytvoriť fyzickú štruktúru databázy.

Návrh databázy sa vykonáva na troch úrovniach (Obr. 7)

- Konceptuálna úroveň
- Logická úroveň
- Implementačná alebo tiež fyzická úroveň

**Konceptuálna úroveň**

Na konceptuálnej úrovni sa snažíme popísať predmetnú oblasť dátovej základne. Pri návrhu sa neberie do úvahy spôsob neskoršej implementácie, návrh musí byť úplne nezávislý od použitej technologickej platformy. Na konceptuálnej úrovni sa používa niekoľko modelovacích techník, medzi najznámejšie patria: Entitno-relačný model, IDEF1X, ORM (Object Role Modeling) a jazyk UML.

**Logická úroveň**

Model na logickej úrovni ešte stále nesmie byť zaťažený implementačnými špecifikami daného riešenia. Logická úroveň určuje ako je obsah systému v danej technológii realizovaný. Pre modelovanie na logickej úrovni sa používa relačný model, sieťový model, hierarchický model prípadne objektovo orientovaný model alebo objektovo relačný model.

**Implementačná úroveň**

Na tejto úrovni sa vyberá konkrétna databázová platforma, na ktorej bude navrhovaná dátová základňa vytvorená. Využívajú sa tu špecifiká použitého vývojového prostredia a konkrétnej databázovej platformy. Implementačný návrh určuje čím je technologické riešenie realizované.

V našom príklade si teda potrebuje vybrať, aký model použijeme, aby sme dostali všetky potrebné údaje a súvislosti týkajúce sa chodu knižnice do počítača. Keďže v súčasnosti je stále najrozšírenejším modelom relačný model, vyberieme si ho tiež. Skôr ako budeme môcť definovať jednotlivé tabuľky v konkrétnom databázovom systéme, budeme si musieť poznatky získané rozhovormi a sedeniami so zadávateľom našej úlohy transformovať do formy, ktorá je zrozumiteľná, prehľadná, ale zároveň dostatočne podrobná. Pre relačný model je vhodným riešením entitno – relačné modelovanie, ktorého grafickým výstupom sú entitno – relačné diagramy.

## Entitno-relačné modelovanie

**Entitno-relačný model** (grafickou reprezentáciou je ER diagram (ERD)) navrhol v 70-tych rokoch (1976) Peter Chen. Je to vysoko úrovňový dátový model používaný na konceptuálnej úrovni. Reálny život vníma ako entity a vzťahy medzi nimi. Základným komponentom modelu je **Entitno-relačný diagram**, ktorý je používaný pre vizuálnu reprezentáciu dátových objektov. Jeho použitie poskytuje nasledovné výhody:

- veľmi dobre sa mapuje na relačný model. Konštrukcie použité v ER modeli môžu byť jednoducho transformované do relačných tabuliek.
- je jednoduchý a ľahko pochopiteľný. Z tohto dôvodu ho môžu používať jednak databázoví vývojári, ako aj koncoví používatelia.

Entitno-relačný model obsahuje tri základné prvky:

- Entita
- Atribút
- Vzťah

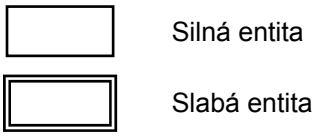
**Entita**

**Entita** je objekt reálneho sveta, ktorý je schopný nezávislej existencie a je jednoznačne odlišný od ostatných objektov. V reálnom živote predstavujú entity veci, o ktorých zhromažďujeme informácie.

Napr. v knižnici sú entitami kniha, čitateľ, pracovník atď. Pre grafickú reprezentáciu sa v diagramoch používa obdĺžnik, do ktorého sa zapisuje názov entity (Obr.8).


Obr. 8 Grafická reprezentácia entít

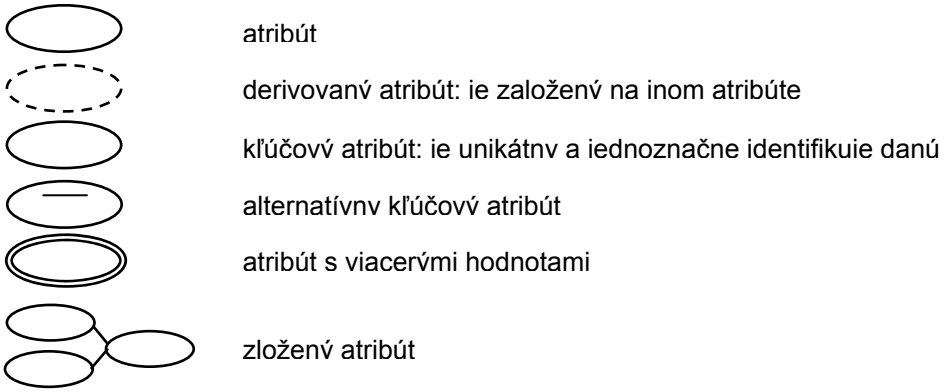
Rozlišujeme tzv. **silné** alebo tiež **regulárne** entity a tzv. **slabé entity**. Silné entity v reálnom živote existujú nezávisle. Slabé entity sú závislé na existencii iných entít, ich kľúčovým atribútom je úplne alebo čiastočne derivovaný kľúčový atribút rodičovskej a v reláciách majú povinnú existenciu vzťahu. Ich grafické znázornenie je na obr. 9.



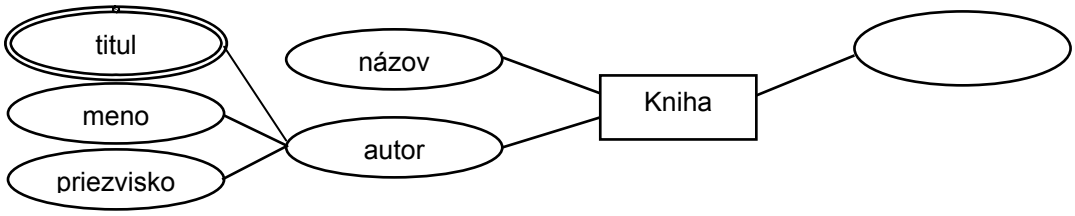

Obr. 9 Grafická reprezentácia silnej a slabej entity

**Atribút**

Atribúty môžeme chápať ako **vlastnosti entity**, pričom jedna entita môže mať niekoľko atribútov. Napríklad kniha má názov, meno autora, počet strán, cenu atď. Relácie tiež môžu mať atribúty, napr. čitateľ si požičal knihu 1.5.2004. Existuje niekoľko základných typov atribútov. Grafická reprezentácia je uvedená na Obr. 10. Na Obr. 11 je príklad atribútov priradených k entite Kniha.




Obr. 10 Grafická reprezentácia atribútov



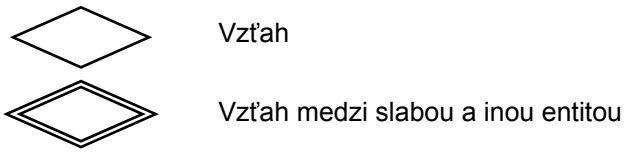

Obr. 11 Príklad vybraných atribútov v entite Kniha

Na úrovni ER modelovania sa nemusíme príliš zaoberať tým, aby sme vytvorili optimálny návrh databázy. Dôležité je, aby sme zachytili všetky potrebné vlastnosti entít a ich vzájomné vzťahy. Ak sa ukáže, že niektorý atribút je príliš zložitý, musíme ho pri transformácii do relačného modelu a v procese normalizácie, rozdeliť. Preto je dôležité, aby sme si v ER diagrame vždy správne zobrazili jednotlivé typy atribútov. Najdôležitejšie typy atribútov sú **primárne, zložené, viaczložkové a derivované**.

**Vzťah, často mylne nazývaný aj relácia**

Pojem relácia predstavuje vzájomné vzťahy medzi dvoma alebo viacerými entitami. Ich grafickú reprezentáciu môžeme vidieť na Obr. 12. Na tomto mieste musíme upozorniť na časté zamieňanie pojmov relácia a vzťah. Musíme vždy vedieť, v ktorej úrovni návrhu sa práve nachádzame, v ERD môžeme označiť reláciou vzťah medzi dvomi entitami, ale napríklad v procese transformácie ERD do relačného modelu sa termínom relácia myslí častejšie tabuľke podobná štruktúra.

Typickým vzťahom v knižnici medzi entitami kniha a čitateľ je vypožičanie knihy. Čitateľ si požičal knihu. Vzťahy by mali byť pomenované, a to plnovýznamovým slovesom, nakoľko medzi dvomi entitami môže byť aj niekoľko vzťahov, napr. čitateľ si požičal knihu, čitateľ daroval knihu. Na Obr. 13 je príklad vzťahu medzi entitami Čitateľ a Kniha.




Obr. 12 Grafická reprezentácia vzťahu


Obr. 13 Príklad vzťahu medzi entitami Čitateľ a Kniha

Relácia môže byť aj **rekurzívna**, t.j. jedna entita môže vo vzťahu vystupovať viackrát pod rôznymi rolami. Napr. zamestnanec riadi zamestnanca.

Ďalším dôležitým pojmom v ER diagramoch je **násobnosť** (**kardinalita**). Jedná sa o počet výskytov objektov obidvoch entít, ktoré sa zúčastňujú v danom vzťahu. Zobrazujeme uvedením hraničných hodnôt nad spojovacou čiarou medzi entitami (Obr. 14).Rozlišujeme nasledovné vzťahy:

**1:1** – vzťah, pri ktorom na obidvoch stranách vystupuje len jeden objekt danej entity. Tento vzťah sa v reálnom živote nevyskytuje veľmi často. Typickým príkladom je vzťah medzi manželmi (samozrejme ak neberieme do úvahy spoločnosť, ktorá toleruje bigamiu). Manžel má len jednu manželku, rovnako ako jedna manželka môže mať súčasne len jedného manžela. Ďalším častým výskytom tohto typu vzťahov sú rezervačné systémy – cestujúci si kupuje miestenku alebo letenku, na jednom mieste v lietadle alebo vo vlaku môže sedieť iba jeden cestujúci.

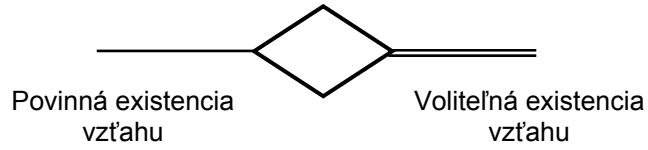
**1:N** – na jednej strane vzťahu je len jeden objekt danej entity, ale na druhej strane môže byť viac objektov danej entity, napr. jednu knihu môže mať súčasne požičanú len jeden čitateľ, ale jeden čitateľ môže mať požičaných naraz niekoľko kníh. Exaktnejším príkladom môže byť trvalé bydlisko. Každý obyvateľ môže mať uvedené iba jedno trvalé bydlisko, zatiaľ čo, pokiaľ nežije ako pustovník na samote, v danom bydlisku žije viacero obyvateľov.

**M:N** – je vzťah, v ktorom na obidvoch stranách vystupuje viac objektov danej entity. Takýto vzťah je pre ďalšiu prácu s návrhom príliš zložitý a je vhodné ho určitým spôsobom upraviť. Nemôžeme ho prostriedkami ER modelovania priamo graficky znázorniť. Rieši sa to vytvorením tzv. **väzobnej entity**, ktorá sa zaradí medzi dve entity. Vzniknú tak väzby 1:N a N:1, čomu hovoríme **dekompozícia vzťahu M:N**.


Obr. 14 Znázornenie násobnosti vzťahu

Pri vzťahoch medzi entitami je ešte potrebné poznať pojem **parcialita**. Určuje povinnosť alebo voliteľnosť existencie každého vzťahu. Pri návrhu sa totiž môžeme stretnúť napríklad s otázkou, či je

nutné, aby bol pracovník priradený do určitého oddelenia alebo môže existovať pracovník, ktorý ešte do žiadneho oddelenia priradený nie je (Obr. 15).




Obr. 15 Znázornenie participácie vo vzťahu

## Rozšírený entitno-relačný model

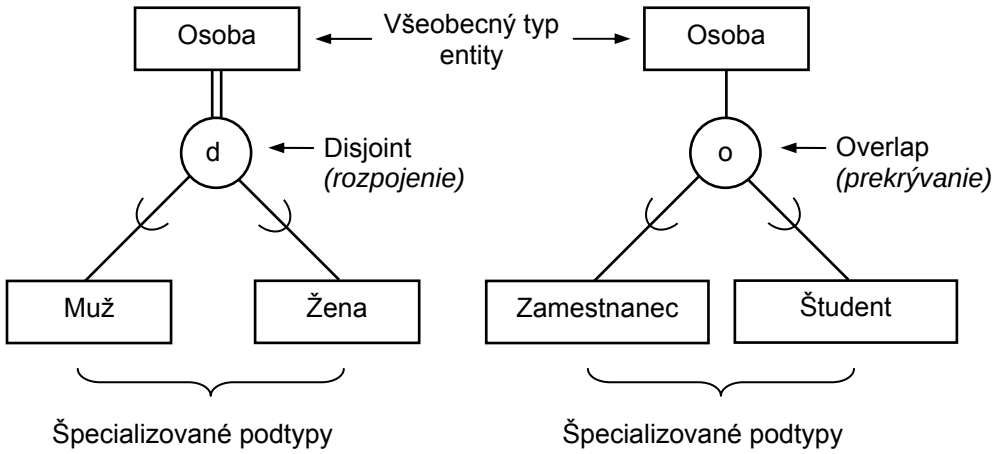
Základný ER model bol uvedený v polovici 70-tych rokov. Od tej doby sa ale firemné pravidlá a uchovávané údaje stali oveľa komplexnejšími a tento model prestával postačovať. V roku 1986 popísali Toby Teorey, Dongqing Yang a James Fry **rozšírený ER model** (Enhanced Entity Relationship EER), ktorého najdôležitejším prínosom je koncept tzv. **supertypov a podtypov**. Tento koncept nám umožňuje modelovať všeobecné entitné typy, tzv. super-typy, a tieto ďalej rozčleňovať do niekoľkých špecializovaných entitných typov, nazývaných podtypy. Každý podtyp dedí atribúty od svojho supertypu, ale môže mať aj vlastné špeciálne atribúty.

Rozšírený entitno-relačný zavádza do pôvodného entitno-relačného modelu nové pojmy: **špecializácia a zovšeobecnenie, agregácia a kompozícia**.

### Špecializácia, zovšeobecnenie

**Špecializácia** je proces definovania jedného alebo viacerých podtypov supertypu. Hľadá sa maximum rozdielov medzi entitami identifikovaním ich osobitných charakteristík.

**Zovšeobecnenie** je proces definovania všeobecného entitného typu zo súboru špecializovanejších entitných typov hľadaním ich spoločných charakteristík. Reprezentuje logický vzťah medzi jednou entitou, označovanou ako **rodičovská entita** a jednou alebo viacerými entitami označovanými ako **dcérske entity**. Každá inštancia (výskyt) dcérskej entity je potom aj inštanciou rodičovskej entity a každá vlastnosť rodičovskej entity je tiež vlastnosťou dcérskej entity. Zovšeobecnenie je **úplné**, ak každá inštancia rodičovskej entity je tiež inštanciou jednej z jej dcérskych entít, v opačnom prípade sa jedná o **čiastočné** zovšeobecnenie. Zovšeobecnenie je **výhradné**, ak každá inštancia rodičovskej entity je inštanciou maximálne jednej dcérskej triedy, v opačnom prípade je **prekrývajúce** sa. Napríklad z entít Muž a Žena zovšeobecnená entita Osoba je úplná (súbor mužov a žien zahŕňa všetky osoby) a jedná sa o výhradné zovšeobecnenie – osoba je muž alebo žena (Obr. 16).

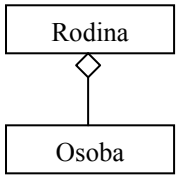



Obr. 16 Zovšeobecnenie

### Agregácia, kompozícia

**Agregácia** reprezentuje vzťah „má“ alebo „je časťou“ medzi entitami, kde jedna reprezentuje celok a druhá časť (Obr. 17).

**Kompozícia** je špecifickou formou agregácie. Reprezentuje situáciu, keď medzi entitami existuje veľmi silný vzťah, t.j. jedna entita môže existovať len spolu s druhou entitou.




Obr. 17 Agregácia

## Stratégie návrhu

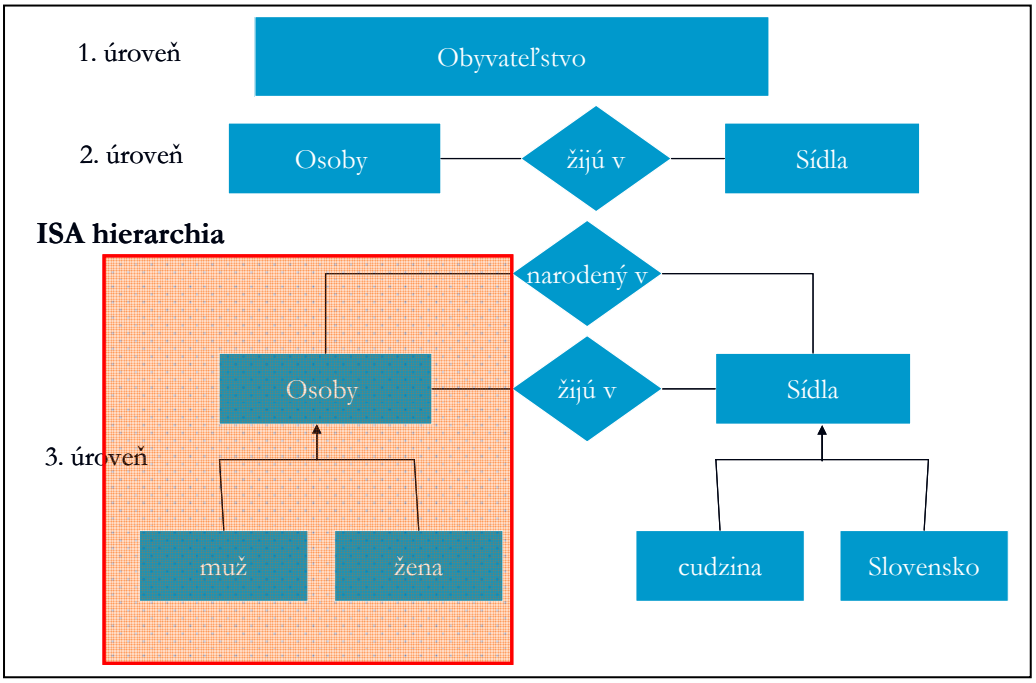
V tomto okamihu máme dostatok prostriedkov na to, aby sme mohli transformovať naše poznámky do ER diagramu. Počas návrhu databázy knižnice sa však môžeme stretnúť s veľmi častým problémom návrhárov, hlavne, ak nemáme dostatočné skúsenosti v návrhu databázy. Čím viac rozmýšľame nad jednotlivými entitami, ktoré chceme v budúcej databáze evidovať, tým viac sa zamotávame do podrobností, alebo na druhej strane, niektoré dôležité veci bagatelizujeme. Problém spočíva v tom, že nevieme, z ktorého konca návrhu začať.

Pre tento prípad použijeme niektorú zo stratégií, ktoré platia vo všeobecnosti a používajú sa napríklad aj v programovaní. S použitím **stratégií návrhu** získame postupom času množstvo skúsenosti a zistíme, že nech navrhujeme databázy riešiace rôznorodé oblasti našej činnosti, existujú v nich veľmi podobné postupy alebo typy entít. Teraz si v krátkosti popíšeme základné stratégie návrhu databáz:

**Zhora dole**

V tejto stratégií postupujeme v smere od všeobecných, globálnych pohľadov k podrobnejším. Postupne delíme popisovaný problém sa podproblémy (Obr. 18), všeobecné entity na konkrétnejšie. Niekedy sa môžeme v tejto súvislosti stretnúť s pojmom **ISA hierarchia**, čo pochádza z anglického „is a“. Typickým príkladom ISA hierarchie je napríklad Osoba – (Muž alebo žena).

Hlavnou nevýhodou tohto postupu je, že tento postup si vyžaduje vysokú mieru abstraktného myslenia, a preto je vhodný pre návrhárov, ktorí už získali dostatok teoretických a empirických skúseností. Musíme si byť vedomí faktu, že chyby v návrhu môžu spôsobiť veľké škody, lebo budeme musieť prepracovať celú časť modelovanej databázy.

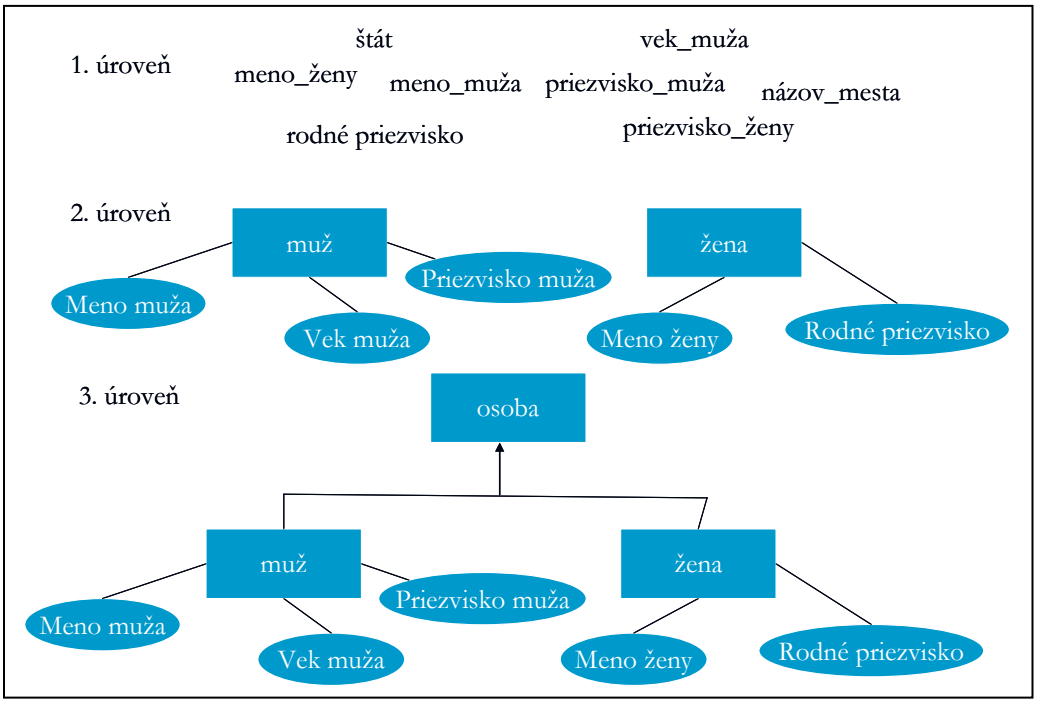



Obr. 18 Stratégia Zhora dole

**Zdola nahor**

V stratégii zdola nahor postupujeme opačným smerom (Obr. 19). Na začiatku si zadefinujeme čo možno najviac atribútov, o ktorých predpokladáme, že ich budeme v databáze evidovať. Potom začneme atribúty zoskupovať do skupín podľa významu, vznikajú čiastkové nezávislé pojmy. Tieto pojmy sa postupne integrujú do vyšších celkov a hierarchií, čím sa nám postupne „vynoria“ všetky potrebné entity.

Výhodou tohto postupu je rýchly návrh databázy, nevýhodou častá reštrukturalizácia jednotlivých entít.




Obr. 19 Stratégia Zdola nahor

**Zvnútra von**

Ďalšou rozšírenou stratégiou je stratégia zvnútra von. V nej si vyberieme základné dôležité pojmy a postupne do návrhu priberáme vzdialenejšie pojmy. Táto stratégia sa podobá stratégií zdola hore, ale v návrhu postupujeme v jednej rovine abstrakcie do šírky, nie ako v stratégii zdola hore v hierarchii.

**Zmiešaná**

Poslednou stratégiou, ktorú si spomenieme, je kombinácia stratégií zhora dole a zdola hore. Najprv si vytvoríme akúsi **skeletálnu schému.** Do nej postupne zabudujeme čiastkové schémy, ktoré podrobne rozpracujeme. V každej čiastkovej schéme postupujeme s pomocou stratégie, ktorá nám pre daný prípad najviac vyhovuje. Nakoniec integrujeme všetky čiastkové schémy do výsledného celku.

Ak použijeme ľubovoľnú z uvedených stratégii, musíme si byť vedomí toho, že nie vždy dostaneme rovnaký návrh. V ďalších krokoch v návrhu databázy je ale zabezpečené, aby sme žiadne dôležité charakteristiky modelovaného priestoru problému nezanedbali.

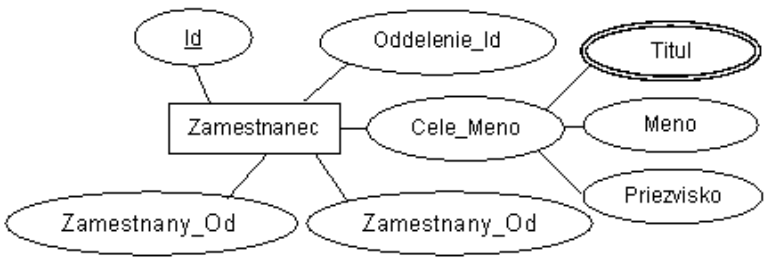
## Transformácia ER diagramov do relačného modelu

Nasledujúcim krokom návrhu relačných modelov databáz je transformácia ERD do relačnej schémy, do relačného modelu. Hoci sa už pomaly približujeme k rozhodnutiu, ktorý databázový systém bude pre nás najvhodnejší, v ktorom implementujeme databázu knižnice, ešte stále sme v konceptuálnej úrovni návrhu. Cieľom nášho návrhu zostáva pritom postaviť model tak, aby bol schopný zodpovedať akúkoľvek otázku, samozrejme týkajúcu sa údajov uložených v databáze a druhým cieľom by malo byť obmedzenie, presnejšie minimalizácia možných redundancií, t.j. zhromažďovania nadbytočných údajov. Od výslednej databázy očakávame, že bude mať schopnosť odpovedať na (skoro) ľubovoľnú otázku. Táto schopnosť závisí od úplnosti a štruktúry dátového modelu. Najdôležitejšie pravidlo znie, že jednotlivé atribúty veľmi ľahko spojím, ale ťažko rozpojím.

Počas **transformácie ERD do relačnej schémy** postupujeme približne nasledovne:

**Entity**

Z jednotlivých entít ERD vytvoríme relácie, tabuľkám podobné štruktúry. Z kľúčových atribútov entít vytvoríme primárny kľúč. Jednotlivé atribúty entity sa stanú atribútmi novej relácie. Každý jej




Obr. 20 Transformácia entít na relácie, vzniká relácia Zamestnanci

| Zamestnanci |
|---|
| **ID** |
| Zamestnany_Od |
| Zamestnany_Do |
| Cele_meno |
| Titul |
| Meno |
| Priezvisko |
| Oddelenie_id |

atribút získa meno z pôvodného atribútu entity (Obr. 20). Týmto spôsobom transformujeme silné, ale aj slabé entity.

**Vzťahy**

Zložitejšie prebieha transformácia u vzťahov. Na základe doteraz získaných poznatkov máme vytvorené jednotlivé relácie. Ďalší krok v tvorbe databázovej aplikácie, ktorá nahradí prácu v miestnej knižnici, je definovanie vzťahov medzi reláciami. Podobne ako pri definícii relácií platí, že základné

princípy sa javia ako úplne prirodzené a vychádzajú z poznania sémantiky, čiže významu dátového modelu.

Entity, medzi ktorými existuje vzťah, sa nazývajú **účastníci vzťahu** a počet účastníkov vzťahu určuje stupeň vzťahu. Najčastejšie sa stretneme s **binárnymi** vzťahmi, menej často s **unárnymi** a **ternálnymi** vzťahmi.

Účasť entity vo vzťahu môže byť úplná alebo čiastočná, pričom často sa stretneme aj s pojmami povinná a voliteľná účasť alebo slabá a regulárna entita. Všetky tieto pojmy vyjadrujú, či daná entita môže existovať aj bez účasti vo vzťahu. Ak máme napríklad entity Čitatelia a Výpožičky, entita Čitatelia má v ich vzťahu čiastočnú účasť, lebo môžeme zapísať čitateľa aj bez toho, aby si hneď musel nejakú knihu vypožičať. Na druhej strane, entita Výpožičky má vo vzťahu úplnú účasť, lebo výpožičku môže uskutočniť jedine čitateľ.

S účasťou vo vzťahu je spojená aj **voliteľnosť** (**parcialita**) vzťahu, t.j. otázka, či sa daná entita musí alebo nemusí daného vzťahu zúčastniť, lepšie povedané voliteľnosť alebo povinnosť existencie vzťahu. Parcialita hovorí o tom, či musí každá kniha patriť do nejakého žánru, či učiteľ musí byť zároveň triednym učiteľom a pod.

Ak zistíme, že medzi uvažovanými entitami existuje určitý vzťah, musíme ho vhodne namodelovať. Všetko, čo k tomu potrebujeme je, že zoberieme príslušné atribúty z jednej relácie, tzv. **primárnej relácie**, a zahrnieme ich do druhej relácie, ktorá sa nazýva **cudzia relácia**. Uvažované atribúty však nemôžeme vybrať náhodne, do úvahy prichádzajú iba tie atribúty, , ktoré jednoznačne identifikujú primárnu entitu, predstavujú jej kandidátny kľúč. Tieto atribúty sa potom v cudzej relácii nazývajú **cudzí kľúč**.

V procese modelovania vzťahov sa často stretneme aj s prípadom, kedy s určitosťou vieme, že medzi dvoma entitami existuje vzťah, ale naviac máme aj nejaké atribúty, ktoré strácajú zmysel, akonáhle sú uvedené ako súčasť jednej zo zúčastnených relácií. V tejto situácii je východiskom vytvorenie abstraktnej relácie, ktorá bude vsunutá medzi pôvodné relácie. Typickým príkladom je modelovanie vzťahu typu M:N (DÁTUM_VYPOŽIČANIA v relácii Výpožičky nemožno priradiť do relácie Knihy), alebo situácia, kedy nás zaujíma história vzťahu.

**Kardinalita**

Maximálny počet inštancií jednej entity, ktoré môžeme asociovať s jednou inštanciou inej entity sa nazýva **kardinalita** vzťahu Existujú tri všeobecné typy kardinality, a síce jedna k jednej (1:1), jedna k viac (1:N) a viac k viac (M:N).

Vzťah typu 1:1 predstavuje vzťah entít X a Y, v ktorom platí, že ľubovoľnú inštanciu entity X je možné asociovať najviac s jednou inštanciou entity Y. Ľudovo povedané jednému záznamu v tabuľke Čitatelia zodpovedá jediný záznam v tabuľke Osobné_údaje_čitateľov. S takýmto typom vzťahu sa stretneme zriedkavejšie, napríklad v prípade, kedy chceme niektoré „citlivé“ údaje oddeliť, aby sa nezobrazovali každému používateľovi. Druhým príkladom je situácia, ak pracujeme s reláciou, ktorá má mnoho atribútov, ale len s niektorými z nich sa pracuje častejšie. Vtedy môžeme atribúty rozdeliť do dvoch relácií, ktoré sú vo vzťahu 1:1, čím docielime zrýchlenie práce s databázou. Ďalším príkladom výskytu vzťahu 1:1 je rezervačný systém leteniek. Vzťahy typu 1:1 sa transformujú tak, že sa primárny kľúč jednej tabuľky stane cudzí kľúčom druhej tabuľky.

Vzťah typu 1:N predstavuje najčastejší typ vzťahu medzi dvoma reláciami. Predstavuje vzťah, kedy jednej inštancií relácie X môžeme priradiť nula, jednu alebo až viacero inštancií relácie Y. Identifikovať primárnu a cudziu reláciu možno v tomto vzťahu tak, že entita na strane jedna je vždy primárna relácia, jej kandidátny kľúč skopírovaní do cudzej relácie na strane viacej sa stane cudzím kľúčom. V prípade 1:N môžeme preniesť iba primárny kľúč z tabuľky na strane 1 do tabuľky na strane N, v ktorej vytvorí cudzí kľúč. V našom príklade bude primárnou reláciou relácia Žánre, jej kandidátny kľúč ID_Žánru sa skopíruje do relácie Knihy, čím sa z neho stane cudzí kľúč.

Vzťah M:N nemôžeme modelovať priamo, ale potrebujeme k tomu spojovaciu, často abstraktnú, reláciu, ktorá má s každým účastníkom vzťahu vzťah typu 1:N. Tá bude pozostávať z primárnych kľúčov oboch entít, prípadne doplnených o atribúty, ktoré charakterizujú vzťah. V tomto type vzťahu môžeme viacerým inštanciám relácie X priradiť viacero inštancií relácie Y.

**Unárne a ternálne vzťahy**

**Unárne** vzťahy majú jediného účastníka, relácia je spojená sama so sebou. Najtypickejším príkladom je vzťah vedúci_oddelenia – zamestnanec, primár – lekár, lebo vedúci oddelenia je tiež zamestnanec a primár je tiež lekárom, pričom často o nich potrebujeme evidovať rovnaké údaje. Vo všeobecnosti platí, že pomocou unárneho vzťahu sa modelujú rôzne hierarchie. Unárne vzťahy sa modelujú rovnako ako binárne, rozdiel je len v tom, že primárna a cudzia reláciu je tá istá relácia. V prípade unárnych vzťahov prichádza do úvahy ľubovoľná kardinalita.

**Ternálne** vzťahy sa podobne ako unárne vzťahy vyskytujú menej často, majú väčšinou nasledujúci tvar: X robí Y pre Z. Nemôžeme ich priamo modelovať a ani neexistuje nejaký všeobecne platný predpis. Ako príklad možno uviesť prípad, kedy máme viacerých dodávateľov, ktorí dodávajú ten istý výrobok, a my chceme vedieť, od ktorého dodávateľa pochádza výrobok, ktorý si kúpil náš zákazník.

V tomto okamihu by sme už mohli nadobudnúť dojem, že sa nám úspešne podarilo zobraziť všetky údaje z poznámok o knižnici do podoby relácií, a chceli by sme ich konečne implementovať v konkrétnom SRBD. Skutočnosť je ale taká, že nás čaká ešte jeden veľmi dôležitý krok, a to **normalizácia databázového návrhu**. Predtým si však popíšme niekoľko ďalších pojmov, nevyhnutných pre nadobudnutie predstavy, ako to s tými databázami v skutočnosti je.

**Redundancia**

Majme napr. tabuľku Objednávky s poliami ID_Objednavky, Meno_predavaca, Oddelenie, Funkcia, Firma, Nazov_vyrobku, Pocet_kusov. Ak chceme zapísať všetky výrobky konkrétnej objednávky, musíme vždy zapísať aj informácie o oddelení a funkcií, ktorú zastáva predavač. Takto navrhnutá tabuľka je zdrojom zbytočných chýb, lebo pri každej novej objednávke, presnejšie výrobku z objednávky, sa môžeme pri zadávaní informácií o oddelení a funkcii predavača ľahko pomýliť. Navyše, údaje o predavačovi budú v systéme zapísané až vtedy, keď sa mu podarí vystaviť aspoň jednu objednávku.

Ešte horší prípad nastane, ak budú tieto nadbytočné informácie uložené vo viacerých tabuľkách, napríklad ak si vytvoríme samostatnú tabuľku Predavači a do nej dáme meno, priezvisko, funkciu a oddelenie, v ktorom pôsobí predavač. Potom nesmieme zabudnúť, že údaje o oddelení a funkcií evidujeme na dvoch miestach, a v prípade ich zmeny (predavač povýši alebo prejde do iného oddelenia), musíme zmenu uskutočniť aj v druhej tabuľke. Tomuto problému sa hovorí **aktualizačná anomália**.

Pozor však, existujú prípady, kedy sú polia v tabuľkách iba naoko redundantné. Typickým prípadom je evidencia jednotkovej ceny. Tento atribút sa môže nachádzať v tabuľke Výrobky a aj v tabuľke Objednávky. Zdalo by sa, že skôr patrí k charakteristike výrobku, a preto ho môžeme z tabuľky Objednávky odstrániť. V tom momente však stratíme informáciu o tom, ako sa cena výrobku v čase menila, lebo pole jednotková cena v tabuľke Výrobky vyjadruje vždy iba jeho aktuálnu cenu, zatiaľ čo v tabuľke Objednávky máme aj informáciu o dátume jej vystavenia, preto vieme zistiť, kedy mal výrobok akú cenu.

**Bezstratová dekompozícia**

Redundanciu údajov môžeme odstrániť, ak rozdelíme pôvodné relácie na ďalšie, pričom ale zabezpečíme, že sa tieto novovytvorené relácie budú dať spätne spojiť a my nestratíme žiadne informácie. Vtedy hovoríme o **bezstratovej dekompozícii**.

Nech R je relácia dekomponovaná na schémy R1,R2... Rk. Nech D je množina funkčných závislostí (vysvetlíme nižšie) medzi atribútmi relácie R. O dekompozícii hovoríme, že je **bezstratová** (vzhľadom na D), ak pre každú populáciu relácie R, ktorá vyhovuje závislostiam D platí, že ju môžeme získať prirodzeným spojením jej projekcií R1..Rk.

Existujú postupy zaručujúce bezstratovosť kompozície, ale nám stačí pravidlo, ktoré hovorí, že ak v relácii R(A,B,C), kde A, B, C sú atribúty relácie R, **platí funkčná závislosť** A->B, môžeme reláciu R bez straty informácie rozložiť na projekcie R1 (A,B) a R2(A,C).

## Normalizácia - alfa a omega databázového návrhu

Ak sa chceme vyhnúť situácii (často márne), kedy máme po úpornej drine vytvorenú skoro hotovú databázovú aplikáciu, mali by sme sa ešte počas prípravnej práce s papierom a perom (alebo

v špecializovanom programe) pokúsiť aplikovať určité pravidlá, ktoré nám zaručia, že sa nejakej fatálnej chyby nedopustíme.

Čo rozumieme pod pojmom fatálna chyba? Možno ste už zažili niečo také – pripravili ste si jednotlivé tabuľky a s vervou ste sa pustili do tvorby aplikácie. K jednotlivým tabuľkám ste si vytvorili formuláre, tlačové zostavy a pod. Potom ste sa pustili do napĺňania databázy. A v tom momente ste narazili napríklad na jeden z týchto problémov:

- údaje, ktoré vkladáte cez formulár, by potrebovali ešte rozdeliť na menšie časti,
- pri hľadaní údajov, ktoré spĺňajú vaše požiadavky sa vám niektoré z nich nezobrazili, hoci s určitosťou môžete povedať, že ste ich do databázy vkladali,
- pribudla vám nejaká nová kategória alebo skupina, do ktorých si evidované údaje rozdeľujete a nemáte ju kam zadať.

Všetky tieto problémy majú jediné riešenie – zmeniť štruktúru tabuliek. Dá sa tomuto stavu predísť?

Vieme, že prvým krokom v tvorbe databázy riešiacej nejaký reálny problém je, že si zadefinujeme jednotlivé relácie a vzťahy medzi nimi. Na relácie potom aplikujeme už dávno pred nami vytvorené a časom overené princípy – **princípy normalizácie**. Princípy normalizácie sú pravidlá, ktoré riadia štruktúru údajov. Pravidlá navzájom súvisia a postupne sú od jednej úrovne k druhej čoraz prísnejšie.

Jedna, možno trochu odborná definícia pojmu normalizácia môže mať takúto podobu: **Normalizácia** je **reverzibilný** (vratný) proces postupného nahrádzania danej množiny súhrnom relácií, ktoré majú jednoduchšiu a regulárnejšiu štruktúru.

**Reverzibilita** zaručuje, že pôvodný súhrn informácií je možné obnoviť, a preto použitím normalizácie nedochádza ku strate informácií.

**Ciele normalizácie:**

- odstrániť nežiaduce vlastnosti spôsobované aktualizáciou, vkladaním a rušením,
- redukovať potreby reštrukturalizácie pri pridaní nového atribútu,
- umožniť reprezentáciu každej relácie v databáze,
- získať rýchle vyhľadávacie algoritmy,
- minimalizovať objem údajov v databáze.

Predpokladajme, že sú dané typy objektov, ich charakteristiky a vzťahy medzi nimi. Potrebujeme nájsť databázovú reprezentáciu. Pri jej navrhovaní je vždy najdôležitejšie vhodne navrhnúť ERD, potom ich transformovať na relácie a definovať vzájomné vzťahy. Bolo by vhodné, aby relácie vystihovali situácie v reálnom svete, aby boli efektívne a aby bola zachovaná konzistentnosť údajov.

Hlavným problémom pri vytváraní relácií je, ak hodnota jedného atribútu jednoznačne určuje hodnoty iných atribútov a my to v prvom okamihu nepostrehneme – napr. rodné číslo jednoznačne určí meno aj priezvisko čitateľa. Z toho vyplýva, že ak by sme v našom príklade knižnice, evidovali pri každom zápise do výpožičiek rodné číslo, meno a priezvisko (v jednej relácií), spolu s ďalšími údajmi, mohli by sme spôsobiť vznik dvoch typov anomálií:

- **anomália zrušenia** - vzniká, ak sa v tabuľke s výpožičkami po vrátení knihy záznam vymaže. Ak vymažeme posledný (jediný pozostalý) záznam niektorého čitateľa, stratíme údaje o jeho mene, priezvisku... aj napriek tomu, že čitateľom stále zostáva.
- **anomália vloženia** - vzniká, ak do relácie Čitatelia nemožno vložiť informáciu o čitateľovi, pokiaľ nie je zaregistrovaná prvá jeho výpožička, lebo by v tabuľke chýbal údaj o výpožičke.
- **anomália aktualizácie - problém konzistentnosti - v**zniká, ak sa po zmene údajov (čitateľ si zmení meno po svadbe alebo rozvode), nezmenia všetky súvisiace údaje v celej databáze.

Tieto anomálie, i keď nie sú príliš časté, sú nežiaduce, pretože zvyčajne je dosť nepravdepodobné, že si neskúsený používateľ uvedomí, čo nimi vykoná. Problémy konzistencie a anomálie sa však neprejavia, ak je relácia s atribútmi rodné číslo, meno, priezvisko **normalizovaná**. Problémy sa stratia.

Ešte skôr ako sa budeme zaoberať jednotlivými princípmi normalizácie, s ktorými sa môžeme častejšie stretnúť pod názvom **normálne formy,** pozastavme sa pri veľmi dôležitom pojme relačných databáz, pojme **kľúč**.

Z definície relácie vyplýva, že každý vektor, čiže záznam, musí byť jedinečný. Aby bola táto podmienka splnená, musí v každej relácii existovať určitá kombinácia atribútov, ktorá jednoznačne identifikuje každý jednotlivý vektor súradníc. Táto množina sa nazýva **kandidátny kľúč**. V relácii sa môže nachádzať viacero kandidátnych kľúčov, ale každý z nich musí jednoznačne identifikovať všetky možné vektory hodnôt.

Určite ste sa stretli s prípadom, kedy sme za kandidátny kľúč označili atribút s názvom ID. Je to prípad, kedy v relácii neexistuje prirodzený kandidátny kľúč. Napríklad za kandidátny kľúč by sme mohli prehlásiť rodné číslo čitateľa. Keďže sa ale rodné číslo v súčasnosti považuje za osobný údaj, môžeme si zaviesť vlastné číslovanie čitateľov, ktoré nebude mať ďalší, skrytý význam - ID. V tomto prípade vytvoríme **jednoduchý kľúč**.

Iná situácia nastane v tabuľke Výpožičky, v ktorej musíme zachytiť skutočnosť, že čitateľ si vtedy a vtedy požičal knihu. Kombináciou ID_Čitateľa, ID_Knihy a Dátumu_vypožičania dosiahneme jednoznačnú identifikáciu ľubovoľného záznamu. Vytvorili sme **zložený (kompozitný) kľúč**.

Neskôr, už na úrovni konkrétnej implementácie nášho databázového návrhu, sa namiesto pojmu kandidátny kľúč stretneme s pojmom **primárny kľúč**. Vo všeobecnosti možno povedať, že primárnym kľúčom nazveme ten kandidátny kľúč (ak ich je v jednej relácii viacero), ktorý je zložený z najmenšieho počtu atribútov.

Jednotlivé normálne formy môžeme popísať rôzne zložito. Môžeme použiť matematický zápis vychádzajúci z relačného kalkulu a predikátového počtu, ale my sa pokúsime o jednoduchší popis princípov normalizácie.

**Prvá normálna forma (1NF)** sa vzťahuje na štruktúru relácií. Požaduje sa, aby každý atribút (vlastnosť, príp. pole) bol definovaný nad skalárnou doménou (množine, oborom hodnôt) – t.j. aby neobsahoval odkaz na inú reláciu. Čo to znamená?

Znamená to jednu vec. Jednotlivé atribúty v relácii musíme voliť tak, aby sme ich počas práce s údajmi nepotrebovali deliť na ďalšie atribúty. Typickým problémom býva atribút adresa. Ak vieme, že v našej knižnici potrebujeme iba evidovať adresu čitateľa, na ktorú mu pošleme prípadnú upomienku, postačí nám v relácii Čitatelia uvažovať atribút Adresa. Tento atribút bude obsahovať názov ulice, čísla domu, mesta a PSČ.

Iná situácia nastane, ak potrebuje určiť počty čitateľov z jednotlivých miest. Jedným riešením by bolo vytvoriť funkciu, ktorá z atribútu Adresa vyberie informáciu o meste, ale iste tušíme, že by sa nejednalo o jednoduchý kód a celkovo by sa výkon celej databázy prudko (a zbytočne) znížil. Lepším riešením je presunúť informáciu o meste z atribútu Adresa do samostatného atribútu, prípadne relácie. Z uvedeného vyplýva, že je na tvorcovi databázy, aby určil, kedy bude jednotlivé atribúty relácií považovať za **atomárne** – **nedeliteľné**. Na 1NF sa dá previesť jednoduchým postupom každá relácia.

Ešte na jeden typ neskalárnych hodnôt si musíme dať pozor – na opakovanú skupinu hodnôt. O čo ide, najlepšie uvidíme na príklade. Povedzme, že predchádzajúci riaditeľ knižnice zaviedol pravidlo, že jeden čitateľ si smie na jeden krát požičať najviac tri knihy. Toto obmedzenie by nás mohlo zviesť k tomu, že by sme v relácií Výpožičky vytvorili atribúty Id_Citatela, Datum_vypozicania, Prva_kniha, Druha_kniha, Tretia_kniha. V tomto prípade by sme si však veľmi skomplikovali ďalšiu našu prácu. Napríklad, ak by sme chceli zistiť, ktorí čitatelia a kedy si vypožičali konkrétnu knihu, museli by sme v dotaze v klauzule WHERE uviesť tria atribúty Prva_kniha, Druha_kniha, Tretia_kniha spojené spojkou OR. Z uvedeného vyplýva, že ak sa stretnete s prípadom, kedy tú istú informáciu musíte hľadať vo viacerých stĺpcoch tabuľky, často sa jedná o problém s nedodržaním 1NF.

**Druhá normálna forma (2NF)**

Relácia je v druhej normálnej forme, ak je v prvej normálnej forme, a navyše všetky jej atribúty sú závislé na celom kandidátnom kľúči. Táto definícia vyjadruje skutočnosť, že sa nemáme snažiť v jedinej relácií zachytiť dve entity. Typickým prípadom je, ak by sme v relácii Výpožičky okrem ID_Knihy a ID_Citatela uvádzali aj atribúty Meno a Priezvisko. Kandidátnym kľúčom tejto relácie by boli atribúty ID_Knihy a ID_Citatela. Atribúty Meno a Priezvisko sú **funkčne závislé** iba od ID_citatela, teda relácia nie je v 2NF. Ak problémy vznikajúce pri navrhovaní relácií sú závislé na tom, že hodnoty niektorých atribútov plne určujú hodnoty iných, nastáva situáciu, ktorú možno

vyjadriť pojmom **funkčná závislosť**.

Jej matematická verzia má nasledujúci tvar:
Nech A, B sú atribúty relácie R a D(A), D(B) sú ich domény. Nech f je funkcia, že D(A) -> D(B), teda, pre každú D(A) existuje maximálne jedna D(B). Táto funkcia je závislá na čase (hodnoty sa časom môžu meniť), preto budeme túto závislosť nazývať nie funkciou, ale funkčnou závislosťou, napr. RČ -> Priezvisko.
Ak existuje f: A->B, hovoríme, že doména B **je závislá (funkčne závislá**) na doméne A alebo, že A determinuje B.
Ak A->B aj B->A, potom hovoríme o **jednoznačnej korešpondencii (1:1)**. Nech v relácii R platí funkčná závislosť A->B, Hovoríme, že B **silne funkčne závisí** od A, ak neexistuje žiadna vlastná (iná ako A) podmnožina A' taká, že A'->B. To by bolo napr., že iné rodné číslo vráti vždy inú kombináciu mena a priezviska. V praxi to síce pravda byť nemusí – môže existovať 152 Jožkov Mrkvičkov, ale po pridaní ďalších atribútov – adresa, dátum narodenia.... túto závislosť vieme dosiahnuť.
Možno povedať, že pri plnení požiadaviek 2NF dochádza často k vzniku ďalších relácií (a rozbitiu pôvodných) , ale ak potrebuje, vieme z nich reverzibilným procesom získať pôvodné informácie. Naviac, môžeme pridať čitateľa bez výpožičky alebo vymazať všetky výpožičky čitateľa bez toho, aby sme odstránili jeho osobné údaje, prípadne ho aj bez problémov premenovať.

**Tretia normálna forma (3NF)**
Relácia je v tretej normálnej forme, ak je v druhej normálnej forme, a zároveň všetky jej nekľúčové atribúty sú navzájom nezávislé. Alebo iná definícia - Relácia R je v tretej normálne forme, ak je v 2NF a žiaden atribút, ktorý nie je zložkou niektorého kľúča relácie R nie je tranzitívne závislý od žiadneho kľúča relácie R.
Inak povedané – relácia má tú vlastnosť, že každý atribút, ktorý nie je zložkou niektorého kľúča relácie, závisí silne od kľúča a nezávisí od neho tranzitívne.
Na vysvetlenie slovného spojenia **tranzitívne závislý**, uvažujme nasledujúci príklad - študentov UKF a ich knižnicu. O každom z nich potrebujeme určiť katedru a fakultu. Rozhodli sme sa preto vytvoriť reláciu R(RČ, KATEDRA, FAKULTA). Zrejme každá katedra patrí len jednej fakulte, teda atribút FAKULTA je silne funkčne závislý od KATEDRY a spôsobuje obdobné ťažkosti ako pri reláciách, ktoré neboli v 2NF. Môžeme sa preto stretnúť nasledujúcimi problémami:

- informácia o umiestnení katedry a fakultu sa opakuje pri každom študentovi
- pri presune katedry treba prepísať údaje o každom príslušnom študentovi
- nemožno zaznamenať údaje o katedre, ktorá nemá študentov – čitateľov.

príčinou ťažkostí je skutočnosť, že platí:
RČ -> katedra a katedra -> fakulta.

Ak to zovšeobecníme, dostaneme takýto zápis: množina atribútov C je tranzitívne závislá od A v R, ak platí A -> B a B -> C. Ťažkosti, ktoré takto vzniknú odstránime zasa rozdelením na dve relácie, ktoré už sú v tretej normálnej forme:
R1 (RČ,KATEDRA)
R2 (KATEDRA, FAKULTA)
**Ďalšie normálne formy**
Vo väčšine návrhov databáz si vystačíme s prvými troma normálnymi formami. V niektorých prípadoch však potrebujeme na údaje aplikovať ďalšie normálne formy. Niekedy sa stretneme s rozšírením tretej normálnej formy, ktorá sa nazýva **Boyce-Coddova normálna forma (BCNF)**. Táto norma platí pre prípad relácií s viacerými kandidátnymi kľúčmi, pričom musia byť splnené nasledujúce podmienky:

- Relácia musí mať dva a viac kandidátnych kľúčov.
- Najmenej dva kandidátne kľúče musia byť zložené.
- Kandidátne kľúče sa musia v niektorých atribútoch prekrývať.

Príkladom môže byť tabuľka obsahujúca zoznam škôl a jazykov, ktoré študent absolvoval

| Meno | Škola | Jazyk |
|---|---|---|
| Kováč | ZŠ | Nemčina |
| Kováč | SŠ | Slovenčina |
| Kováč | ZŠ | Slovenčina |
| Kováč | SŠ | Nemčina |
| Novák | ZŠ | Slovenčina |
| Novák | SŠ | Slovenčina |
| Novák | VŠ | Slovenčina |

Relácia ŠTUDENT je v 3NF, lebo podľa 1NF sú jej atribúty jednoduché, lebo podľa 2NF je kandidátnym kľúčom relácie kombinácia atribútov MENO+ŠKOLA+JAZYK a podľa 3NF nemá v relácii čo od čoho závisieť. Zaznamenanie údaja o novom jazyku pridanom študentovi komplikuje problém, že pre každú školu treba pridať všetky jazyky a pre každý jazyk všetky školy. Dekompozíciou uvedenej relácie získavame: R1(MENO, ŠKOLA) a R2(MENO, JAZYK).

| Meno | Škola |
|---|---|
| Kováč | ZŠ |
| Kováč | SŠ |
| Novák | ZŠ |
| Novák | SŠ |
| Novák | VŠ |

| Meno | Jazyk |
|---|---|
| Kováč | Slovenčina |
| Kováč | Nemčina |
| Novák | Slovenčina |

Ťažkosti v relácii Študent sú spôsobené „multizávislosťou“, čiže múdro povedané, že atribút Y „**multizávisí“** od atribútu X v relácii R, ak platí, že každá hodnota atribútu **X určuje nejakú množinu hodnôt atribútu Y** a táto množina pritom nezávisí od hodnoty iných atribútov relácie R.

V našom príklade pridaním mena sa pridajú všetky školy a pridaním mena sa pridajú aj všetky jazyky. Pritom škola a jazyk sú navzájom nezávislé, škola aj jazyk „multizávisia“ od mena. V našej relácii platí:

meno --- škola
meno---- jazyk

Relácia R je v **štvrtej normálnej forme**, ak v prípade, že obsahuje multizávislosť X---Y, kde Y nie je podmnožina X a XY nezahŕňajú všetky atribúty R, tak X zahŕňa i kľúč relácie R (v našom prípade meno). Jednoducho platí, že v jednej relácii sa nesmú spájať viaceré nezávislé opakované skupiny.

**A na záver...**

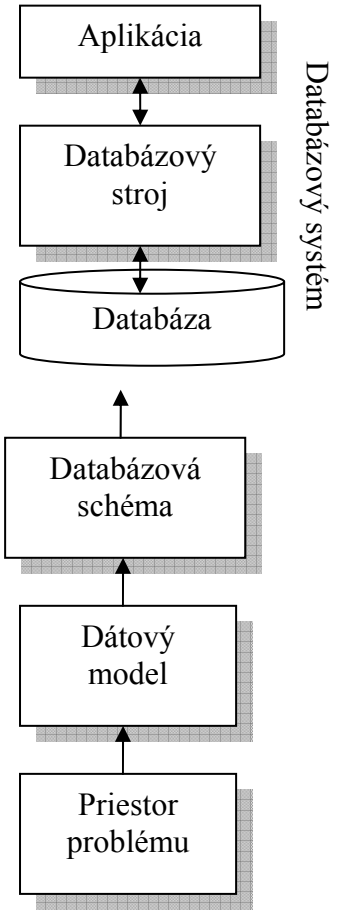
O normalizácii v procese dátového modelovania môžeme povedať, že závisí na celkovej sémantike daného modelu. Je na nás, ktoré normálne formy budeme v procese návrhu uvažovať. V drvivej väčšine prípadov si vystačíme s prvými troma princípmi normalizácie. Vyššie normálne formy musíme použiť v špeciálnych prípadoch. Pri dekompozícii je dôležité, aby bol proces reverzibilný, a aby nedošlo ku strate informácií.

## Transformácia relačného modelu do databázovej schémy

Ak sa nám podarilo z poznámok vytvoriť si dátový model, môžeme pristúpiť k ďalšiemu kroku.

Nad dátovým modelom sa nachádza definícia fyzického rozvrhnutia systému, t.j. zoznam implementovaných tabuliek a pohľadov – nazývame ju **databázová schéma**.

Najprv si vysvetlime, čo je to **pohľad**. Pohľad (v angl. prostrediach view) si predstavme ako virtuálne tabuľky, ktoré zobrazujú informácie, spĺňajúce presné kritéria určitej skupiny používateľov systému. Nad rovnakou tabuľkou alebo tabuľkami môžeme definovať viacero pohľadov, ktoré umožnia pozerať sa na rovnaké dáta niekoľkými rôznymi spôsobmi. Napríklad, ak máme tabuľku s informáciami o zamestnancoch a ich priemernom plate, pomocou pohľadu môžeme zakryť nepovolaným osobám informácie o plate.




Obr. 18 Postupnosť návrhu a jednotlivé súčasti databázovej aplikácie

Alebo, ak máme študenta a predmety, ktoré si zapísal v tomto semestri. Jeden pohľad si vytvoríme pre študenta, ktorý chce vedieť menovite, ktoré predmety si zapísal a druhý pohľad pre tvorcu rozvrhov, ktorého skôr zaujíma, koľko študentov si zapísalo daný predmet.

Databázová schéma vznikne prevedením myšlienkového modelu do dátového modelu, popísaného pojmami, pomocou ktorých ich popisujeme už voči konkrétnemu SRBD, nazývanému aj **databázový stroj** (database engine). Databázový stroj predstavuje mechanizmy, ktoré uskutočňujú vlastnú fyzickú manipuláciu s údajmi – ukladajú ich na disk a spätne ich po vyžiadaní načítavajú. Spolu s vlastnou databázou a aplikačným rozhraním tvoria databázový systém. Na úrovni databázovej schémy operujeme s pojmami ako tabuľka, spúšť, uložená procedúra a pod.

**Tabuľka, zoznam** - ide o skupinu evidovaných údajov, ktoré môžeme usporiadať do tabuľky. V riadkoch sú objekty (n-tice), v stĺpcoch ich vlastnosti (napr. pri evidencii potravín sú v riadkoch jednotlivé potraviny - chlieb, mlieko, maslo,... a v stĺpcoch ich charakteristiky - cena, hmotnosť, akosť) (Obr. 19). Každá databázová tabuľka musí obsahovať primárny kľúč, dôvod sme si vysvetlili už počas tvorby ERD a ich transformácie.

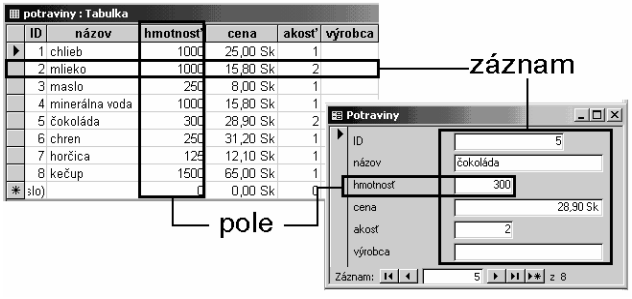
Tabuľky sú implementáciou entít v konkrétnom databázovom systéme. Preto môžeme poznamenať, že rovnako ako entity, aj tabuľky uchovávajú údaje o skutočných aj abstraktných objektoch, napríklad o udalostiach.

Tabuľky môžeme rozdeliť na:

- **údajové**, charakterizované častou manipuláciou s údajmi,
- **validačné** (vyhľadávacie, **číselníky**), ktoré obsahujú údaje, ktoré sa menia iba zriedka, ale sú dôležité pri vyhľadávaní najrozličnejších informácií.

**Stĺpec (atribút)** – Vlastnosti entity zobrazenej v tabuľke. Popisujú rovnakú vlastnosť celej triedy objektov.

**Pole (field)** - údaj popisujúci rovnakú vlastnosť evidovaného objektu. V prípade zobrazenia databázy do tabuľky - stĺpec

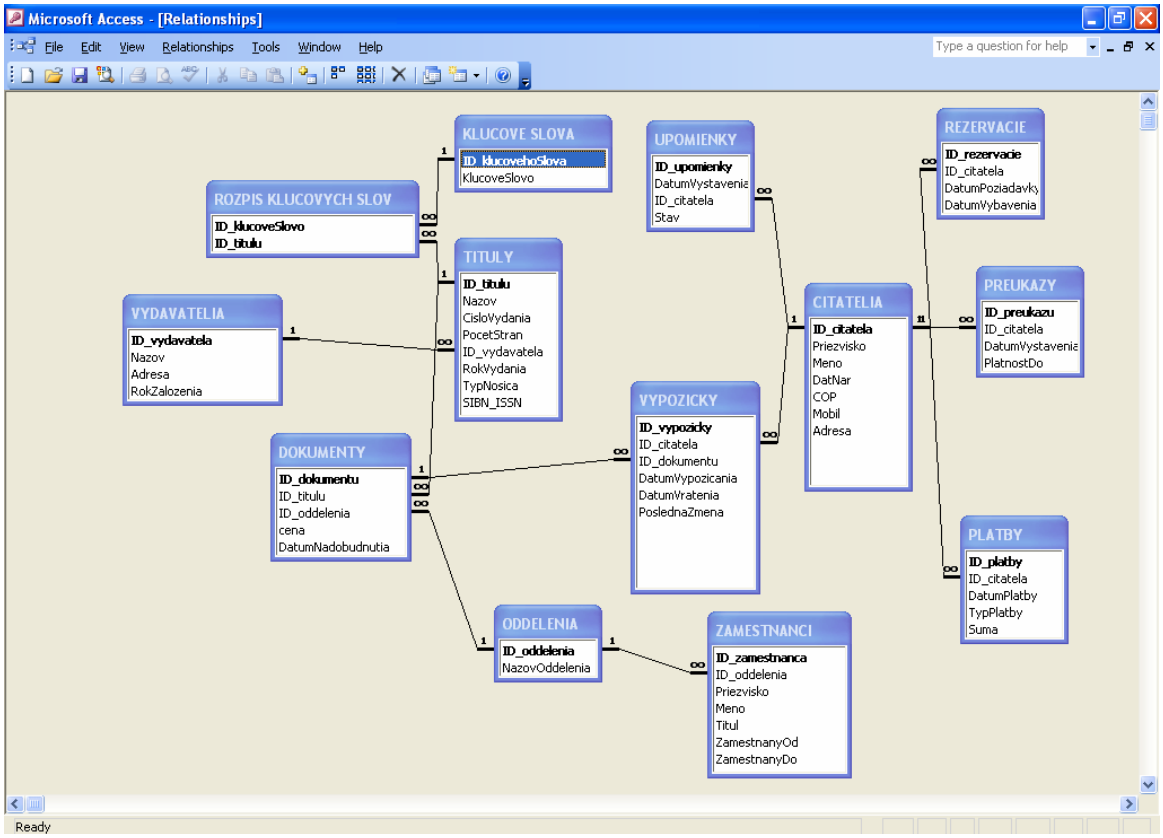



Obr. 19 Základné pojmy implementácie do konkrétneho DBS

(napr. cena alebo hmotnosť pre potraviny),
**Záznam** (často aj **record** alebo **veta, n-tica, vektor hodnôt**) - všetky charakteristiky jedného objektu (napr. všetko o chlebe, masle,...).
**Doména (Obor hodnôt)** – je úzko spätá s atribútom. Špecifikuje, v akom rozsahu môžu byť v konkrétnom atribúte vkladané údaje. Doména má tri typy vlastností:

- **Všeobecné** – meno, popis
- **Fyzické** – dátový typ, dĺžka, formát
- **Logické** – východisková hodnota, rozsah, obsah hodnoty NULL

**Spúšť (trigger)** - postupnosť príkazov, ktoré sa spustia ako reakcia na nejakú udalosť.




Obr. 20 Výsledný návrh databázy knižnice

Konečne sme sa dostali k výberu konkrétneho SRBD, v ktorom budeme náš návrh implementovať. Ak sme dodržali popisovaný postup, máme zaručené, že nech sa rozhodneme pre hociktorý zo štandardných relačných SRBD, naša databáza bude pracovať bez problémov (Obr. 20).

Akonáhle pre SRBD zadefinujeme, ako majú podľa našej predstavy údaje vyzerať, SRBD vytvorí fyzické objekty, do ktorých už môžeme ukladať naše dáta. Typickými predstaviteľmi SRBD sú BDE, Microsoft Jet (ktorý je v pozadí prostredia MS Access), MS SQL Server, Oracle, MySQL, a mnoho ďalších. Možno preto povedať, že pod pojmom databáza sa skrývajú fyzické tabuľky, pohľady, požiadavky, uložené procedúry a pravidlá, ktoré zabezpečia ochranu uložených dát. Z doposiaľ povedaného vyplýva, že skutočná a jednoznačná definícia takého bežne zaužívaného pojmu ako je pojem databáza, je náročná úloha.

Pojem databáza nezahŕňa ani samotnú aplikáciu zloženú z formulárov a tlačových zostáv, ani iný podporný softvér – middleware.

## Zaostrené na SRBD

Prístup k údajom uloženým v databáze obstaráva program, ktorému sa hovorí **SRBD – systém riadenia bázy dát**. Tento krkolomný názov vznikol z anglického **DBMS – DataBase Management System**. Medzi SRBD patria také programy, ako je Oracle, Informix, MS SQL Server, Progress. Nejde práve o lacné programy, ich cena sa pohybuje v desiatkach či skôr v stovkách tisícok korún. Pre výučbu sú však v súčasnosti všetky prístupné zadarmo, najčastejšie sa táto verzia označuje pojmom Express. Na poli SRBD existujú programy šírené zadarmo ako freeware, napr. PostgreSQL, Firebird, či MySQL.

V začiatkoch počítačov existovalo všeobecné programové vybavenie s podporou pre prácu so súbormi (file managment systems), neskôr pribudli špecializované SRBD.

Zloženie systému riadenia bázy dát:

- prekladače jazykov,
- programy pre prácu s katalógmi dát,
- výkonné programy,
- servisné programy.

**Prekladače**

- *pre definičný jazyk*

ktorým prikazujeme vytvorenie štruktúr, súborov... Dnes je v klasických SRBD implementovaný v príkazoch menu, resp. v interaktívnom rozhraní bez nutnosti písania príkazov. (vytvorenie, zálohovanie, utajenie, cache....)

- *pre manipulačný jazyk*

klasický jazyk kvôli existencii prekladačov a lenivosti užívateľov – programátorov (nepotrebujú sa učiť nič nové, resp. len minimum).

- *pre užívateľské jazyky*

používajú slová prirodzeného jazyka, sú neprocedurálne a využívajú sa hlavne pri selekcii.

**Katalóg dát**

obsahuje definície tabuliek a ostatných objektov a niekedy aj informácie o prevádzke databázy – protokol činnosti, resp. údaje o dobe prístupu, užívateľoch....Programy pracujúce s ním musia byť „inteligentné“ aby dokázali riešiť kolízie pri strate dát ako z údajovej, tak aj definičnej časti.

**Výkonné programy**

realizácia:

- ukladania, aktualizácie a výberov
- ochrana údajov pred neoprávneným prístupom (prekrýva sa s činnosťou administrátora)
- riadenie práce viacerých užívateľov
- kontrola integrity

Požiadavky na ne sú vysoké najmä vzhľadom na rýchlosť a minimum potrebného miesta v pamäti.

**Servisné programy**

sú určené viac-menej pre administrátora.

- *návrh štruktúry bázy údajov*

Samotný návrh je nealgoritmizovateľný, pretože vždy ide o špecifický proces závislý na zadaní úlohy a povahe riešiteľa. Napriek tomu sú ponúkané funkcie na určitú optimalizáciu (napr. vzhľadom na veľkosť položiek, odstránenie redundancie...) – niekedy pomôžu, ale nevyriešia vždy všetko.

- *informácie o stave systému*

ponúkajú údaje o štruktúre a organizácii údajov, prípadne komentáre od tvorcov databázy, možno protokolovať (špehovať) činnosť užívateľov apod.

- *napĺňanie BD*

systémy zvyčajne ponúkajú prostriedky, ktoré dokážu čítať údaje zo vstupných súborov iného typu ako má DS (import) alebo ich načítavajú zo špeciálnych periférií.

Niekedy sa používajú aj generátory, ktoré vložia do databáz údaje na ktorých sa činnosť systému testuje.

- *reorganizácia BD*

nezriedkavou je štruktúra pridávania, zmeny alebo zrušenie položiek v návrhu štruktúry databázy. Systémy obsahujú nástroje na automatizáciu činnosti. POZOR!!! Nie vždy sú stopercentné – zistíte. Reorganizácia je pri rozsiahlych systémoch proces časovo veľmi náročný, preto sa zvyčajne obchádza, kde len môže.

- **Ochrana dát pred zničením**

inteligentné nástroje určené na eliminovanie náhodných problémov, zálohovanie a obnovu údajov.

- **Generovanie výstupných zobrazení**

slúžia na automatizované vytváranie výstupov na tlačiareň, obrazovku alebo www-stránku. Zvyčajne je možné automatické výstupy manuálne upravovať a na základe zmien vygenerovať dané zostavy nanovo.

## Dátová integrita

Doteraz sme sa zaoberali modelovaním entít a definovaniu vzťahov medzi nimi. Ďalším krokom v procese dátového modelovania je zachytenie a inmplementácia pravidiel, pomocou ktorých databázový systém zistí, že reálne dáta, ktoré obsahuje, sú správne. Túto činnosť zabezpečuje práve **dátová integrita.**

Dátovú integritu si však nemôžeme mýliť s aplikačnými pravidlami (business rules), ktoré v sebe zahrňujú všetky obmedzenia definované v systéme, nielen obmedzenia stanovené dátovou integritou. K aplikačným pravidlám patrí napríklad bezpečnosť systému.

Dátovú integritu môžeme implementovať na rôznych úrovniach, pričom platí, že vyššia úroveň integrity sprísňuje princípy stanovené nižšou úrovňou integrity. Môžeme sa stretnúť s nasledujúcimi úrovňami integrity:

1. **doménová integrita** – definuje obor hodnôt jednotlivých atribútov, definuje platné hodnoty konkrétneho atribútu
2. **prechodová integrita** – definuje prípustné stavy, ktorými môže daný atribút prechádzať. Typickým príkladom je stav objednávky. Prípustné stavy objednávky sú napríklad zadaná, priobjednaná, prichystaná, doručená, čiastočne vybavená, stornovaná, vybavená. Na začiatku je objednávka zadaná, môže prejsť do stavu prichystaná, čiastočne vybavená, stornovaná. Zo stavu čiastočne vybavená môže tiež prejsť do stavu vybavená alebo naopak do stavu stornovaná (čím vzniká pomerne veľký problém s vrátením peňazí).
3. **entitová integrita** – zabezpečuje integritu jednotlivých entít v systéme, napríklad vďaka existencii primárneho kľúča.
4. **referenčná integrita** – kontroluje zachovanie vzťahov medzi reláciami, čím vlastne zabraňuje situácii, kedy by sa z cudzieho kľúča stala sirota, t.j. žiadny riadok v cudzej relácii nesmie obsahovať takú hodnotu cudzieho kľúča, ktorá nemá zodpovedajúci záznam v primárnej relácii. Siroty môžu vzniknúť troma spôsobmi:
   - do cudzej relácie sa pridá riadok s kľúčom, ktorý nezodpovedá žiadnemu kandidátnemu kľúču primárnej relácie
   - kandidátny kľúč v primárnej relácii sa zmení
   - záznam primárnej relácie, na ktorú sa sirota odkazuje, sa odstráni

   V súvislosti s referenčnou integritou sa možno stretnúť s pojmami aktualizácia a odstránenie údajov v kaskáde.
5. **databázová integrita** – kontroluje databázu ako celok.
6. **transakčná integrita** – na rozdiel od všetkých ostatných má procedurálny charakter, t.j. obsahuje postupnosť krokov, ktorá ovláda spôsoby manipulácie s databázou. Pred a po skončení transakcie musia byť v databáze splnené všetky definované obmedzenia integrity, ale počas spracovania transakcie môžu byť niektoré z nich dočasne porušené a databáza sa môže nachádzať v nekonzistentnom stave.

Prvé tri integritné obmedzenia platia na úrovni relácie. Referenčná integrita zaisťuje zachovanie potrebných vzťahov medzi reláciami, databázová integrita kontroluje databázu ako celok a transakčná

integrita riadi spôsoby manipulácie s údajmi v jednej alebo medzi viacerými databázami.
Na integritu sa môžeme pozerať aj iným spôsobom. Stav bázy údajov je daný stavom jej objektov. Pod objektami rozumieme súbory, záznamy, polia, relácie, atď. Zo všetkých možných stavov, ktoré môže databáza nadobudnúť, sú pre používateľa zaujímavé len tie, ktoré reprezentujú **reálnu** (pravdivú) informáciu. Takýto stav sa nazýva **konzistentný**.

Nekonzistentný stav môže nastať napr. vtedy, keď sa v položke vek vyskytne záporné číslo, alebo pri kombinácii veku a stavu bude dvojica napr. [10, rozvedený] (i keď jeden nikdy nevie :-)).

Systémy riadenia bázy dát poskytujú funkcie, pomocou ktorých možno zabrániť, aby k takýmto stavom dochádzalo (väčšinou môžeme v návrhu tabuľky nastaviť obmedzujúce kritériá pre každé pole tabuľky).

Ďalšou dôležitou požiadavkou je požiadavka na integritu – celistvosť databázy. Je to pohotová fyzická disponibilita – databáza vie zareagovať na požiadavku používateľa a dať mu k dispozícii údaje, ktoré žiadal. Báza údajov musí byť pritom v konzistentnom stave.
Integritu možno narušiť nasledovnými spôsobmi:

- zlyhanie hardware (poškodenie disku),
- zlyhanie software (operačného systému alebo aplikačného programu, ktorý pracuje s databázou),
- chybnými údajmi – vinou používateľa.

Pred narušením integrity je nutné databázu chrániť. V krátkosti uvedieme konvenčné metódy ochrany databázových súborov:

**1. viacnásobné kópie súboru**

Existuje niekoľko navlas rovnakých kópií na rôznych miestach (HDD alebo zálohovacie zariadenia). Výhodou je, že možno súčasne brať údaje z viacerých miest a v prípade poškodenia údajov na jednom médiu máme okamžite k dispozícii tie isté údaje na inom médiu. Nevýhoda spočíva v dvoj (alebo viac) násobnom nároku na pamäťové prostriedky a v nutnosti zápisu zmien do každej kópie osobitne a súčasne.

**2. kopírovanie súborov**

V určenom časovom slede sa vytvára kópia súboru. V prípade poruchy sa obnoví a údaje sú v stave ako pred poslednou zálohou. Aby sa nemuseli vytvárať denne kompletné kópie, zvykne sa urobiť zálohovanie napr. raz za týždeň (hovoríme o kontrolnom bode) a po ďalšie zálohovanie sa vykonáva zápis iným spôsobom (pozri ďalej).

**3. viaceré verzie súborov**

V prípade zálohovania sa postupuje tak, že sa ponechá starý súbor, a uloží sa aj nový – vytvára sa viac generácií zálohy (používa sa najmä v bankách).

**4. inkrementálne kopírovanie**

Po vytvorení kontrolného bodu (kompletné zálohovanie) sa zálohujú len časti, ktoré boli zmenené – údaje, ktoré sa zmenili sa označia a rovnako sa označia aj tie, ktoré ich majú nahradiť.

**5. zaznamenávanie zmien**

Po kontrolnom bode sa zaznamenávajú zmeny (operácie, ktoré sa s údajmi vykonali) – zaznamená sa pôvodná hodnota a zaznamená sa aj nová. Vďaka týmto súborom je potom možné realizovať jednak obnovu pôvodných údajov spätne, jednak sa dopracovať k aktuálnym údajom od kontrolného bodu. Súboru, ktorý postupnosť zmien uchováva sa hovorí žurnálový. Kvôli zabezpečeniu je ho možné viesť vo viacerých kópiách.

Okrem ochrany údajov v súboroch v databázových systémoch funguje aj ochrana, ktorá databázu chráni pred nekonzistentným stavom. Napr. operácia výberu z banky by mohla viesť k tomu, že na účte by zostal k dispozícii záporný zostatok.

Takúto operáciu by mal slušný databázový systém odmietnuť a informovať o tom používateľa (resp.

program, ktorý ju prikázal). Operácia meniaca údaje v databáze však zriedkakedy býva elementárna, zvyčajne ide o postupnosť – transakciu. Transakcia je postupnosť operácií nad objektami databázy, ktorá z hľadiska používateľa realizuje jednu ucelenú operáciu.

Napríklad pri presune hotovosti 5000,- z účtu A so sumou 50000 na účet B so sumou 13000. V momente presunu je databáza v nekonzistentnom stave, lebo na jeden účet bola suma zapísaná, ale z druhého ešte odpísaná nebola alebo opačne.

## Relačná algebra

V predchádzajúcich kapitolách sme hovorili o reláciách, ktoré plánuje použiť, a o ktorých predpokladáme, že sa z nich stanú po výbere konkrétneho vývojového prostredia a SRBD skutočné, fyzické tabuľky.

Existujú však prípady, kedy je výhodnejšie použiť odvodené relácie, ktoré vzniknú na základe iných relácií, nie len na základe určitých atribútov. Takýmto odvodeným reláciám sa v rôznych prostrediach hovorí rôzne, môžeme sa stretnúť pojmami pohľad (view), dotaz (query), dynamická sada (dynaset, recordset) a pod.

Odvodené relácie sa definujú pomocou **relačných operátorov**. Na vyjadrenie relačných operácií sa používa niekoľko jazykov, často sa stretnete napríklad s jazykom SQL (Structured Query Language). Pre vytváranie odvodených relácií sa používa príkaz SELECT, ktorý má tento všeobecný zápis:

```
SELECT <ZoznamPoli>
FROM <ZoznamMnozinZaznamov> <TypSpojenia> JOIN <SpojovaciaPodmienka>
WHERE <VyberoveKriteria>
GROUP BY <ZoznamPoliNaZoskupenie>
HAVING <VyberoveKriteria>
ORDER BY <Zoznam PoliNaZoradenie>
```

Treba upozorniť na to, väčšina operácií relačnej algebry, a teda aj SQL, pracujú s **logickými operátormi**, t.j. operátormi, ktoré vrátia výsledok typu boolean. Na rozdiel od klasickej algebry v databázach pracujeme aj s hodnotou Null. Z tohto dôvodu sa hovorí o **trojhodnotovej logike**, čo sa prejaví aj na pravdivostných tabuľkách:

| AND | TRUE | FALSE | NULL |
|---|---|---|---|
| TRUE | TRUE | FALSE | NULL |
| FALSE | FALSE | FALSE | NULL |
| NULL | NULL | NULL | NULL |

| OR | TRUE | FALSE | NULL |
|---|---|---|---|
| TRUE | TRUE | TRUE | NULL |
| FALSE | TRUE | FALSE | NULL |
| NULL | NULL | NULL | NULL |

Konečne sa dostávame k vlastným relačným operátorom. Existujú štyri typy relačných operátorov:

1. **reštrikcia** (**obmedzenie**) – vráti iba záznamy, ktoré vyhovujú výberovému kritériu. Vyjadruje sa pomocou podmienky v klauzule WHERE. Táto podmienka môže byť teoreticky ľubovoľne zložitá, môže obsahovať operátory OR, AND, NOT. (SELECT * FROM tabulka WHERE podmienka). Môžeme konštatovať, že reštrikcia robí horizontálny výrez z pôvodnej relácie.
2. **projekcia** (**premietanie**) – vráti vybranú podmnožinu atribútov, robí teda vertikálny výrez z pôvodnej relácie. (SELECT stlpec1, stlpec2 FROM tabulka)
3. **spájanie** – je najbežnejším relačným operátorom. Zabezpečujú reverzibilnú povahu dekompozície, čiže aj keď sme boli nútený pravidlami normalizácie vytvoriť viacero relácií, pomocou operátoru spojenia ich môžeme spätne zostaviť na základe porovnania jedného alebo viacerých polí. V SQL je spájanie vyjadrené slovom JOIN. Spojenie môže byť:
   - **vnútorné** (INNER), ktoré sa ďalej delí na **spojenie na rovnosť** a **theta-spojenie**. V spojení na rovnosť sa porovnanie robí na základe rovnosti porovnávaných hodnôt polí. (SELECT meno, priezvisko FROM citatelia INNER JOIN vypozicky ON citatelia.id_citatela = vypozicky.id_citatela). Vyberie len tie záznamy, kde sa obe polia rovnajú.

     Možno povedať, že všetky spojenia, v ktorých operátorom porovnania nie je =, sú

theta-spojenia. Operátormi porovnania sú <, <=, <>, =>, >.Typickým príkladom je nájdenie mena a priezviska ľudí, ktorí sú vyšší ako priemer.

- **vonkajšie** (OUTHER), ktoré môže byť **ľavé** (LEFT), **pravé** (RIGHT) a **úplné** (FULL). Vonkajšie epojenie pracuje aj s hodnotou Null. Ak sa jedná o ľavé vonkajšie spojenie, tak sa do výsledku dostanú všetky záznamy, v ktorých sa hodnoty porovnávaných polí rovnajú, ako aj záznamy z prvej relácie, ktorým v druhej relácii prislúcha hodnota Null. Pri pravom vonkajšom spojení je to opačne. (SELECT * FROM tabulka1 LEFT JOIN tabulka2 ON podmienka)

4. **delenie** – vráti všetky záznamy z jednej relácie, ktorých hodnoty sa zhodujú zo všetkými zodpovedajúcimi hodnotami druhej množiny záznamov.

Uvedené operátory možno kombinovať.

K týmto operátorom sa pridávajú ďalšie štyri, ktoré pochádzajú z teórie množín, pričom sú mierne upravené pre použitie s reláciami. Sú to

1. **zjednotenie** – zreťazenie dvoch relácií, čiže záznamy z druhej relácie sa pripoja za záznamy prvej relácie. Podmienkou je, aby mali obe relácie rovnaký počet atribútov s rovnakým oborom hodnôt. V SQL sa používa vyhradené slovo UNION.
2. **prienik** – používa sa na vyhľadanie duplicitných záznamov, t.j. záznamov, ktoré sa nachádzajú v oboch reláciách.
3. **rozdiel-** používa sa na vyhľadávanie „sirôt“. Výsledkom sú záznamy, ktoré patria iba do jednej relácie, v druhej sa nenachádzajú.
4. **kartézsky súčin** – spojenie dvoch relácií A a B, medzi ktorými neexistuje vzťah. Výsledkom je A x B. Každý záznam z tabuľky A sa skombinuje so všetkými záznamami z tabuľky B.

## Interná organizácia údajov

Jednou z najdôležitejších častí systému riadenia bázy dát je fyzická vrstva. Keďže databázové systémy pracujú s veľkým množstvom dát, nie je možné tieto držať v systémovej pamäti. Z dôvodu potreby ukladania dát na disk je potrebná aj služba databázového systému, ktorá to zabezpečí. Na služby operačného systému sa nemožno spoliehať. Hlavným dôvodom je rozdielnosť súborových systémov rôznych operačných systémov. Navyše, prepojenie na služby OS by znamenalo vznik závislosti databázového systému na konkrétnom operačnom systéme. Taktiež by mohlo dochádzať k fragmentácii údajov. V niektorých prípadoch však už pozorujeme prepojenie databázových a operačných systémov, napríklad MS SQL server firmy Microsoft je optimalizovaný pre operačný systém tej istej firmy.

Pamäte, na ktoré možno ukladať údaje, hierarchicky delíme na **primárne, sekundárne** a **terciárne**. Primárne pamäte sú priamo dostupné z CPU. Patria k nim **registre, cache a operačná pamäť.** Druhou skupinou pamätí sú sekundárne pamäte, kam patria periférne zariadenia s priamym prístupom. Údaje v sekundárnych pamätiach nie sú dostupné z CPU, musia byť najprv skopírované do primárnej pamäte. Patria sem virtuálna pamäť, magnetické disky, CD-ROM, .... . Terciárne pamäte sú tvorené perifémymi zariadeniami so sekvenčným prístupom – napr. magnetické pásky. Databázy sú ukladané na magnetických diskoch, keďže sú príliš veľké na to, aby sa zmestili do operačnej pamäte. Cena za uloženie údajov je rádovo menšia ako u primárnych pamätí.

**Charakteristiky diskov**

- Operácie čítania a zápisu (READ/WRITE) prenášajú bloky dát medzi diskom a operačnou pamäťou.
- Prístup k dátam resp. zápis dát na disky je „drahou“ operáciou, každý prístup na disk, musí byť starostlivo zvážený.
- Disk je zariadenie s priamym prístupom k dátam.
- Čas prístupu k údajom (oproti RAM) nie je rovnaký, pre je veľmi dôležité, aká je vzájomná poloha údajov na disku.

Prístup na disk môžeme charakterizovať niekoľkými parametrami:

- Prístupová doba k bloku údajov.
- Seek time – nastavenie ramena na požadovanú stopu (1-20 milisekúnd).
- Rotational latency – nastavenie hlavy na požadovaný sektor.
- Transfer time – prenos bloku dát (cca 1msek/4KB).
- Seek time a rotational latency sú dominantné.
- Rotational delay – 0-10 milisekúnd.

V snahe o čo možno najrýchlejší prístup k údajom je potrebné udržiavať súvisiace údaje blízko seba - na tom istom **bloku**/sektore, na tom istom track-u, na tom istom cylindri. **Bloky** v súbore by mali byť na disku ukladané sekvenčne, aby sa minimalizoval seek a rotational delay. **Disk je médium s priamym prístupom** – nie sekvenčný. Čas pohybu hlavičky zahŕňa seek time a rotational latency. Cieľom je redukovať počet prístupov na disk a vyhľadávací čas.

Najnižšia vrstva SRBD spravuje priestor na disku. Vyššie vrstvy ju volajú, keď potrebujú alokovať resp. dealokovať stránku, čítať resp. zapisovať do stránky. Pri požiadavke na sekvenciu stránok by mali byť stránky naalokované sekvenčne na disku

Rôzne operačné systémy používajú rôzne súborové systémy. Buffer manažment v SRBD musí mať schopnosť označovať (pin) stránky v buffri, presúvať stránky na disk, meniť stratégiu nahradzovania stránok a predvýberu stránok na základe prístupového vzoru.

Rozlišujeme dva typy formátov stránok:

- Záznamy s pevnou šírkou
- Záznamy s premenlivou šírkou

Vyššie vrstvy SRBD pracujú so záznamami alebo súbormi záznamov. Súborom sa rozumie kolekcia stránok obsahujúcich záznamy. Súbor musí podporovať operácie insert, delete, modify na zázname. Čítanie záznamu sa robí podľa record id.

## Základné organizácie údajov

K základným organizáciám údajov na disku patria **hromada (heap), sekvenčná organizácia, zreťazená organizácia, zotriedená organizácia a hash.**

**Hromada – Heap**

Záznamy sú uložené v ľubovoľnom poradí v blokoch, ktoré môžu fyzicky nasledovať za sebou. Ak nenasledujú za sebou, musia byť prepojené smerníkmi alebo adresy blokov súboru musia byť niekde uložené. Záznamy sa vyhľadávajú tak, že sa od začiatku do konca súboru číta záznam po zázname a porovnáva sa, či to je hľadaný záznam. Nájdený záznam sa aktualizuje a uloží na to isté miesto. V prípade zmazania sa označí záznam za neplatný, čím sa vytvorí diera v súbore, ktorú možno využiť pri nasledujúcom vkladaní do súboru. Nový záznam sa potom vloží na miesto prvej diery.

**Netriedené (heap) súbory**

Najjednoduchšia štruktúra súboru obsahuje nezotriedené záznamy. Keď sa súbor mení, alokujú, resp. dealokujú sa ďalšie diskové stránky. Na podporu operácií na záznamoch je potrebné pamätať si informácie o stránkach v súbore, voľné miesto na stránkach a záznamy na stránkach.

**Sekvenčná organizácia**

Sekvenčná organizácia je vhodná pre aplikácie, ktoré vyžadujú sekvenčný prístup. Záznamy sú zotriedené podľa kľúča. Vkladanie záznamu prebieha tak, že sa nájde miesto, kde má byť záznam vložený, a ak je v bloku voľné miesto, tak sa tam záznam uloží. Ak už miesto v bloku nie je, vytvorí sa nový blok tzv. **blok pretečenia** a záznam sa do neho uloží.

**Zreťazená organizácia**

Vyhľadávanie v zreťazenej organizácii spočíva z postupného sledovania smerníkov a z testovania, či

sme už našli hľadaný záznam. Aktualizácia údajov pozostáva z po nájdení vety, ktorú treba zmeniť sa zrobí prostá náhrada záznamu s novými hodnotami pričom musí byť zachovaný smerník ak má zostať záznam v pôvodnej reťazi, smerník ukazujúci na zmazávaný záznam sa nahradí smerníkom zmazávaného záznamu. Pri vkladaní sa reťaz predĺži o jeden záznam. Vkladaný záznam sa umiestni na začiatok alebo na koniec.

**Organizácia s utriedenými záznamami**

Záznamy sú uložené na adresách nasledujúcich za sebou tak, že ich usporiadanie zodpovedá usporiadaniu nad jednou alebo viacerými položkami – **kľúčmi**. Na vyhľadanie záznamu sa používa binárne vyhľadávanie podľa kľúča Po vyhľadaní môžeme aktualizovať všetky položky okrem položiek triediaceho kľúča. Pri prvom vytriedení sa nechávajú v každom bloku diery, ktoré možno využiť pri nasledujúcom vkladaní ďalšieho záznamu. V prípade vyčerpania miesta v bloku, ďalšie záznamy sú ukladané do zvláštnych blokov pretečenia a sú prístupné pomocou smerníka v záhlaví bloku. Ak chcem nejaký záznam zmazať, jednoducho sa označí za neplatný a vznikne diera. Tú treba obvykle preniesť na koniec bloku.

**Organizácia Hashom**

Organizácie hashom je vhodná pre aplikácie vyžadujúce priamy prístup (random access). Záznamy sú nezotriedené. Adresa bloku, v ktorom sa nachádza požadovaný záznam, sa vypočíta pomocou **hashovacej funkcie**. Nech K je množina všetkých kľúčov a B je množina adries bucket-ov. **Hashovacia funkcia je zobrazenie z K do B**. **Bucket** je najčastejšie blok na disku.

Vkladanie záznamu s kľúčom Ki znamená vyrátať hodnotu hashovacej funkcie, ktorá poskytne adresu bucketu pre záznam. Vyhľadávanie záznamu s kľúčom Ki znamená vypočítať hodnotu hashovacej funkcie, ktorá poskytne adresu bucket-u, v ktorom nájdeme postupným prejdením záznamov požadovaný záznam. Odstránenie záznamu znamená jeho vyhľadanie v buckete a jeho vymazanie.

**Hashovanie**

Hashovacia funkcia musí byť zvolená so zreteľom na to, že rozdelenie záznamov môže byť rovnomerné, ale aj náhodné. Veľkosť databázy rastie veľmi rýchlo. Máme 3 možnosti vytvorenia statickej hashovacej funkcie - zvolenie funkcie založenej na momentálnej veľkosti databázy, zvolenie funkcie založenej na predpokladanej veľkosti databázy alebo môžeme periodicky prerábať hashovací súbor podľa veľkosti databázy. Inou možnosťou ako reagovať na zmenu veľkosti databázy je použitie dynamického hashovania.

Hashovanie môže byť:

- Interné
- Externé
- Statické
- Dynamické

# Indexy

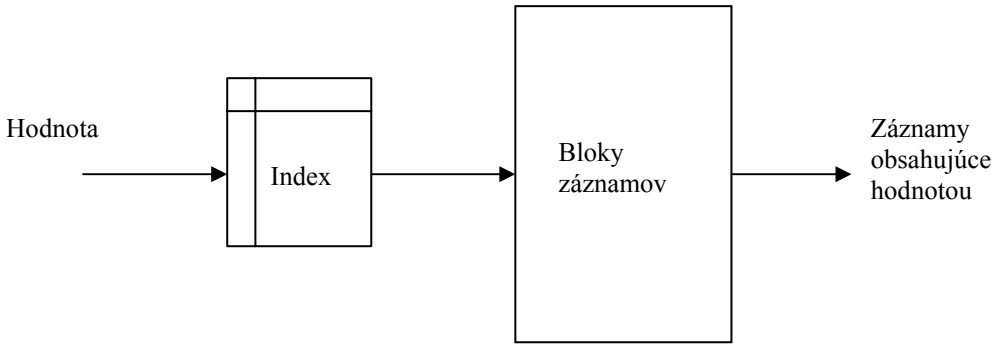
## Čo je to index?

Okrem vlastných dát, ako meno zamestnanca, výška platu a pod. , sú v databáze uložené aj dodatočné pomocné informácie. Tieto informácie majú za úlohu zrýchliť spracovanie požiadaviek na databázu.

Jednou z najviac používaných pomocných informácií je **index**. Index je v podstate špeciálna zotriedená prístupová štruktúra podobná indexom v knihách. Indexy v knihách sú zoznamy dôležitých pojmov zotriedené v abecednom poradí na konci knihy, kde pri každom pojme sa nachádzajú čísla strán, na ktorých sa daný pojem vyskytuje. Teda vyhľadanie pojmu je veľmi rýchle, hľadaný pojem sa nájde v indexe a s nim aj čísla strán, na ktorých sa pojem nachádza. Ak by takýto index neexistoval, nájdenie pojmu by zodpovedalo sekvenčnému prehľadávaniu knihy. Ak uvedenú analógiu aplikujeme na počítačový systém, prídeme k tomu, ak máme dôležité záznamy relatívne malé resp. záznamov je málo, potom je možné, že bude pamäť systému natoľko postačujúca, aby ich uchovala.

Ale ak je týchto údajov veľmi veľa na to, aby sa všetky vošli do hlavnej pamäte, potom sa údaje musia získavať z pevného disku. Takéto presuny z disku do hlavnej pamäte sú náročné a hlavne časovo veľmi drahé. Za čas strávený presunom údajov z disku do pamäte by sa dalo vykonať niekoľko stotisíc operácií, a preto je počet prístupov na disk určujúci pre celkový čas vykonávania programu.

Indexy ja javia ako jedno z efektívnych vylepšení databáz. Indexy môžeme celkom jednoducho umiestniť do pamäte, vyhľadať potrebné informácie a následne ich získať prístupom na disk, čo je iste efektívnejšie a hlavne časovo menej náročnejšie než prehľadávať postupne celú databázu resp. tabuľku.




## Výhoda indexov

Najväčšou výhodou indexov je, že ich môžeme niekoľko nad tými istými dátami a následne používať pre konkrétny dotaz vybrať index, ktorý nám najviac vyhovuje. Na zvýraznenie výhody indexov použijeme analógiu. Predstavme si, že sme v knižnici, ktorá obsahuje 1000000 kníh a má ich zoradené podľa mien autorov. Potrebujeme nájsť 3 knihy, ale pre každú z týchto kníh poznáme len niektorú z informácií (meno autora, názov titulu, ISBN číslo). Nájdenie knihy s menom autora by bolo celkom jednoduché, zájdeme medzi poličky s knihami začínajúcimi na meno autora. Iná situácia však nastane, ak by sme potrebovali nájsť ďalšie 2 knihy. Museli by sme stráviť v knižnici obrovské množstvo času a tak namiesto hľadania medzi poličkami použijeme katalógové lístky zoradene podľa titulov. Následne použijeme ďalšie lístky zoradené podľa ISBN čísel. Tieto katalógové lístky predstavujú indexy.

Použitie indexov v databázových systémoch nie je nevyhnutné. Vytvárať dotazy a manipulovať s dátami môžeme aj bez nich, avšak celý proces bude pomalší. Primárny dôvod pre použitie indexov je zvýšenie rýchlosti vykonávania databázových operácií. Využitím indexov dokážeme požadované informácie nájsť oveľa jednoduchšie ako pri sekvenčnom prehľadávaní celej tabuľky. Indexy taktiež urýchľujú dotazy, ktoré spájajú tabuľky a urýchľujú operácie ORDER BY a GROUP BY. Vytváranie indexov na kľúčových hodnotách je taktiež cestou k zabezpečeniu jedinečnosti záznamu. Indexovať je potrebné napr. polia, na základe ktorých sa vykonáva triedenie (zoraďovanie) dát. Zoznam zákazníkov sa spravidla zoraďuje podľa mena a napr. objednávky sa triedia podľa dátumu. V týchto prípadoch sa dáta netriedia podľa primárneho kľúča ani sa nevykonáva operácia spájania tabuliek (JOIN). Vytvorenie indexu na týchto poliach vedie k uľahčeniu a zefektívneniu procesu triedenia.

Využívanie indexov má oproti väčšine výhod aj nevýhody. Vytváranie indexov zaberá istý priestor na disku a čas. Pri údržbe alebo modifikácii databáz je potrebné vykonávať aj modifikácie indexov.

Maximálny počet indexov v danej tabuľke závisí na tom, ako často sa daná tabuľka bude aktualizovať. (Index vyžaduje istú réžiu iba v prípade pridávania nového záznamu, alebo pri takej aktualizácii, kedy sa mení niektoré z polí indexu.) Napr. tabuľku objednávok systém zrejme aktualizuje často. Neodporúča sa preto v tomto prípade vytvárať veľké množstvo indexov. Stačí len vytvoriť primárny kľúč a indexy potrebné pre operácie spájania tabuliek. Viac indexov je osožné využiť napr. v tabuľke výrobky, ktorá sa neaktualizuje často ale jej dáta systém využíva rôznymi spôsobmi.

Indexy je užitočné vytvárať na:

- všetko čo potrebujeme často vyhľadávať,
- logické primárne a cudzie kľúče,
- stĺpec, ktorý je často využívaný v zotriedenom poradí,
- stĺpec, ktorý je pravidelne využívaný v spojeniach, nakoľko systém dokáže realizovať spojenie oveľa rýchlejšie so stĺpcami, ktoré sú indexované než s tými, ktoré indexované nie sú,
- stĺpec, ktorý je často vyhľadávaný pre vyjadrenie intervalu hodnôt,
- stĺpec, ktorý je pravidelne využívaný klauzulou WHERE (Keď sa vykonáva SELECT, optimalizátor dotazov pozerá na klauzulu WHERE na zistenie, či existujúci index je použiteľný. Indexovaný stĺpec použiteľný pri klauzule WHERE môže zvýšiť efektivitu vykonávania operácií UPDATE a DELETE.

Index sa neodporúča vytvárať:

- ak index je nepoužiteľný pre optimalizáciu dotazu,
- ak viac ako 10 až 20% záznamov sa opakujú,
- ak stĺpce obsahujú iba jeden, dva alebo tri jedinečné hodnoty,
- ak stĺpec, ktorý má byť indexovaný je príliš dlhý (viac ako 20 bytov)
- ak použitím indexov vznikne neprehľadnosť a údržba indexov je väčšia ako ich využitie.

### Samotná štruktúra indexov

Prístupová štruktúra index je zvyčajne definovaná nad jednoduchým poľom, nazývanom **indexovacie pole**. Indexy vytvárajú tzv. **indexový súbor** resp. **indexovú tabuľku**, čiže zoznam záznamov s dvoma poliami, kde prvé pole obsahuje **indexový kľúč** resp. **hodnotu indexového poľa** a druhé pole obsahuje **smerník** na blok dátového súboru, kde sú záznamy uložené.

Samotné záznamy v indexovom súbore môžu byť implementované rôzne napr.

- <hodnota kľúča je smerník na záznam>
- <vyhľadávací kľúč, smerník na záznam>
- <vyhľadávací kľúč, pole smerníkov na záznamy>

Prvá možnosť je použitá v prípade, že použijeme hodnotu kľúča ako parameter funkcie, ktorá priamo určí záznam v tabuľke. Táto možnosť je akýmsi špeciálnym indexom a je založená na organizácii dátového súboru a nevytvára si zvláštny indexový súbor.

Druhá a tretia možnosť vytvárajú indexové súbory zvlášť pozostávajúce z párov, pričom tretia možnosť poskytuje lepšie využitie priestoru. Musíme ale podotknúť, že takto vytvorené záznamy v indexe sú rôznej dĺžky v závislosti od vyhľadávacieho kľúča.

## Klasifikácia indexov

Množstvo databázových systémov umožňuje použitie rôznych indexov v závislosti od požiadaviek, a taktiež môžu slúžiť k overeniu hodnôt, ktoré majú byť jedinečné. V tejto časti popíšeme jednotlivé druhy indexov, ich použitie, výhody, nevýhody a iné nutné vlastnosti.

V podstate existujú dva základné druhy indexov:

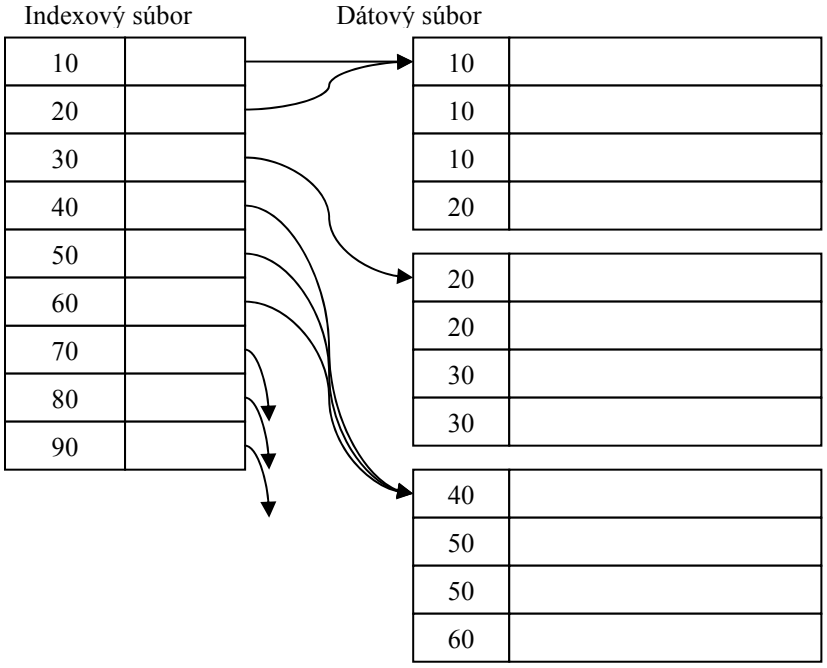
- **zotriedené indexy** – vyhľadávacie kľúče sú zotriedené v indexovom súbore, a preto je možné použiť binárne vyhľadávanie.
- **hash indexy** – vyhľadávacie kľúče sú rovnomerné distribuované do blokov (typicky diskový blok) uchovávajúcich jednu alebo viacero hodnôt. Na distribuovanie slúži tzv. **hašovacia funkcia**.

### Združovací (Cluster) index

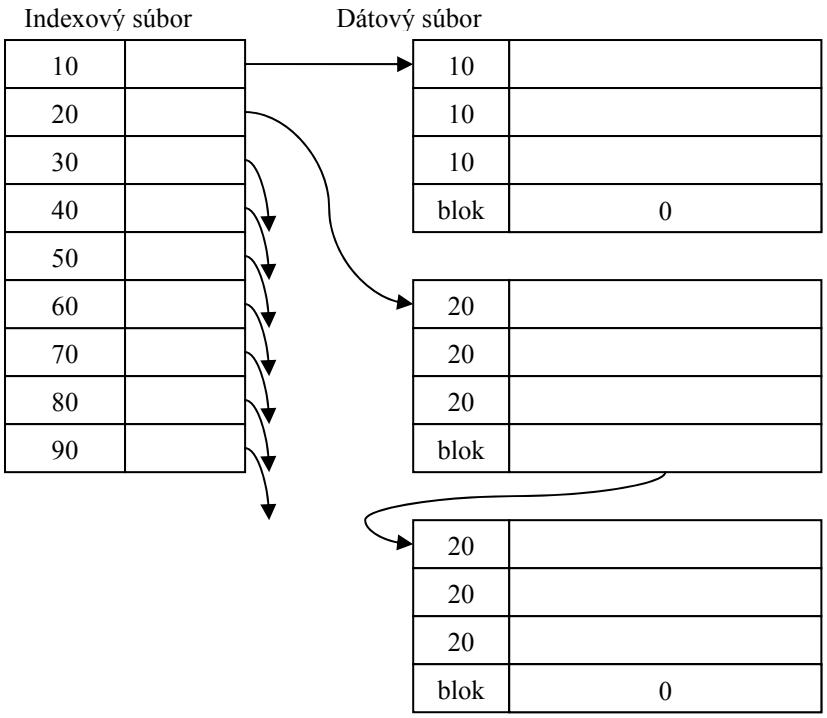
Ak je indexový súbor zoradený rovnako ako dátový súbor hovoríme o **združenom indexe**. Z toho

vyplýva aj to, že združený index môže byť definovaný len jeden pre tie isté údaje. Tento index nie je zadefinovaný nad primárnym kľúčom, a teda môže obsahovať viaceré rovnaké hodnoty, ktoré sú združované. Združovací index udržuje riadky tabuliek usporiadané podľa vyhľadávacieho kľúča, preto nájdenie riadkov tabuľky s rovnakými hodnotami je veľmi efektívne. Stačí nájsť prvý výskyt záznamu, a potom už len vyčítať hodnoty sekvenčne.

Pretože združovací index patrí medzi usporiadané indexy má rovnakú štruktúru. Jeho hodnoty sú usporiadané podľa vyhľadávacieho kľúča. Indexový súbor pozostáva z položiek s dvoma poľami. Prvé pole je toho istého typu ako pole v dátovom súbore, nad ktorým bol index vytvorený a druhé pole obsahuje smerník na záznam do dátového súboru určenú vyhľadávacím kľúčom. Najčastejším riešením implementácie takéhoto indexu sú B-stromy.




Problémom tohto typu indexu je jeho vysoká cena udržiavania pri operáciách INSERT, UPDATE a DELETE. Najväžnejšie je to pri operáciách INSERT a UPDATE, keďže je dátový súbor zoradený. Pridaním hodnoty resp. jej aktualizovaním sa musia záznamy dátového súboru preusporiadať. Riešením je ponechať voľné bloky na pridanie nových hodnôt, ak sa tieto bloky použijú pridaním záznamov, potom sa vytvára spojkový zoznam overflow blokov.




Združovací index

| vhodný | nevhodný |
|---|---|
| Stĺpce sprístupňované sekvenčne | Stĺpce často sa meniace |
| Stĺpce sprístupňované klauzulou ORDER BY alebo GROUP BY | Používanie veľmi dlhých prehľadávacích kľúčov |
| Dotazy vracajúce veľké objemy dát | |

## Nezdružovací (Non-cluster) index

Ak nie je vytváraný index združovací, tak sa nazýva **nezdružovací index**. Tento index má štruktúru úplne rovnakú ako ostatné indexy, avšak vyhľadávacie kľúče v indexovom súbore sú zotriedené podľa hodnôt stĺpca, nad ktorým je index definovaný. Hlavný rozdiel oproti združovaciemu indexu je ten, že neurčuje fyzické umiestnenie záznamov v dátovom súbore, ktorý môže byť usporiadaný podľa združovacieho indexu.

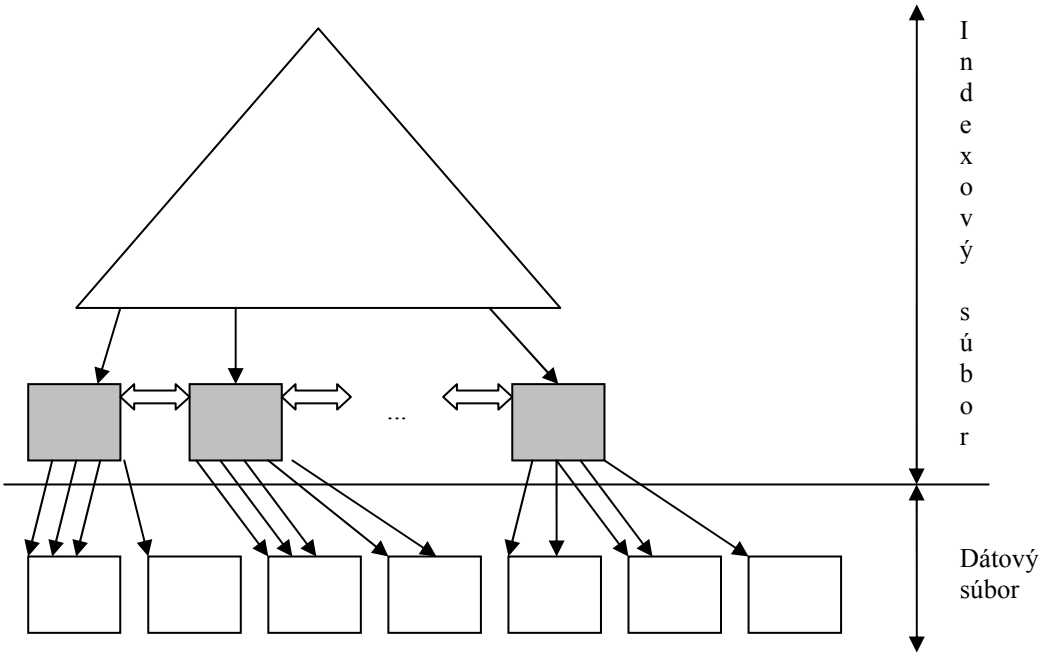
Pritom je možné vytvoriť niekoľko nezdružovacích indexov nad viacerými stĺpcami tých istých údajov. Na implementáciu sa používajú B-stromy, kde listové uzly obsahujú smerníky na záznamy v dátovom súbore. V závislosti od súborovej organizácie smerníky odkazujú priamo na riadok alebo blok, na ktorý ukazuje hodnota združeného indexu. Smerníky však môžu (väčšinou) odkazovať na rôzne bloky dát, čo zvyšuje operácie I/O, ktorých počet je určený počtom nájdených záznamov. Keďže index je zotriedený inak ako dátový súbor operácie INSERT, UPDATE a DELETE sú jednoduchšie.

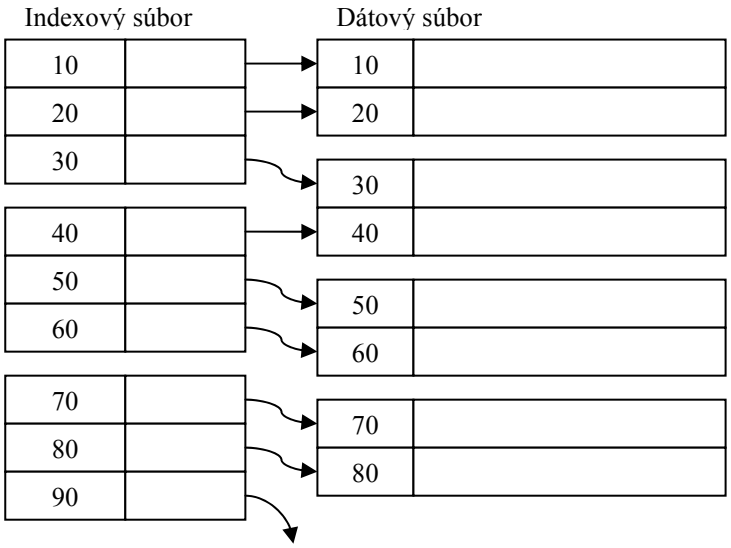
Použitie nezdružovacieho indexu

| vhodné |
|---|
| Dotazy, ktoré vracajú len malú časť dát |
| Pre stĺpce frekventovane používané v dotazoch s klauzulou WHERE |
| Pre dotazy vyžadujúce spájanie pomocou JOIN a GROUPING |

## Hustý (Dense) index

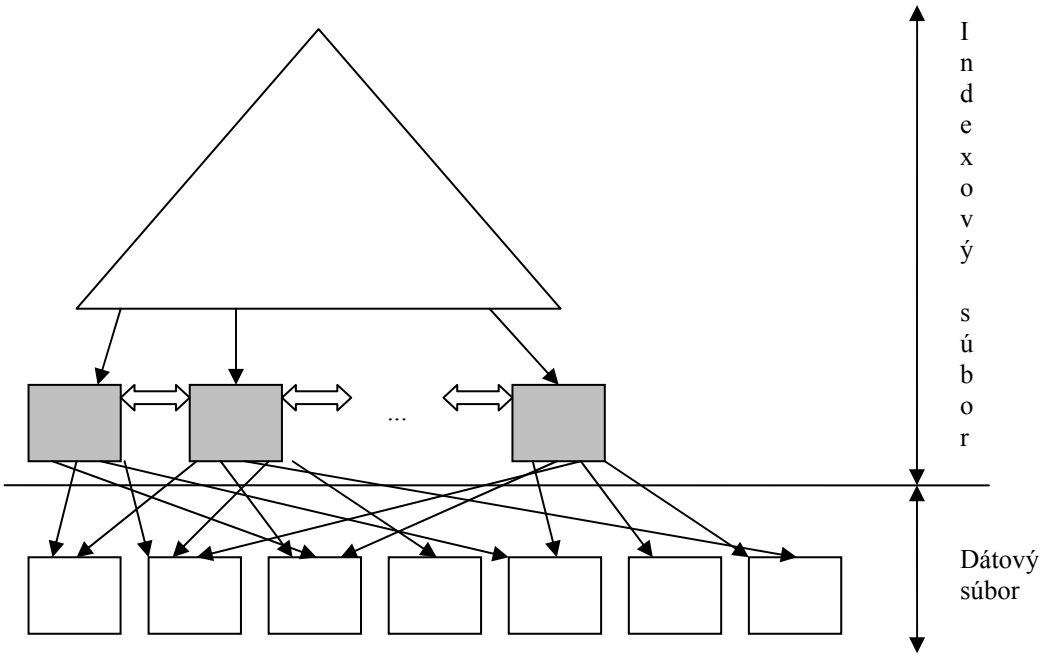
Index sa nazýva hustý, ak pre každý kľúč existuje aspoň jedna položka v indexovom súbore. Hustý index je pomerne veľký, ale na druhej strane niektoré optimalizačné techniky sú založené práve na hustom indexovaní. Keďže je indexový súbor usporiadaný, je možné nájsť záznamy veľmi rýchlo binárnym vyhľadávaním v pamäti, v ktorej sa index zvyčajne nachádza. Tabuľka môže obsahovať niekoľko rovnakých hodnôt kľúča, a preto je v indexe len jeden záznam pre daný kľúč odkazujúci sa na údajový blok.

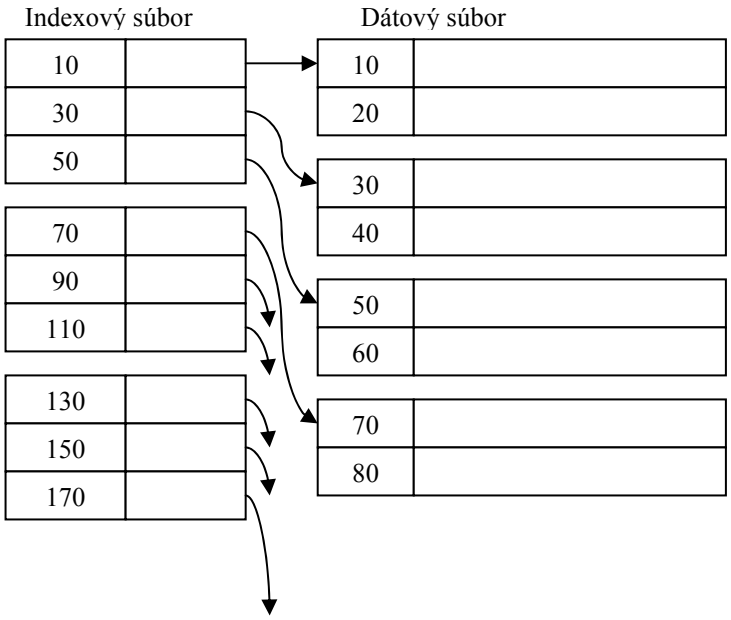



## Riedky (Sparse) index

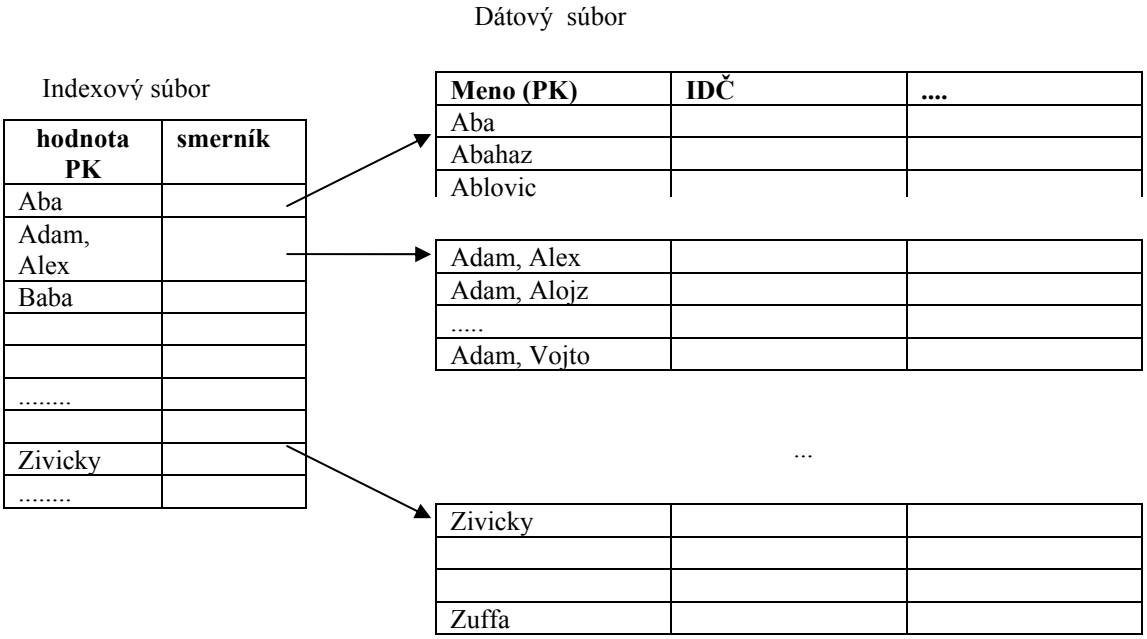
Riedky index obsahuje pre každý kľúč záznamy, ktoré sa odkazujú na dátové bloky, zvyčajné je to prvý záznam v dátovom bloku. Z toho vyplýva, že oproti hustému indexu zaberá oveľa menej priestoru, ale za cenu horšej časovej výkonnosti v pamäti. Algoritmus vyhľadávania pracuje tak, že najskôr sa binárne prehľadá riedky index, a potom znovu binárne prehľadá odkazovaný blok. Typickým príkladom riedkeho indexu môže byť cluster index . Riedky index môže byť len jeden.




## Primárny index

Zotriedený súbor záznamov, ktoré majú pevnú dĺžku a sú zadefinované nad primárnym kľúčom nazývame **primárny index**. Ide vlastne o riedky index, ktorý obsahuje toľko záznamov, koľko je v dátovom súbore blokov. Hodnota kľúča v indexovom súbore je totožná s hodnotou primárneho kľúča prvého záznamu v bloku. Prvý záznam bloku tvorí bázovú adresu, na ktorú sa smerník z indexu odkazuje.

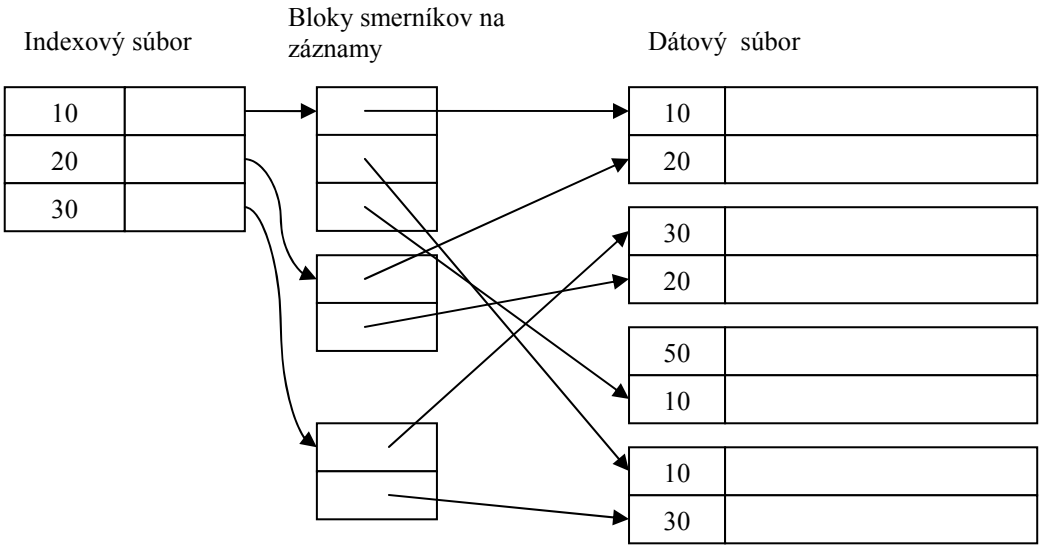



Problematické sú operácie INSERT,UPDATE a DELETE, kde pri INSERT a UPDATE operáciách sa presúvajú záznamy medzi blokmi a modifikujú sa hodnoty indexov. Tento problém sa rieši tzv. **overflow blokmi** pre ďalšie záznamy pre blok. Ak sa vytvorí priveľa overflow blokov, poklesne výkon a je nutná reorganizácia indexu. Operácia DELETE využíva označovanie zrušených záznamov

## Sekundárny index

Sekundárny index je zadefinovaný nad sekundárnym kľúčom. Pole v dátovom súbore, nad ktorým je zadefinovaný sekundárny kľuč, môže nadobúdať jedinečné alebo duplicitné hodnoty. Ak dátový súbor obsahuje iba jedinečné hodnoty, potom hovoríme, že vytvára tzv. **unique (jedinečný) index**. Sekundárny index je hustým indexom, pretože obsahuje smerník na každý záznam v dátovom súbore. Záznamy v indexovom súbore nie sú smerníkmi na záznamy, ale na bloky, kde sa záznamy nachádzajú.

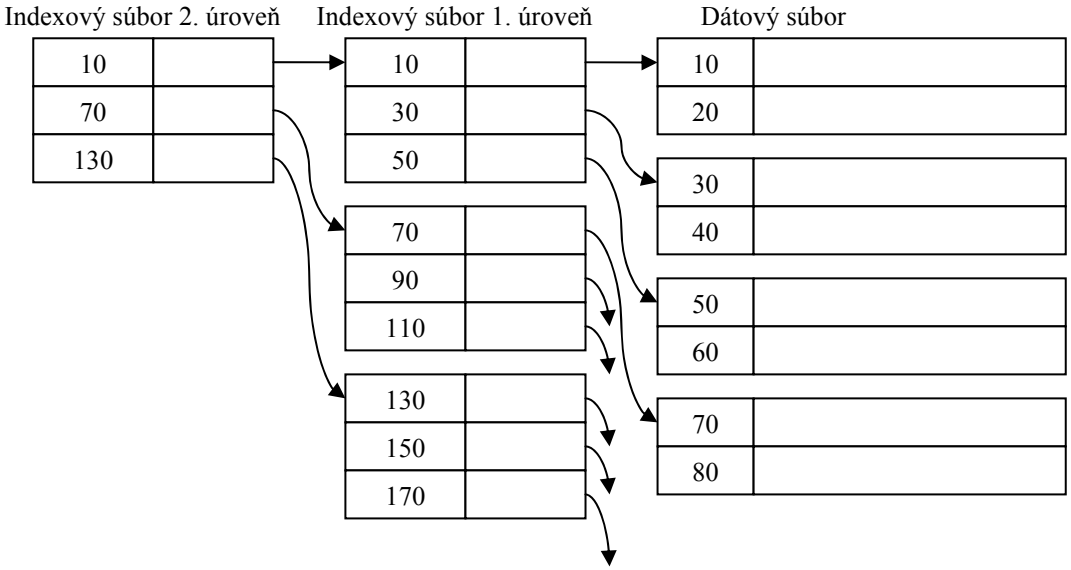
Je možné vytvoriť sekundárny index aj pre duplicitné hodnoty, kedy sa v indexovom súbore nachádzajú záznamy rôznej dĺžky. Záznam obsahuje hodnotu kľúča a zoznam smerníkov na jednotlivé záznamy v blokoch.



Použitie sekundárnych indexov je vhodné pre dotazy odpovedajúce na atribúty, ktoré netvoria primárny kľuč. Súbor môže mať viacero sekundárnych indexov. Nevýhodou sekundárnych indexov je, že sú menej efektívne ako primárne indexy.

## Viacúrovňové indexy

Myšlienka indexov spočíva v snahe, aby sa celý index nachádzal v pamäti, čím sa jeho prehľadávanie výrazne urýchli a nebude vyžadovať diskové operácie. Avšak niekedy sa stane (veľké objemy dát), že sa vytvorí index, ktorý je väčší ako pamäť a nemôže sa vtesnať celý do pamäte. Preto boli zavedene **viacúrovňové indexy**, ktoré umožňujú redukovať indexový súbor **blokovým faktorom**, a zároveň tak zrýchliť vyhľadávanie. Viacúrovňový index sa skladá z indexového súboru, na ktorý referujeme ako na **prvú úroveň**. Tento súbor je zotriedený a obsahuje jedinečnú hodnotu pre každý kľuč. Nad touto prvou úrovňou vytvoríme ďalší index, tvoriaci druhu úroveň viacúrovňového indexu. Takýmto spôsobom sa dá postupovať až kým je index celý v pamäti. Pri tvorbe každej novej úrovne sa využíva hodnota blokovacieho faktora, ktorý je pre každú úroveň rovnaký. Záznamy v indexovom súbore majú tu istú veľkosť.




Viacúrovňový index sa dá aplikovať na ľubovoľný index, pokiaľ má prvá úroveň jedinečné hodnoty a záznamy majú pevnú dĺžku. Pri viacúrovňovom indexe sú problematické operácie INSERT, UPDATE a DELETE, pretože sa pri zmenách hodnôt musí každá úroveň aktualizovať. Riešením tohto problému by bolo rezervovať priestor pre nové záznamy v každom bloku. Ide o tzv. **dynamické viacúrovňové indexovanie**, ktoré je najčastejšie implementované štruktúrou B-stromov.

## Bitmapový index

Bitmapový index, ako už názov napovedá, sa vytvára ako mapa bitov pozostávajúca z 0 a 1. Tento druh indexu je špeciálne navrhnutý pre zefektívnenie dotazov nad viacnásobnými kľúčmi t.j. zložené dotazy odpovedajúce na podmienky vo WHERE klauzule. Bitmapový index zlepšuje dotazy využívajúce boolovské výrazy (AND, OR, NOT ), kedy sa výrazy vyhodnotia do výslednej bitmapy, z ktorej sa získajú záznamy spĺňajúce podmienky v dotaze.

Štruktúra indexu je mierne odlišná od klasických indexov v tom, že pre každý kľuč index obsahuje namiesto smerníka bitmapu. V tejto bitmape každý bit reprezentuje záznam v tabuľke t.j. jeden riadok. Ak je bit v bitmape nastavený na 1, tak riadok ma v atribúte hodnotu indexového kľúča, inak je bit nastavený na 0. Na získanie záznamov sa používa **mapovacia funkcia**, ktorá podľa pozície bitu v bitmape vráti id záznamu s danou hodnotou.

| Meno | Pohlavie | Príjem |
|---|---|---|
| Jano | M | P1 |
| Jana | Z | P2 |
| Ivan | M | P1 |
| Jožo | M | P3 |
| Anna | Z | P1 |
| Tomáš | M | P3 |

Bitmapy pre pohlavie

| | | | | | | |
|---|---|---|---|---|---|---|
| M | 1 | 0 | 1 | 1 | 0 | 1 |
| Z | 0 | 1 | 0 | 0 | 1 | 0 |

Bitmapy pre príjem

| | | | | | | |
|---|---|---|---|---|---|---|
| P1 | 1 | 0 | 1 | 0 | 1 | 0 |
| P2 | 0 | 1 | 0 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 1 | 0 | 1 |

SELECT COUNT(*) FROM CUSTOMER
WHERE pohlavie = 'M' AND príjem = 'P1'

| Pohlavie = M | | Príjem = P1 | | |
|---|---|---|---|---|
| 1 | | 1 | | 1 |
| 0 | AN | 0 | | 0 |
| 1 | D | 1 | = | 1 |
| 1 | | 0 | | 0 |
| 0 | | 1 | | 0 |
| 1 | | 0 | | 0 |

Vybrané riadky budú 1. a 3.

Výhodou použitia bitmapového indexu je efektívnejšie využitie miesta, ak mame veľmi veľa rovnakých hodnôt. Tak napríklad, ak máme záznam o veľkosti 100 bytov, potom jedna bitmapa je 1/800 z celkovej veľkosti záznamu. Ďalšou výhodou je to, že výpočtový systém je založený na binárnych výpočtoch hodnôt, a preto je možné pri správnej implementácii s využitím inštrukcii CPU (tvoria ich slova, z bitových máp) ešte viac zvýšiť výkon výpočtu.

### Funkčný (Function-based) index

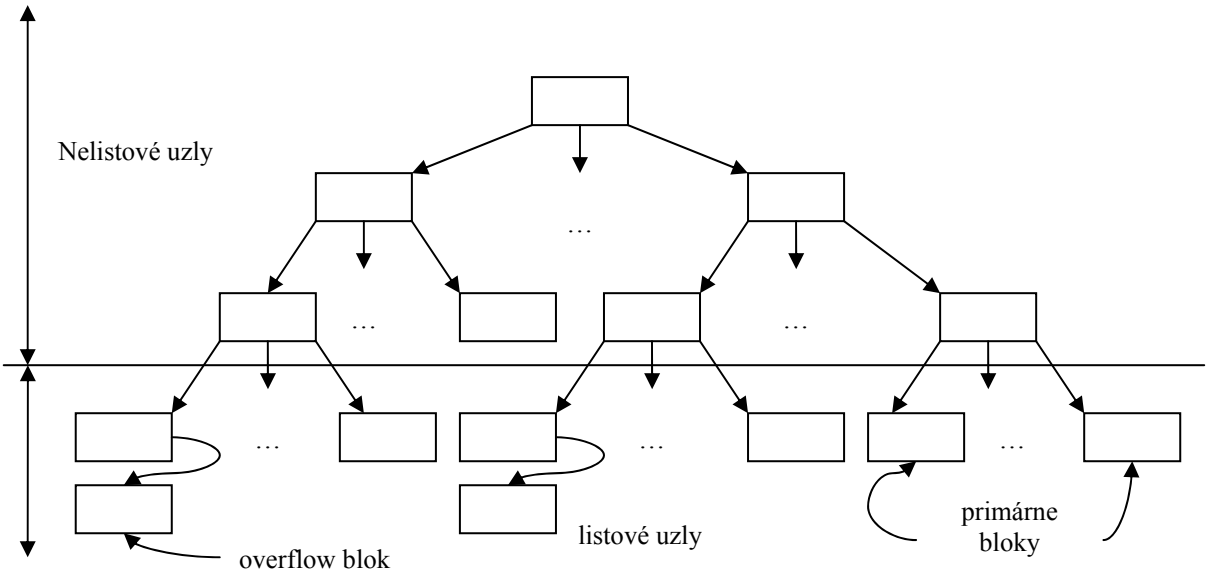
Funkčné indexy poskytujú efektívny mechanizmus na spracovanie dotazov obsahujúcich funkciu v ich WHERE klauzule. Vytvorenie takéhoto indexu je vhodné ako predspracovanie náročnejších výpočtov, čím sa zvýši výkonnosť pri operáciách SELECT, v ktorých sa následne použijú hodnoty indexu. Výpočet hodnôt sa deje pri operáciách INSERT a UPDATE. Vytvoriť funkčný index je možné nad hodnotami z viacerých stĺpcov a na implementáciu použiť B-stromy alebo bitmapový index. Funkcie použiteľné na vytvorenie indexu môžu byť aritmetické výrazy, SQL funkcie, funkcie balíka, C volania a výrazy obsahujúce SQL funkcie pričom musia byť deterministické.

## Stromová štruktúra indexov

Stromové štruktúry poskytujú efektívne vyhľadávanie v rozsahu určených hodnôt, rovné hodnoty, a taktiež umožňujú efektívne operácie INSERT a DELETE . V podstate je možné použiť buď statickú (ISAM1 ) alebo dynamickú štruktúru (B-stromy a B+-stormy).

### Indexed Sequential Access Method (ISAM)

Myšlienka ISAM je založená na vytváraní viacúrovňového indexu rekurzívne, tak aby sa celý index zmestil do jedného bloku. To umožňuje vytvorenie stromovej štruktúry ilustrovanej na obrázku.




Uzly stromu sú v podstate diskové bloky, ktoré obsahujú odkazy na ďalšie bloky až k listovým uzlom. Smerníky na záznamy v dátovom súbore sú v **listových uzloch** resp. **blokoch**. V prípade, že sa listové bloky naplnia pridávaním hodnôt vytvoria sa ďalšie bloky tzv. **overflow bloky**, ktoré sú spojenie s niektorým z listových blokov. ISAM štruktúra je kompletne statická, pretože pri vytváraní indexového súboru sú všetky uzly vrátane listových uzlov alokované a usporiadané podľa vyhľadávacieho kľúča okrem overflow blokov. Tieto bloky môžu vznikať a zanikať dynamicky, ak sa vyžaduje vloženie nových hodnôt. Požiadavka je, aby overflow blokov bolo čo možno najmenej.

Pri vyhľadávaní hodnôt sa používa algoritmus identický pre B+-stromy, kde sa využíva prehľadávanie od koreňa a na základe porovnania hodnoty v uzle sa rozhodne, ktorý substrom sa ma použiť ďalej až do lištového uzla. V listovom uzle sa hodnota odkaze na blok v dátovom súbore a hodnoty sú získané sekvenčne. Pre vkladanie a mazanie hodnôt z blokov je podobne ako pri vyhľadávaní, čím sa zisti listový uzol kam sa má hodnota zapísať alebo odkiaľ sa má zmazať. Ak pri mazaní hodnôt nastane prípad, že listový uzol zostane prázdny, tak sa nezmaže, ale ostáva prázdny až do jeho naplnenia.

Počet I/O prístupov je rovný výške stromu pri zložitosti log F N, kde N je počet primárnych listových uzlov a F je počet potomkov uzla. Toto číslo I/O je omnoho menšie než pri binárnom vyhľadávaní.

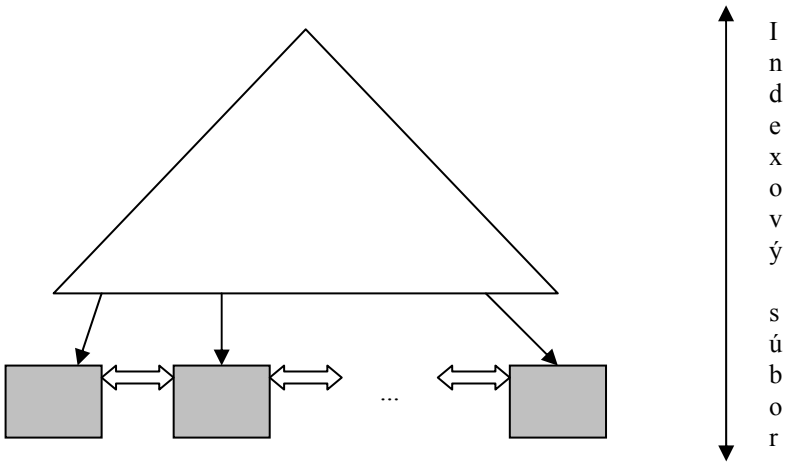
Nevýhodou je vytvorenie veľkého množstva overflow blokov, riešením je alokovanie bloku tak, aby bolo možné pridávať aj ďalšie hodnoty pripadne reorganizácia stromu. Výhodou oproti B-stromov je zamykanie len listového uzla, ktorý sa vyžaduje.

## B-stromy

B – strom je špeciálnym prípadom vyváženého n-árneho orientovaného koreňového stromu. Je zovšeobecnením 2-3 stromov . Každý nelistový vrchol (okrem koreňa) má počet synov v intervale <n/2, n>. Koreň B – stromu je buď koreňom alebo má aspoň 2 synov Všetky listy majú rovnakú hĺbku.

## $B^+$-stromy

Statická štruktúra ISAM má nevýhodu vo vytváraní dlhých overflow blokov, čo má za následok zhoršenie výkonnosti. Tento nedostatok motivoval k vytvoreniu dynamickej štruktúry, ktorá by zvládala prispôsobiť sa zmenám pri mazaní a pridávaní záznamov. **B+-strom** je často používaný vyvážený strom, ktorého vnútorné uzly sú používané na vyhľadávanie listu v strome a lištové uzly obsahujúce záznamy do dátového súboru. Keďže je táto štruktúra dynamická, nie je možné alokovať listové bloky sekvenčne, ako tomu bolo v ISAM. Na efektívne získanie všetkých možných listových blokov sa musia tieto bloky navzájom poprepájať pomocou smerníkov. Listy sú poprepájaná obojsmerným zoznamom, preto môžeme prechádzať listami v oboch smeroch.




Pre implementáciu bude potrebná dodatočná informácia (listový, interný uzol, počet položiek v uzle, odkazy na rodičovské a susedné uzly atď.).

Operácie nad B+-stromami (SEARCH, INSERT, DELETE) sú vykonávané podobne ako pre B-stromy s tým rozdielom, že položka sa vždy zaraďuje do (vyraďuje z) listového uzla. Potom sa prípadne modifikujú interné uzly.

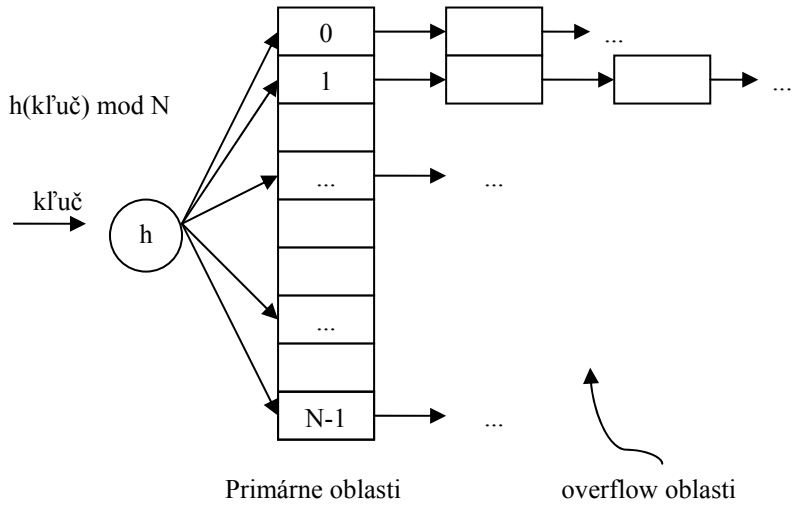
## Hash Indexy

Indexácia založená na hašovaní je súborová organizácia vhodná najmä na dotazovanie presných hodnôt. Podstata spočíva v hašovacej funkcii, ktorá mapuje hodnoty kľúčov do oblastí zvaných (buckets), tieto oblasti obsahujú jeden alebo niekoľko záznamov resp. adresy na záznamy .

Funkcia sa používa na získanie záznamov pre operácie SELECT, INSERT, DELETE, pričom pre rôzne kľúče sa môže stať, že sa budú odkazovať do tej istej oblasti. Preto je potrebné po získaní adresy jednej oblasti prehľadať túto oblasť sekvenčne na získanie záznamu. Funkcia môže byť dvojakého typu

- náhodná
- rovnomerná

Ideálny prípad je vtedy, ak funkcia je zvolená tak, aby rovnomerne rozdeľovala záznamy do jednotlivých oblastí. Ak tomu tak nie je, môže nastať situácia, že sa oblasti, kam sa záznamy umiestňujú naplnia a je preto potrebne vytvoriť overflow oblasti. Overflow oblasti sú pospájané ako spojkový zoznam.



### Hash indexy nedostatky

Ako každý index aj hash indexy majú niektoré nedostatky, ktoré môžu prekážať pri ich zavedení.

- Hash indexy nepodporujú intervalové vyhľadávanie.
- Hoci podporuje mnohonásobné kľúče, nepodporuje vyhľadávanie na základe čiastočných kľúčov.
- Dynamické vzrastanie veľkosti hash indexového súboru produkuje vyplňujúce stránky.

### Rozšírené hašovanie

Pri rozšírenom hašovaní Interval hašovacej funkcie je rozšírený, aby vyhovoval pridávaním oblastí.

V akomkoľvek danom čase máme unikátnu hash funkciu. Nevýhodou je potrebný extra priestor pre adresár. Ak sa adresár nevmesti do pamäti, potom musíme preniesť ďalšiu stránku do pamäte.

## Paralelizmy v databázových systémoch

Transakcia musí začať a môže prebehnúť úspešne alebo neúspešne. O neúspešných transakciách sa hovorí ako o zničených, o úspešných ako o potvrdených.
Transakcia má nasledovné vlastnosti:

- realizáciou sa dostáva BD z konzistentného stavu do konzistentného stavu
- transakcia je atomická – nemôže prebehnúť len časť: buď prebehne celá alebo po nej v databáze nezostanú žiadne zmeny
- transakciou sa môžu meniť hodnoty objektov v databáze, ale počas vykonávania transakcie nesmú byť menené hodnoty viditeľné pre iné transakcie.

V prípade poruchy je potrebné, aby sa obnovili transakcie, ktoré boli potvrdené pred poruchou a aby nezostali v báze dát následky po žiadnej zničenej transakcii

**Transakcia je definovaná ako každé jedno vykonanie programu používateľského programu v SRBD** a líši sa od vykonania programu mimo SRBD. Na zvýšenie výkonu musí SRBD zabezpečiť vykonávanie viacerých transakcií naraz. Avšak aby nedošlo k nežiaducim výsledkom je potrebná kontrola tohto súbežného vykonávania. Taktiež je potrebné zabezpečiť jednu z najdôležitejších vlastností transakcií a to že sa vykoná buď celá , od začiatku až po koniec, alebo sa naopak nevykoná vôbec. Ak nastane napríklad výpadok systému počas vykonávania transakcie, SRBD musí zabezpečiť vrátenie databázového systému späť do stavu v akom sa nachádzal pred začiatkom vykonávanie danej transakcie. Táto oblasť sa nazýva „crash recovery“.
Používateľ píše programy na prístup/modifikáciu údajov pomocou niektorého vyššieho dotazovacieho jazyka, ktorý daný SRBD podporuje. Na porozumenie toho, ako SRBD zvláda vykonanie uvedeného programu, si musíme uvedomiť, že transakcia je vlastne postupnosť čítania a zapisovania databázových objektov.

- **Čítanie objektu** – najprv je daný objekt prenesený do hlavnej pamäte z disku a potom je jeho hodnota zapísaná do premennej programu
- **Zápis objektu** – objekt je najprv modifikovaný v pamäti a až potom je zapísaný na disk.

Databázové objekty sú jednotky, z ktorých programy čítajú resp., do ktorých zapisujú informácie. Jednotky môžu byť stránky, záznamy a podobne. Transakcia je videná SRBD ako séria , alebo zoznam akcií. Akcie, ktoré môžu byť transakciou vykonané, sú zápis a čítanie databázových objektov. Transakcia môže byť definovaná ako súbor týchto akcií, ktoré sú čiastočne zoradené. Naviac, k týmto dvom akciám (**read** a **write**) je potrebné, aby mala každá transakcia definované ešte dve akcie. Sú to akcia **commit** (úspešné ukončenie vykonávania transakcie ) a **abort alebo rollback** (ukončenie vykonávania transakcie a vrátenie všetkých zmien prevedených touto transakciou do pôvodnej podoby).Transakcie majú niekoľko základných vlastností, označovaných anglickou skratkou ACID:

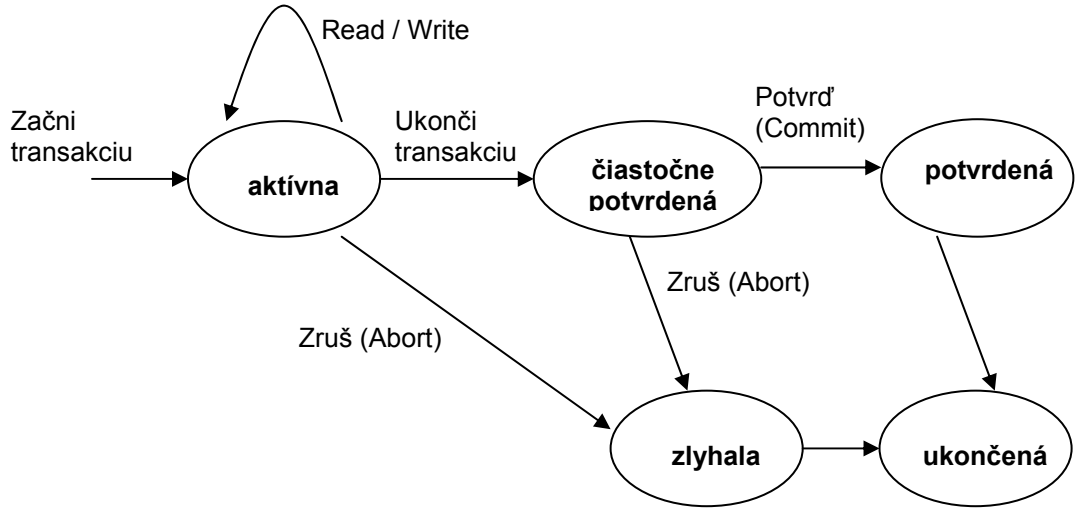
**1. Atomičnosť (atomicity)** – nedeliteľnosť transakcie. Znamená to že transakcia je vykonaná celá alebo sa nevykoná vôbec. Transakcia môže byť neukončená kvôli trom dôvodom.
Po prvé, transakcia môže byť ukončená ešte pred jej celým vykonaním, kvôli výskytu abnormalít

počas vykonávania. Ak je transakcia prerušená SRBD kvôli interným problémom počas vykonávania, je následne opäť spustená.
Po druhé, úspešné ukončenie transakcie nenastane kvôli pádu systému (napríklad vypadne prívod elektrickej energie a podobne).
Posledným , tretím dôvodom, prečo sa nevykoná celá transakcia je, že nastanú nepredvídané situácie (ako napríklad prebehne čítanie hodnoty, ktorá nebola predpokladaná, alebo je problém s prístupom na disk a podobne), a sama transakcia sa rozhodne ukončiť priebeh svojho vykonávania.
Samotné zabezpečenie atomičnosti sa uskutočňuje spätným zrušením akcií, ktoré boli realizované. To znamená, že po prevedení všetkých zmien späť bude databáza opäť v stave ako bola pred začatím vykonania danej transakcie. Na to, aby bolo možné uskutočnené zmeny vrátiť späť do pôvodného stavu, je potrebné, aby si SRBD viedol záznam o prevádzaných zmenách, ktorý sa nazýva **log**.
**2. Konzistentnosť (consistency) –** každá transakcia musí ponechať databázu konzistentnou. Používateľ je zodpovedný za konzistentnosť databázy. Nemôžeme očakávať, že SRBD bude schopný detekovať nekonzistentný stav, ktorý bude spôsobený v programe používateľa.
**3. Nezávislosť (isolation)–** činnosť transakcie je neovplyvniteľná inou transakciou Nezávislosť je zabezpečená garanciou, že aj keď sa viacero transakcií vykonáva súčasne, vždy je vytvorený **rozvrh,** v ktorom je presne povedané ako sa budú **sériovo** vykonávať jedna transakcia za druhou tak, aby sa navzájom neovplyvňovali.
**4. Trvanlivosť (durability)–** výsledky transakcie pretrvávajú v databáze aj po jej skončení. Ak skončila transakcia úspešne, musí byť jej efekt zapísaný na disk aj keby došlo k pádu systému. Na zabezpečenie tejto vlastnosti sa taktiež používa log. Ak systém spadne ešte pred tým ako sa prevedené zmeny uložia na disk, na vrátenie menených údajov, ktoré je potrebné uložiť na disk, slúži práve tento záznam zmien. Nezapísané zmeny databázy, sú zapísané hneď po reštarte systému.

**Stavový diagram transakcií**
Každá transakcia sa môže nachádzať v rôznych stavoch.

1. Transakcia sa dostáva do stavu **aktívna** pri začatí vykonávania prvého príkazu v bloku transakcie.
2. Po skončení vykonávania transakčných operácií prechádza do stavu **čiastočne potvrdená**
3. V stave **čiastočne potvrdená** sa aplikujú algoritmy obnovy (recovery algorithms), ktoré majú spoľahlivo overiť, že výsledok transakcie bude trvale zaznamenaný.
4. Ak je výsledok testov úspešný transakcia prechádza do stavu **potvrdená**. V tomto stave sa transacia považuje za úspešne vykonanú, takže všetky zmeny musia byť trvalo zaznamenané. Potom transakcia prechádza do stavu **ukončená**.
5. Ak zlyhá niektorý z testov pre úspešné, trvalé uloženie zmien, alebo je transakcia zrušená v stave **aktívna**, prejde do stavu **zlyhala**. Potom môže byť aplikované odvolanie (rollback) transakcie, aby sa systém dostal do stavu pred začatím transakcie.
6. Stav **ukončená** zodpovedá stavu kedy transakcia opúšťa systém.

**Súbežné transakcie**




Obr: Stavový diagram transakcií

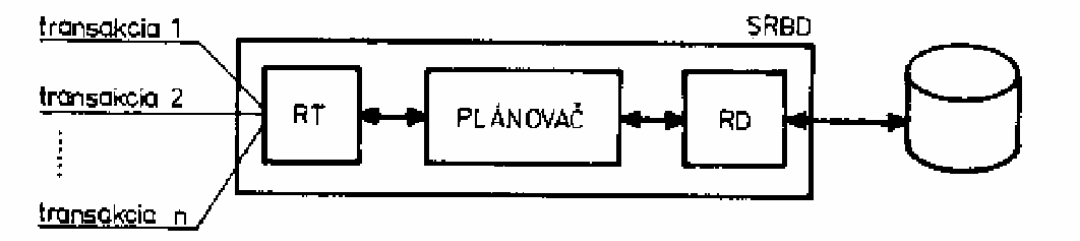
SRBD prekladá akcie rôznych transakcií na zvýšenie výkonu, na zvýšenie priepustnosti, a zníženie času odozvy. Avšak nie je možné prekladať transakcie stále a len tak ako sa nám páči, je potrebné dodržať určité pravidlá. Nech sú dané dve transakcie T1 a T2. Ich vzájomná postupnosť môže byť napr. nasledovná:

| $T_1$ | $T_2$ |
|---|---|
| $A := READ(X)$ | |
| | $A := READ(X)$ |
| $A := A + 5$ | |
| $WRITE(X,A)$ | |
| | $A := A + 5$ |
| | $WRITE(X,A)$ |

| $T_1$ | $T_2$ |
|---|---|
| $A := READ(X)$ | |
| $A := A + 5$ | |
| $WRITE(X,A)$ | |
| | $A := READ(X)$ |
| | $A := A + 5$ |
| | $WRITE(X,A)$ |

| $T_1$ | $T_2$ |
|---|---|
| | $A := READ(X)$ |
| | $A := A + 5$ |
| $A := READ(X)$ | |
| | $WRITE(X,A)$ |
| $A := A + 5$ | |
| $WRITE(X,A)$ | |

Porovnajme zapísané výsledky transakcií po ich skončení. Výsledky napr. pre 5,10 nie sú pri všetkých troch rovnaké. Pri procese v databáze závisí na postupnosti, akou sa jednotlivé operácie vykonávajú. Činnosť databázy popisuje nasledovná schéma:




Pre vykonávanie súbežných transakcií sú dôležité nasledovné časti:

- **RT - modul riadenia transakcií**: obracajú sa sem transakcie s požiadavkou čítania a zápisu
- **RD - modul riadenia dát**: vykonáva realizáciu zápisu a čítania podľa pokynov riadiaceho modulu a posúva plánovačovi správu o prečítaní,
- **plánovač**: zabezpečuje synchronizáciu požiadaviek viacerých transakcií a zaraďuje požiadavky do rozvrhu
- **rozvrh**: poradie, v akom sa transakcie realizujú.

Ak by sme nedovolili paralelné spracúvanie transakcií, zostali by sme pri sériovom spracovaní – transakcie by išli za sebou.

**Rozvrh**

Rozvrh je zoznam akcií ( reading, writing, aborting, committing) zo súboru transakcií, pričom poradie dvoch akcií z jednej transakcie T v rozvrhu musia byť v rovnakom poradí ako sú v danej transakcii. Z toho je jasné, že rozvrh charakterizuje potenciálne poradie vykonanie jednotlivých akcií. Rozvrh, ktorý obsahuje buď abort alebo commit pre každú transakciu, ktorých akcie sa nachádzajú v danom rozvrhu, sa nazýva **kompletný**. Kompletný rozvrh musí obsahovať všetky akcie všetkých transakcií, ktoré sa nachádzajú v danom rozvrhu. Ak sa akcie rôznych transakcií neprekrývajú, tak takýto rozvrh nazývame **sériový**. **Sériový rozvrh** je také poradie vykonania operácií, že sa vykonajú bezprostredne za sebou.. Popri zaistení integrity je však ďalšou dôležitou požiadavkou maximálna paralelizácia transakcií. Ak rozvrh neobsahuje abort a ani commit pri jednotlivých transakciách, znamená to, že rozvrh nie je kompletný.
**Sériovateľný rozvrh** je taký, ktorý dosiahne rovnaký výsledok, ako niektorý zo sériových rozvrhov. Serializovateľný rozvrh nad súborom S transakcií je rozvrh, ktorého efekt na hocijakú konzistentnú databázu je garantovane identický s dopadom nejakého kompletného sériového rozvrhu nad tým istým súborom S. Dôležitým faktom je skutočnosť, že vykonávanie transakcií za sebou v rôznych poradiach môže priniesť rôzne výsledky, ale všetky by mali byť akceptovateľné. SRBD nezaručuje, ktorý z možných výsledkov sa dostaví.
**Obnoviteľný rozvrh** je rozvrh, v ktorom sú transakcie potvrdené až potom ako všetky transakcie, ktorých zmeny boli čítané transakciami, boli potvrdené.

## Plánovače

Dva hlavné typy plánovačov sú:

- **zamykanie**
- **časové pečiatky**

**Zamykanie**
Realizujme nasledovné rozvrhy pre T1 a T2:

Budeme vykonávať transakciu z príkladu s týmito hodnotami
a) $X = 50, \quad Y = UA, \quad Z = UB,$
b) $X = 100, \quad Y = UB, \quad Z = UA.$

$T_1$ označme vykonanie transakcie s hodnotami z bodu a), $T_2$ s hodnotami z bodu b). $A$ je lokálna premenná v rámci vykonania transakcie. Znázornime tri možné rozvrhy transakcií $T_1$ a $T_2$:

Rozvrh I

| $T_1$ | $T_2$ |
|---|---|
| $A$ := READ($UA$) | |
| $A$ := $A$ − 50 | |
| WRITE($UA$, $A$) | |
| $B$ := READ($UB$) | |
| ... | |
| $B$ := $B$ + 50 | |
| WRITE($UB$, $B$) | |
| | $B$ := READ($UB$) |
| | $B$ := $B$ − 100 |
| | WRITE($UB$, $B$) |
| | $A$ := READ($UA$) |
| | $A$ := $A$ + 100 |
| | WRITE($UA$, $A$) |

Rozvrh II

| $T_1$ | $T_2$ |
|---|---|
| $A$ := READ($UA$) | |
| $A$ := $A$ − 50 | |
| | $A$ := READ($UB$) |
| WRITE($UA$, $A$) | |
| | $A$ := $A$ − 100 |
| | WRITE($UB$, $A$) |
| | $A$ := READ($UA$) |
| $A$ := READ($UB$) | |
| $A$ := $A$ + 50 | |
| | $A$ := $A$ + 100 |
| WRITE($UB$, $A$) | WRITE($UA$, $A$) |

Rozvrh III

| $T_1$ | $T_2$ |
|---|---|
| $A$ := READ($UA$) | |
| $A$ := $A$ − 50 | |
| | $A$ := READ($UB$) |
| | $A$ := $A$ − 100 |
| WRITE($UA$, $A$) | |
| $A$ := READ($UB$) | |
| | WRITE($UB$, $A$) |
| | $A$ := READ($A$) |
| $A$ := $A$ + 50 | |
| WRITE($UB$, $A$) | |
| | $A$ := $A$ + 100 |
| | WRITE($UA$, $A$) |

Súčet stavov účtov $UA$ a $UB$ po vykonaní transakcií $T_1$ a $T_2$:

podľa rozvrhu I : $UA + UB$,
podľa rozvrhu II : $UA + UB$,
podľa rozvrhu III : $UA + UB + 100$.

Typickým problémom je zmena prečítanej hodnoty počas jej spracúvania inou transakciou. Viď rozvrh III.
Na riešenie sa používa **zamykanie**. Pokiaľ je objekt uzamknutý pre transakciu T, nemá k nemu prístup žiadna iná transakcia, až do chvíle, kým sa neodomkne.
Predpokladajme teda, že transakcia je naprogramovaná tak, že:

- pred operáciami READ/WRITE objekty zamkne
- nepokúsi sa zamknúť objekt, ktorý už zamkla
- pred ukončením všetky objekty odomkne.

```
LOCK (X);
A : = READ (X);
A : = A + 5;
WRITE (X, A);
UNLOCK (X);
```

Ak dôjde k operácii nad zamknutým objektom, transakcia vycúva, alebo čaká.

Zamykanie však sériovateľnosť transakcií nezaručí.

Majme rozvrhy:

Rozvrh I

| $T_1$ | $T_2$ |
|---|---|
| LOCK($A$) | |
| LOCK($B$) | |
| $B := A + B$ | |
| $A := A - B$ | |
| UNLOCK($B$) | |
| | LOCK($B$) |
| UNLOCK($A$) | |
| | $B := 2 * B$ |
| | UNLOCK($B$) |
| | LOCK($A$) |
| | $A := A + 1$ |
| | UNLOCK($A$) |

Rozvrh II

| $T_1$ | $T_2$ |
|---|---|
| LOCK($A$) | |
| | LOCK($B$) |
| | $B := 2 * B$ |
| | UNLOCK($B$) |
| LOCK($B$) | |
| $B := A + B$ | |
| $A := A - B$ | |
| UNLOCK($A$) | |
| UNLOCK($B$) | |
| | LOCK($A$) |
| | $A := A + 1$ |
| | UNLOCK($A$) |

Rozvrh II. nie je sériovateľný.

**Sériovateľnosť** rozvrhov je zaručená až pri dvojfázovom zamykaní:

- objekt môže byť v každom čase zamknutý len pre jednu transakciu
- ak transakcia odomkla aspoň jeden objekt, nesmie už žiaden ďalší zamknúť

realizácia zvyčajne prebieha tak, že transakcie najprv objekty podľa potreby zamykajú, a potom (zvyčajne ku koncu) odomykajú.
Transakcie zamykajú a odomykajú objekty dvojfázovo. V prvej fáze zamykajú objekty bez toho, aby čo i len jeden odomkli, a v druhej fáze, v ktorej žiaden objekt nesmú zamknúť, objekty postupne odomykajú. Dvojfázové zamykanie sa často realizuje tak, že transakcie odomknú zamknuté objekty až tesne pred ukončením. Postup, systém, zamykania a odomykania objektov v rámci transakcie je daný v **protokole zamykania**.

**Uviaznutie**
Dvojfázové zamykanie je riešením problému sérializovateľnosti. Zamykanie však môže viesť k inému problému, dobre známemu z oblasti operačných systémov, a to **uviaznutiu** (**deadlock**).
Majme opäť dvojicu transakcií a ich rozvrh:

| | $T_1$ | $T_2$ |
|---|---|---|
| 1 | LOCK($A$) | |
| 2 | | LOCK($B$) |
| 3 | READ($A$) | |
| 4 | | READ($B$) |
| 5 | LOCK($B$) | |
| 6 | | LOCK($A$) |
| 7 | READ($B$) | |
| 8 | UNLOCK($A$) | |
| 9 | UNLOCK($B$) | |
| 10 | | READ($A$) |
| 11 | | UNLOCK($A$) |
| 12 | | UNLOCK($B$) |

Plánovač sledujúci zamykanie objektov nemôže vyhovieť požiadavke transakcie $T_1$(zamknúť $B$) v 5. kroku rozvrhu, pretože $B$ je zamknutý transakciou $T_2$. Ďalej v 6. kroku rozvrhu nemôže plánovač z opačných dôvodov zamknúť objekt $A$ pre transakciu $T_2$. Došlo k uviaznutiu a obe transakcie sa zastavili.

Uviaznutie sa rieši zvyčajne:
1. Často používaná metóda - periodické sledovanie systému a v prípade zistenia uviaznutia zrušenie jednej požiadavky, najčastejšie zničením príslušnej transakcie.

Uviaznutie možno detekovať analýzou grafu čakania, ktorého vrcholy predstavujú transakcie a orientované hrany vzťahy čakania: T1 - T2 znamená, že transakcia T1 čaká na uvoľnenie objektu zamknutého transakciou T2. K uviaznutiu došlo, ak v grafe čakania existuje cyklus. V prípade existencie cyklu možno jednu z transakcií zničiť.
2. V systémoch, kde je možný vyšší stupeň koordinácie, sa dá uviaznutiu predchádzať, Existuje na to viacero techník. napr. k uviaznutiu nedôjde, ak transakcie zamykajú objekty v poradí, rešpektujúcom nejaké lineárne usporiadanie definované nad týmito objektmi.

Ak by toto usporiadanie v našom príklade bolo dané napr. abecedným usporiadaním nad menami objektov a transakcie T1 a T2 by objekty zamykali v súlade s týmto usporiadaním, k uviaznutiu by nedošlo.
Z hľadiska zvýšenia počtu súbežne vykonávaných transakcií je výhodné rozlišovať medzi zamknutím pre čítanie a zamknutím pre zmenu hodnoty (zápis). Plánovač môže povoliť zamknutie jedného objektu viacerými transakciami pre čítanie a pozdržať žiadosť o zamknutie pre zápis dokiaľ ho všetky transakcie, ktoré mali objekt zamknutý neuvoľnia. Zamykanie pre zápis a jeho realizácia sa nelíšia od jednotného zamykania, o ktorom sme hovorili : ak je objekt zamknutý pre nejakú transakciu pre zápis, nemôže byť zamknutý pre ďalšiu transakciu.
K uviaznutiu nedôjde, ak sa objekty zamykajú v súlade s určitým usporiadaním – napr. podľa abecedy.

**Záverom možno zhrnúť úlohy plánovača nasledovne:**

- riadenie zamykania objektov
- operácie čítania a zmeny povoľovať len transakciám, ktoré majú príslušné objekty zamknuté
- sledovať dodržiavanie dvojfázového zamykania, pri zistení porušenia narúšajúcu transakciu zničiť
- predchádzať uviaznutiu, prípadne ho detekovať

**Časové pečiatky (Timestamp)**
Časová pečiatka (ČP) je druhý spôsob riadenia súbežných transakcií. Časová pečiatka je číslo priradené transakcii alebo objektu bázy dát. Pridelené ČP tvoria rastúcu postupnosť funkciou času. ČP prideľuje transakcii modul riadenia transakcií a je platná pre všetky operácie, používa plánovač, aby pomocou nich riadil vykonávanie konfliktných operácií čítania a zápisu. O operáciách hovoríme, že sú konfliktné, ak sa obe týkajú toho istého objektu bázy dát a aspoň jedna z nich je operácia **WRITE.** Princíp práce plánovača založeného na ČP možno charakterizovať takto: plánovače na báze ČP vytvárajú len sérializovateľné rozvrhy. Existujú viaceré varianty plánovačov pracujúcich s ČP.

**Základný plánovač s využitím časových pečiatok**
Registruje pre každý objekt bázy dát najväčšiu ČP, ktorú mala operácia **READ** čítajúca hodnotu tohto objektu najväčšiu ČP operácie **WRITE.**

Keď plánovač dostane požiadavku s nejakým ČP na prečítanie hodnoty objektu **x**, vykoná takúto činnosť:

- ak ČP < W/CP (X), odmietne požiadavku a zničí transakciu, ktorá ju vyslala,
- v opačnom príp. vyhovie požiadavke a aktualizuje R/CP (x):= max (čp, R/CP (x))

Keď plánovač dostane požiadavku **WRITE** (x) s ČP :

- ak ČP < W/CP (x) OR R/CP (x), odmietne požiadavku a zničí transakciu, ktorá ju vyslala
- v opačnom príp. realizuje požiadavku a aktualizuje W/CP (x):= čp

Zničené transakcie sa spustia znovu s novou (vyššou) ČP. Tento základný plánovač môže spôsobovať časté zničenie transakcií. Jeho modifikáciou je konzervatívny plánovač, ktorý sa snaží vyhnúť odmietnutiam požiadaviek za cenu pozdržania vykonania operácií. Požiadavky z RT ukladá do fondu a vyberá z neho požiadavky s najmenšou ČP. To vedie k zníženiu počtu odmietnutí, za cenu zdržania realizácie transakcií.

**Thomasov plánovač**
Umožňuje znížiť počet odmietnutí pri operáciách **WRITE.** Vychádza zo skutočnosti, že keď má operácia **WRITE** nižšiu ČP ako už realizovaná operácia **WRITE** nad tým istým objektom a novú hodnotu ešte nikto neprečítal, môžeme ju ignorovať, pretože by priradila objektu bázy dát medzitým zastaralú hodnotu.
**Thomasov plánovač** realizuje operácie **READ** rovnako ako základný plánovač.

WRITE (x) s CP = čp spracuje plánovač takto:

- ak ČP < R/CP (x) požiadavku odmietne a zničí transakciu,
- ak ČP < W/CP (x) predstiera pred RT, že požiadavku úspešne realizoval, ale do RD nepošle príslušný príkaz pre zápis,
- akceptuje **WRITE** a aktualizuje W/CP (x):= čp.

**Thomasov plánovač** znižuje pravdepodobnosť odmietnutia transakcií.

## Metódy ochrany transakcií

Základným prostriedkom ochrany je zaznamenávanie zmien do žurnálového súboru. Vďaka nim potom možno transakciu zopakovať alebo z nej vycúvať.
Operácie prebiehajúce v transakciách možno rozdeliť na:

- nechránené operácie (v prípade zničenia transakcie nemusia byť zopakované, resp. obnovené): čítanie, diagnostické správy...
- chránené operácie (v prípade zničenia transakcie alebo rekonštrukcie potrebujú byť vykonané, resp. stornované)
- reálne operácie (následky sa nedajú pri zotavení odstrániť): napr. vyplatenie peňazí zákazníkovi v banke...

Následky potvrdených transakcií by sa nemali vykonávať zotavovaní od kontrolného bodu, ale vycúvaním – použitím kompenzačných transakcií (resp. operácií).

Príklady ochrany transakcií pomocou žurnálového súboru:
**Dvojfázové potvrdzovanie:**

- transakcia nesmie meniť hodnoty objektov skôr, ako je potvrdená,
- transakcia nemôže byť potvrdená skôr, ako sa vytvorí príslušný záznam v žurnálovom súbore.

Najprv sa zaznamenajú zmeny do žurnálového súboru a až potom začne samotná transakcia. Ak sa vyskytne chyba, sú v žurnálovom súbore k dispozícii všetky údaje spred spustenia.
Do žurnálového súboru sa zaznamenajú aj údaje o začatí a ukončení transakcie. Od začatia po koniec transakcie je potrebné zneprístupniť objekty, ktoré by bolo možné meniť inými transakciami – uzamknúť ich.

**Priamy zápis do bázy dát:**
Pri predošlom postupe sa nové hodnoty priraďovali báze údajov až po skončení transakcie, čo je pri dlhšie trvajúcich transakciách v prostredí s možnosťou paralelného spracovania dosť nevhodné.
V takomto prípade je lepší protokol, ktorý bude meniť údaje v báze dát „po jednom“ – predpokladá sa uvoľnenie objektu iným transakciám hneď po zápise jeho zmeny. Predpokladom je nenarušenie konzistentnosti údajov. V prípade zničenia transakcie je možné obnovenie na základe údajov v žurnálovom súbore, no okrem zničenej transakcie je nutné vycúvať aj všetky transakcie, ktoré čítali objekty ňou zmenené.
Z toho plynie potreba mať v žurnálovom súbore údaje aj o tom, ktoré transakcie ktoré údaje čítali.

# Bezpečnosť DBS

Dnes vystupuje do popredia potreba zabezpečenia údajov, ktoré sú evidované v elektronickej podobe. Keďže DBS slúžia hlavne na evidenciu, je nevyhnutné, aby obsahovali mechanizmy, ktoré zaručujú, že k údajom sa dostanú iba osoby s potrebným oprávnením. Musíme robiť všetko pre to, aby sme

zachovali:

- **Integritu databázy** – hrozbou je modifikácia údajov neautorizovanou osobou,
- **Dostupnosť údajov** – nesmie nastať chyba v prístupe autorizovaných osôb,
- **Dôvernosť uložených informácií**– zverejnenie údajov neoprávneným osobám.

Na to, aby sa uvedené požiadavky dosiahli, boli vypracované viaceré postupy, o ktorých si teraz v krátkosti povieme. V prvom rade musíme zobrať do úvahy všetky prvky, ktoré ovplyvňujú bezpečnosť uložených údajov. Medzi ne patrí:

- **Databázový systém** – integrovaný subsystém, ktorý patrí do SRBD.
- **Operačný systém** – „diera" v OS znamená potenciálne riziko aj pre aplikácie, ktoré na ňom „bežia". Napríklad po útoku môže nastať zmazanie súborov na disku, a teda aj súborov, ktoré používa databázový systém.
- **Sieť** – pre DBS musíme využiť všetky prvky ochrany, ktoré chránia aj ostatné súčasti IS
- **Organizačné zabezpečenie** – bezpečnostné smernice organizácie. Ľudský faktor je jedným z najdôležitejších, a preto mu treba venovať zvýšenú pozornosť. V prípade uchovávania údajov je nevyhnutné zabezpečiť, aby žiaden používateľ, ani administrátor, nemal prístup k všetkým údajom bez toho, aby nebol niekým kontrolovaný.

Vieme, že k údajom uloženým v DBS môže pristupovať viacero používateľov. Od úrovne dôvernosti týchto údajov sa odvíja, aký spôsob bezpečný autorizácie používateľov použijeme. K základným spôsobom zaraďujeme:

- Identifikácia a autorizácia na základe loginu a hesla (čo vieš)
- Autorizácia fyzickým predmetom (čo máš)
- Biometrické overenie (čo si)
- Certifikácia (kto to povolil), využitie pri elektronickom podpise

**Bezpečnostné mechanizmy SRBD**

Na zodpovedanie otázok ohľadne bezpečnosti pri práci s DBS možno použiť bezpečnostné mechanizmy ktoré sa používajú vo všeobecnosti všade tam, kde sa má zabezpečiť kontrolovaný prístup k informáciám. Týchto mechanizmov existuje viacero, v základe ich môžeme rozdeliť na diskrétny a mandatórny.

## Diskrétny

Nazývame ho aj Diskrétna kontrola prístupu. Je postavený na princípe „Všetko alebo nič" (používateľ má alebo nemá oprávnenia). Tento prístup podporuje väčšina DBS, pretože je dostatočne flexibilný a ľahko implementovateľný. Umožňuje nám presne definovať, ktorý objekt databázy (databáza, tabuľka, stĺpec, riadok) vidí každý z používateľov. Každý z používateľov môže získať práva na čítanie, vkladanie, aktualizáciu, či odstraňovanie záznamov, ako aj na ich ľubovoľnú kombináciu. Navyše, môžeme definovať aj práva týkajúce sa databázovej schémy:

- Práca s indexmi
- Autorizácia k zdrojom = tvorba tabuliek
- Autorizácia zmien = zmena štruktúry tabuliek
- Autorizácia odstraňovania údajov (truncate, delete)
- Autorizácia odstraňovania tabuliek (drop)

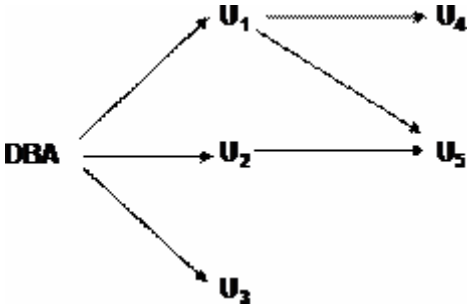
Nedostatkom diskrétneho bezpečnostného mechanizmu je jednoduchšie napadnutie systému.

## Prideľovanie oprávnení

Práva na prístup k údajom definuje Administrátor databázového systému – DBA. Administrátor:

- Vytvára účty
- Prideľuje oprávnenia (grant)
- Odoberá oprávnenia (revoke)
- Definuje bezpečnostnú úroveň

Aby sa predišlo chaosu v prideľovaní práv, je nevyhnutné definovať a priebežne aktualizovať **autorizačný graf**. Autorizačný graf predstavuje zobrazenie štruktúry používateľov s údajom o tom, kto im práva pridelil. Ak to DBA povolil, môžu si používatelia prideľovať navzájom práva.




## Role používateľov

Často zistíme, hlavne ak ako administrátori máme na starosti správu mnohých používateľov, že práva viacerých používateľov systému sú rovnaké a definovanie jednotlivých oprávnení nám zaberá veľa času. V tomto prípade umožňujú DBS definovať role používateľov, čo sú vlastne množiny používateľov s rovnakými právami. Zvýšenú pozornosť musíme v tomto prípade venovať tým používateľom, ktorý by mali patriť do viacerých rolí. Existuje však viacero spôsobov riešenia tohto problému

## Mandatórny bezpečnostný mechanizmus

Nie vždy si však vystačíme s uvedeným spôsob správy bezpečnosti. Niekedy potrebujeme presnejšie rozdeliť používateľov a údaje a definovať ich práva. Preto bol vytvorený mandatórny bezpečnostný mechanizmus, o ktorom si povieme niekoľko základných faktov.
Tento mechanizmus môžeme charakterizovať nasledovne:

- Umožňuje klasifikáciu používateľov a údajov do bezpečnostných tried
- Zavádza viacúrovňovú bezpečnosť
- Poskytuje vysoký stupeň ochrany

Pre tento mechanizmus bolo vypracovaných viacero modelov. K najznámejším patria model Bell-La Padula, model Biba, Multi-level security a Clark-Wilsom – vhodný pre priemysel a firmy.
Ak si zoberieme model Bell-La Padula, tak jeho základnou charakteristickou črtou je:

- každý subjekt (používateľ, rola) a objekt (tabuľka, stĺpec, pohľad) je klasifikovaný do bezpečnostnej triedy.
- Subjekty môžu čítať zhora nadol a zapisovať zdola nahor

Model má vypracované pravidlá prístupu k údajom, využíva klasifikáciu údajov na nasledujúce úrovne:

- Prísne tajné (PT)
- Tajné (T)
- Dôverné (D)
- Vyhradené (V)

- Neklasifikované (N)

Pričom platí, že PT >= T >= D >= V >=N.
Alternatívou k modelu Bell-La Padula je model Biba. Tento model je postavený na opačnom prístupe ako predchádzajúci model. Je charakteristický princípom jednoduchej integrity - zákaz zápisu smerom nahor. Zavádza pojem Integritné hviezdičkové vlastníctvo – zákaz čítania smerom nadol. Dôležité je, aby sme si zapamätali, že keďže oba modely používajú opačný princíp definovania oprávnení, nemožno ich vzájomne kombinovať.

## Riadenie prístupu na základe rolí

Alternatívou k diskrétnemu a mandatórnemu bezpečnostnému mechanizmu je riadenie prístupu na základe rolí. V tomto prípade má každý používateľ pridelenú rolu, ktorá zároveň predstavuje prístup do databázy – nesie všetky oprávnenia pre používateľa. Výhodou tohto prístupu je, že reflektuje organizačnú štruktúru a je veľmi flexibilný. Navyše, môže zahrnúť aj diskrétny alebo mandatórny mechanizmus.

## Autorizácia v SQL

Hovorili sme, že jazyk SQL je silným nástrojom pri práci v relačných DBS, a preto niet divu, že zasahuje aj do oblasti bezpečnosti DBS a správy používateľov. Vo svojom štandarde obsahuje príkazy, ktoré dovoľujú vo forme SQL dotazov prideľovať a odoberať práva, vytvárať používateľov, či role. K základným príkazom patria príkazy GRANT a REVOKE:

- **GRANT**
  - GRANT <zoznam oprávnení> ON <tabuľka> TO <používateľ|rola>
  - GRANT SELECT ON zamestnanci TO user1
  - GRANT UPDATE ON platy TO sef
  - Ak má používateľ práva ďalej prideľovať oprávnenia, pridá sa príkaz WITH GRANT OPTION
  - GRANT INSERT ON dochadzka TO sekretarka WITH GRANT OPTION
- **REVOKE**
  - REVOKE <zoznam oprávnení> ON <tabuľka> TO <používateľ|rola> [RESTRICT | CASCADE]
  - [RESTRICT | CASCADE] – odňatie práv aj používateľom, ktorých vytvoril používateľ, ktorému sa odoberajú práva

Príkazy jazyka SQL podrobnejšie popisuje druhý študijný materiál, ako aj referenčné manuály ľubovoľného DBS, s ktorými sme sa počas tohto predmetu stretli.

## Zdroje


1. GROFF, J. R. - WEINBERG, P.N. : SQL: Kompletní průvodce. Brno - Computer Press. 2005. ISBN 80-251-0369-2
2. FRICK, D. R. : Database Theory & Practice. http://www.frick-cpa.com/ss7/Theory_RelationalDB.asp (2.1.2006)
3. TELNAROVÁ, Z : Úvod do databází. Ostrava 2003. ISBN 80-7042-847-3
4. ZAIANE, O. R. : CMPT 354 Database Systems and Structures. http://www.cs.sfu.ca/CC/354/zaiane/ (4.1.2006)
5. POKORNÝ, J. - HALAŠKA, I : Databázové systémy. Vydavatelství ČVUT, 2003. ISBN 80-01-02789-9
6. RIORDAN, R. R. : Vytváříme relační databázové aplikace. Praha : Computer Press. 2000. ISBN 80-7226-360-9
7. SCHEBER, A. : Databázové systémy. ALFA – Vydavateľstvo technickej a ekonomickej literatúry, Bratislava, 1988.
8. RAMARKISNAN, R.: Database management systems, McGraw Hill, Secon Ed., 2000
9. GARCIA-MOLINA, H. - ULLMAN, J. – WIDOM, J.: Database Systém Implementation, Prentice-Hall, 2000
10. Transakcie (Referát) [online], JOZEF SRPOŇ URL: http://hornad.fei.tuke.sk/~genci/Vyucba/SRBDp/2003-2004/08-TransakcneSpracovanie/Srpon/
11. Transakčné spracovanie (Referát) [online], J8N VIDA URL: http://hornad.fei.tuke.sk/~genci/Vyucba/SRBDp/2003-2004/08-TransakcneSpracovanie/Vida/SRBD_Jan_VIDA_Referat/
12. DELIKÁT, T. : Základy databázových systémov. DELINT, Bratislava 2006, 209 s. ISBN: 80-969484-4-X.