4. Databázové systémy

Osnova

1. Základní pojmy, principy, architektury
2. Ukládání a reprezentace dat, zpracování dotazu, transakce, indexování, hašování, datové modelování
3. Metadata
4. Datové sklady

Výklad

1. Základní pojmy, principy, architektury

- **Databáze**
  Neboli datová základna je **organizovaný soubor dat** (většinou se jedná o relační model – tabulky dat) **uložená na paměťovém médiu**. V širším smyslu jsou součástí databáze i softwarové prostředky (systém řízení báze dat (SŘBD)), které umožňují manipulaci s uloženými daty a přístup k nim.
  Běžně se označením databáze – v závislosti na kontextu – myslí jak uložená data, tak i software (SŘBD).
  Jsou všudypřítomné, potřebuje každá firma a je součástí většiny aplikací.
- **Systém řízení báze dat (SŘBD**, **Database Management System)**
  Jedná se o **software**, který je schopen **efektivně pracovat s velkým množstvím dat**, **manipulovat** (vkládat, modifikovat, mazat) a **definovat strukturu** (čímž se liší od prostého souborového systému).
  Tvoří rozhraní mezi aplikačními programy a uloženými daty.
  Občas se pojem zaměňuje s pojmem databázový systém. Databazový systém však je SŘBD dohromady s bází dat.
- **Databázový systém**
  Striktně myslíme **DB + SŘBD**.
  Funkce:
  - Podporu pro definici datových modelů (například relační, logický, objektový)
  - Správa klíčů: vlastní (interně implementované) indexování, dodržování unikátnosti hodnot ve sloupcích, nad kterými je definován unikátní nebo primární klíč; implementace fulltextového vyhledávání pro fulltextové klíče; implementace cizích klíčů.
  - Využití některého jazyka vyšší úrovně pro manipulaci a definici dat (např. SQL, QBE, datalog, Common English Query) a vyřešení komunikačního kanálu mezi uživatelem či skriptem a SŘBD v tomto jazyku.
  - Autentizaci uživatelů a jejich autorizaci k operacím nad daty (u každého uživatele může být definováno, jaký typ příkazů je oprávněn spouštět)
  - Robustnost a zotavitelnost po chybách bez ztráty dat
  - Správu transakcí, atomicitu jednotlivých příkazů
  - Uložené procedury
  - Triggery
  - Integritu dat; například nepovolením vložení duplicitního řádku s unikátním klíčem nebo řádku s hodnotami NULL u sloupců, které NULL být nesmějí.
  - Profilování, statistické informace o běhu dotazů, procesů, přístupu uživatelů atd.
- **Relační model**
  Relace chápeme matematicky jako obecně n-ární relace tedy libovolný vztah mezi skupinou prvků jedné nebo více množin: $A_1, A_2, A_3, \ldots, A_n$, kde $n \in N$, rozumíme **libovolnou podmnožinu kartézského součinu n množin**.
  Jedná se o strukturu – data v relacích (tabulkách).
  Operace – dotazování, modifikace (SQL, relační alebgra)

- **Relační algebra**
  Relační algebra je nejzákladnějším prostředkem **pro práci s tabulkami**. Mezi základní operace patří **projekce**, **selekce** a **spojení**.
  Jedná se o procedurální jazyk, ve kterém definujeme postup provedení dotazu.

Příklad implementace primitivního DB systému

- Relace jsou v souborech na disku (co relace to soubor se strukturou a v adresáři existuje seznam platných relací a definice struktury – tj. využíváme služeb souborového systému).
- Problémy takovéto implementace:
  - Žádné souběžné zpracování
  - Žádná spolehlivost (operace nemusí být dokončena)
  - Vyhledávání je drahé (nejsou indexy – hledání podle primárního klíče je drahé (vždy je nutné přečíst celou relací))
  - Nedostatečná bezpečnost (práva jsou příliš hrubozrnná: rwx pro vlasníka, skupinu a ostatní)

Databázový systém

Databázové systémy byly vyvinuty kvůli zvládnutí následujících problémů při zpracování souborů v tradičních operačních systémech:

- Redundance a nekonzistence dat
- Problémy s přístupy k datům
- Izolace dat – různé soubory a formáty
- Problémy s integritou
- Jedinečnost (atomicita) aktualizací
- Současný přístup více uživatelů
- Bezpečnostní problémy

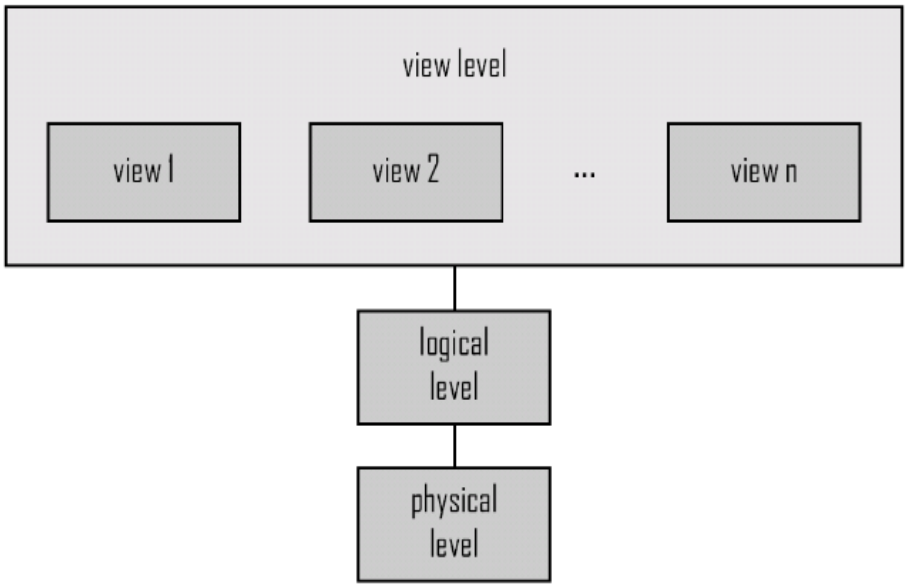
Databázové systémy mají tedy následující vlastnosti:

- Struktury datových souborů jsou odděleny od aplikačních (uživatelských) programů.
- Přístup k datům je možný jen prostřednictvím programů databázového systému.
- Data je možné vyhodnotit jakýmkoliv způsobem.
- Je umožněn přístup více uživatelů současně a vyřešena ochrana dat před zneužitím.
- Správa databáze zahrnuje prostředky pro popis dat a popis algoritmu.

Části DBS:

- **Pohledy** – **různé pohledy na ta samá data** (např. chceme skrýt určité sloupce v tabulce + oprávnění).
  Neboli externí úroveň – reprezentována daty z pohledu uživatele (např. formuláře pro vstup dat, výstupní tiskové sestavy), různí uživatele vidí různě vymezené části databáze, jde o externí schéma.
- **Logická vrstva** – atributy + typy – tj. **tabulky** + **integritní omezení** atd.
  Konceptuální úroveň – popisuje data uložená v databázi a vztahy mezi nimi, jde o logické schéma.

- **Fyzická vrstva** – zakódování dat do bitových řetězců – tj. **způsob uložení dat**.
  Interní úroveň – popisuje fyzický způsob uložení dat na vnějších paměťových médiích a metody přístupu k datům, jde o fyzické schéma.

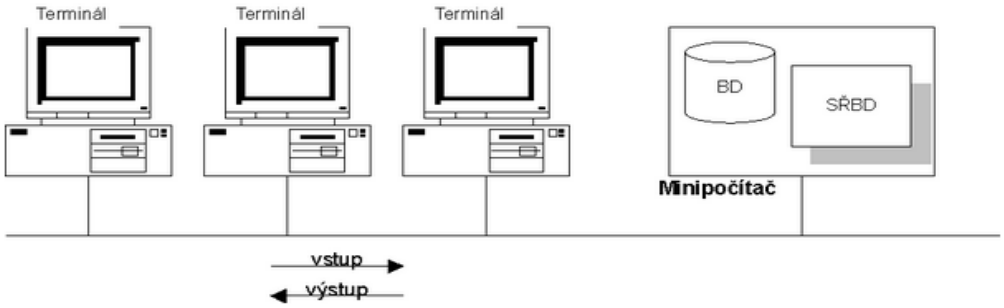


Architektura:

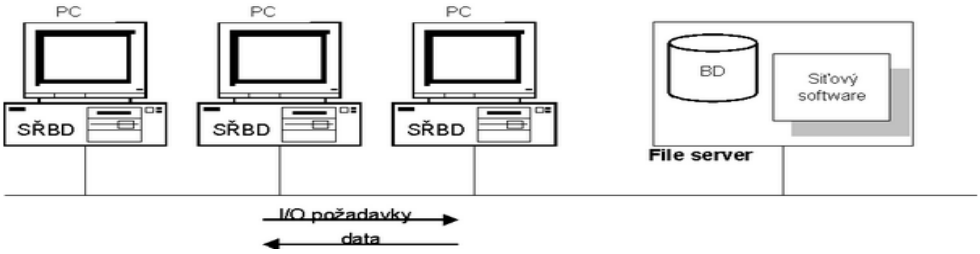
- **Externí úroveň** – reprezentována daty **z pohledu uživatele** (např. formuláře pro vstup dat, výstupní tiskové sestavy – např. **objekt Osoba**), různí uživatele vidí různě vymezené části databáze, jde o externí schéma
- **Konceptuální úroveň** – popisuje data uložená v databázi a vztahy mezi nimi, jde o **logické schéma (tabulky)**
- **Interní úroveň** – popisuje **fyzický způsob uložení dat** na vnějších paměťových médiích a metody přístupu k datům, jde o fyzické schéma

Jiné dělení

- **Centrální architektura** – je báze dat i systém řízení báze dat v centrálním počítači. Komunikaci s uživateli zprostředkovávají terminály na pracovištích.
  Po síti se přenášejí vstupní údaje z terminálu do centrálního počítače a tam se zpracovávají centrálním programem. Tím že data se zpracovávají centrálně, kde může probíhat i více úloh, mohou mít odezvy na dotazy určité zpoždění.

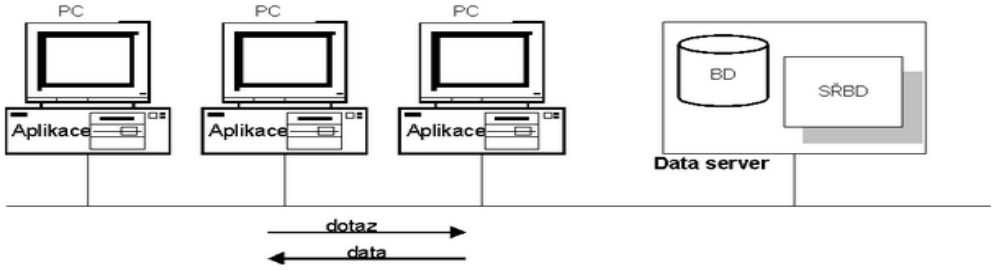


- **Architektura file-server** – souvisí s rozšířením PC a lokálních sítí. Data v BD jsou umístěna na počítači pracujícím jako file-server a jsou sdílena pro jednotlivé aplikace a SŘBD na jednotlivých počítačích. Vzhledem k tomu, že k datům může přistupovat více aplikací, musí být zajištěna ochrana používaných záznamů. Komunikace probíhá podle následujícího postupu:
  - Uživatelský dotaz
  - SŘBD zpracuje dotaz a odešle požadavek na DB
  - file-server odesílá bloky dat na lokální počítač, kde jsou data dále zpracovávána SŘBD
  - výsledky se zpravidla uloží na PC, zobrazí na monitoru , ev, vytiskne jako sestavu.

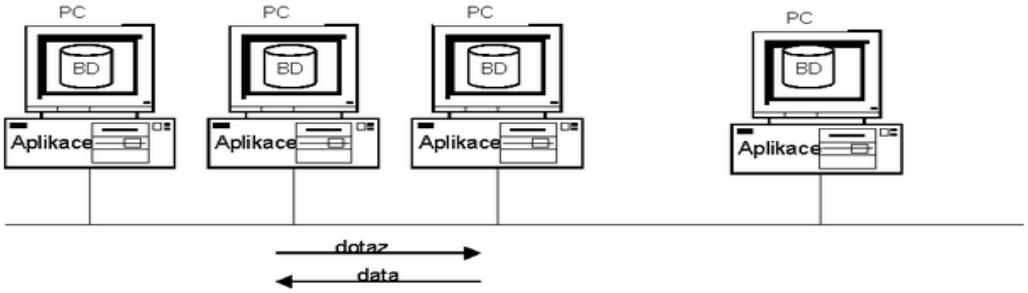


- **Architektura klient-server** – tato architektura je umožněna rozvojem sítí LAN (lokálních sítí) rozdělených na osobní počítače (PC) a databázový server. Na PC běží aplikace, které předávají dotazy na datový server (většinou pomocí strukturovaného jazyka), který je zpracovává a potřebné výsledky předává zpět na PC. Proto je server nejvíce zatíženým počítačem. Průběh je následující:
  - Aplikace na PC připraví dotaz na data v podobě SQL dotazu a odešle jej na server
  - Server zpracuje dotaz
  - Výsledek dotazu je odeslán na PC, kde jej aplikace převede do výstupní podoby.

  Tato architektura snižuje požadavky na množství dat pohybujících se v síti, proto vyhovuje i rozsáhlým aplikacím od renomovaných firem.

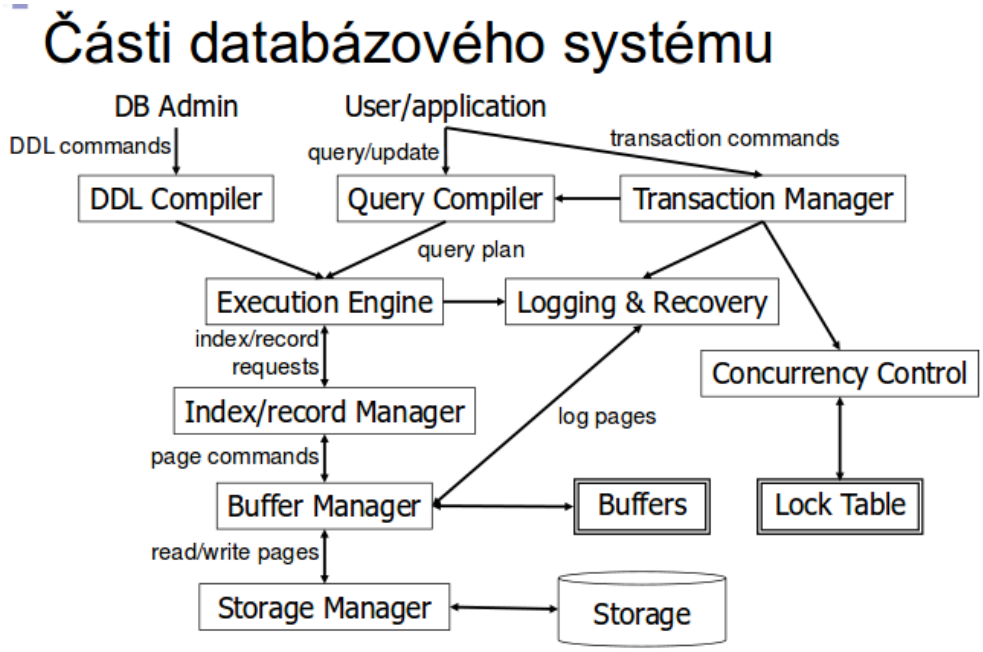


- **Distribuovaných databází** – v tomto systému jsou data rozložena v několika počítačích. Navenek se tváří jako jediná veliká databáze. Je charakterizovaná třemi vlastnostmi:
  - Transparentnost - klientovi připadá, že data jsou zpracovávána na jediném serveru v síti, není specifikováno místo uložení dat, to řídí SŘBD.
  - Autonomnost - s každou částí BD je možno pracovat samostatně , data jsou spojována dynamicky podle potřeby SŘBD.
  - Nezávislost na typu sítě - architektura podporuje různé typy sítí a jejich vzájemné propojení. Pro komunikaci se nejčastěji používá strukturovaný dotazovací jazyk SQL.



Hlavní součásti databázového systému:

- **Storage Manager** – stará se o diskový prostor, kde jsou data
  - správa bloků na disku
  - správa vyrovnávací paměti
- **Query Processor**
  - překlad dotazu
  - optimalizace
  - vyhodnocení dotazu a zvolení nejvhodnějšího
- **Transaction Manager** – stará se o transakce
  - ACID



Jazyk pro definici dat (Data Definition Language, DDL)

Definuje sadu příkazů, které lze použít pro vytvoření, úpravu a odstranění objektů (tabulky, pohledy, procedury, funkce) v databázi

- CREATE (vytvoření)
- ALTER (úprava)
- DROP (odstranění)

Jazyk manipulace s daty (Data Manipulation Language, DML)

Množina příkazů, které se používají pro výběr, vkládání, úpravu a mazání dat v tabulkách.

- SELECT
- INSERT
- UPDATE
- DELETE

Jazyk pro řízení přístupových práv (Data Control Language, DCL)

- GRANT
- REVOKE

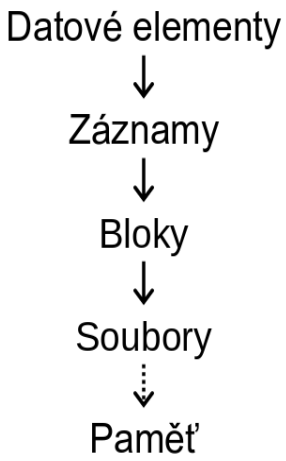
Příkazy pro řízení transakcí

- START TRANSACTION
- COMMIT
- ROLLBACK

2. Ukládání a reprezentace dat, zpracování dotazu, transakce, indexování, hašování, datové modelování

Ukládání a reprezentace dat

- Data lze chápat na několik úrovní:
  - **Datové elementy** – ID, číslo, řetězec
  - **Záznamy** – sada elementů, které spolu souvisí – tabulky, např. tabulka osoby (záznam je řádek v tabulce)
  - **Bloky** – atomické jednotky, které se čtou/zapisují na disk
  - **Soubory** – seznam bloků
  - **Paměť** – persistentní úložiště



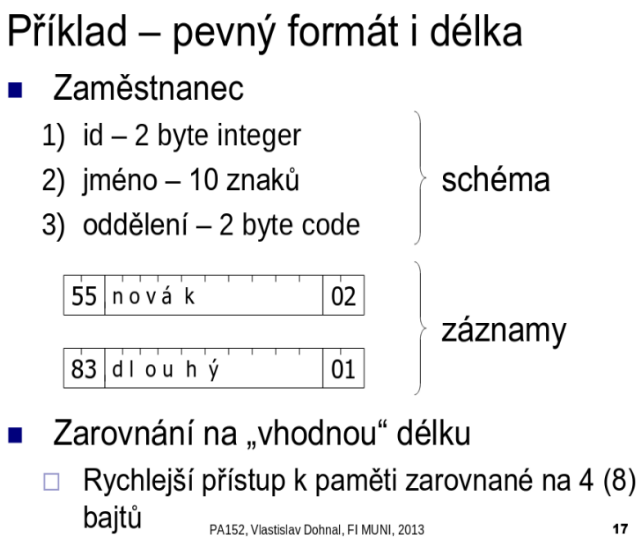
Databáze je uložena v **kolekci souborů**.
Každý **soubor** je tvořen **posloupností záznamů**.
Záznam se skládá z jednotlivých **atributů** (datových elementů), které mají svůj **typ** buď pevné (většinou) nebo proměnlivé délky. V nejjednodušším případě je délka záznamu pevná, každý soubor má pouze záznamy jednoho typu a každá tabulka má právě jeden soubor.
Záznamy (ať už pevné nebo proměnné délky) ukládáme do **bloků** pevné velikosti. Záznamy můžeme oddělovat mezi sebou a rozdělovat/nerozdělovat do více bloků.

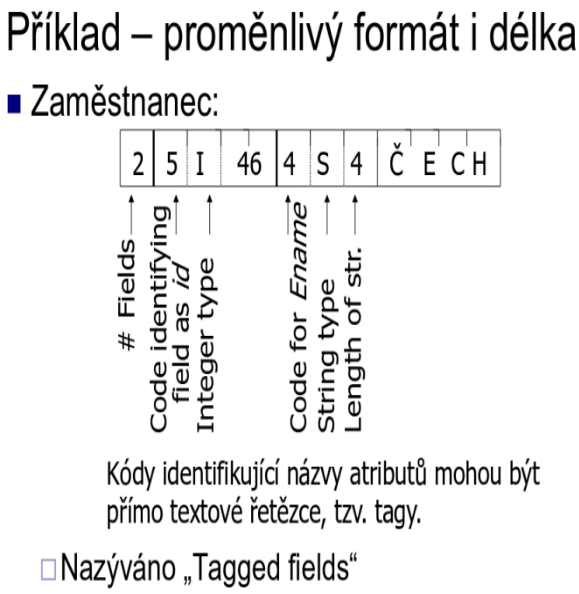
Typy datový elementů

- Celé číslo - podle rozsahu 2 B, 4 B, obvykle přímý kód nebo inverzní kód (ne doplňkový)
- Reálné číslo - plovoucí čírka (IEEE 754), pevná čárka
- Znak - 1 B v ASCII, více bajtové znaky
- Pravdivostní hodnota - obvykle jako celé číslo (true – samé 1, false – samé), s bity se nepracuje snadno, takže alespoň jeden B
- Bitové pole
- Datum - počet dní od počátku, řetězec YYYYMMDD
- Čas - počet sekund od půlnoci, řetězec HHMMSSFF, časové zóny
- Výčtový typ
- Řetězec – proměnlivá délka, pevná délka

Záznam

- Jedná se o **seznam souvisejících datových elementů**. Např. Zaměstnanec: jméno – Novák, plat – 1234, datum_přijetí – 1.1.2000.
- Schéma záznamu popisuje strukturu záznamu:
  - **počet atributů**
  - **typ a název každého atributu**
  - **pořadí** atributů
- Typy záznamů podle schématu:
  - Pevný formát – schéma je společné pro všechny záznamy (je uloženo mimo záznamy)
  - Proměnlivý formát – každý záznam obsahuje svoje schéma, vhodné pro řídké záznamy (když není atribut, tak není ve formátu → není potřeba ukládat null), opakování stejných atributů, vyvíjející se formát
- Typy záznamů podle délky:
  - Pevná délka – každý záznam má stejnou délku (počet bajtů)
  - Proměnlivá délka – ušetření paměti, ale velký problém u při změně atributu – např. o jedno písmeno více je potřeba změna na disku!





Uložení záznamů do bloků:

- Záznamy – pevné délky nebo proměnné délky
- Blok – atomická jednotka! pevné velikosti
- Techniky:
  - Oddělování záznamů

Oddělování záznamů

- Záznamy pevné délky
  - Žádný oddělovač
  - Pamatovat počet a ukazatel na první záznam
- Oddělovač
- Ukládání délek záznamů (nebo počátků)
  - V rámci záznamu
  - V hlavičce bloku

  - Míchání (shlukování)

Míchání (shlukování) záznamů

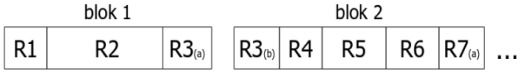
- Záznamy různých relací v jednom bloku
  - Některá data jsou vyžadována současně
  - Uložit je společně → zrychlené čtení
  - Složitější implementace

  - Rozdělování / Nerozdělování záznamů

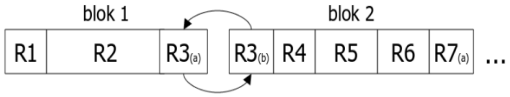
Rozdělování vs. nerozdělování záznamů

- Nerozdělování
  - každý záznam součástí jednoho bloku
  - jednodušší, ale může plýtvat místem

- Rozdělování
  - Záznam „přetéká“ mezi bloky
  - Nutné, pokud je záznam větší než blok!



- Záznam „přetéká“ mezi bloky
  - Musíme udržovat pořadí bloků
    - Lze používat ukazatele



- Záznam je rozdělen na „fragmenty“
  - Bitový příznak, zda je fragmentován
  - Ukazatel na další / předchozí fragment

- Uspořádání záznamů

Uspořádání záznamů

- Záznamy jsou v souboru (a bloku) uspořádány
  - Podle hodnoty nějakého klíče
  - Např. sekvenční soubor
- Důvod:
  - Efektivní čtení záznamů v daném pořadí
  - Např. pro merge-join, order by, …

- Odkazy na záznam

Odkazy na záznamy

- Vhodná varianta = **kombinace**
  - Fyz. adresa záznamu = fyz. adresa bloku + offset
    - Offset je pořadí záznamu v bloku
      - V hlavičce je obvykle seznam odkazů na záznamy.
  - Výhody
    - Lze přesouvat záznamy v bloku
      - Beze změny fyz. adresy
    - Lze přesunout záznam do jiného bloku
      - V původním místě udělám odkaz na nový blok + offset
    - Lze zrušit map table
  - Nevýhoda
    - Nízká flexibilita přesouvání bloků (defragmentace)

Odkazy na záznamy

- Používaná varianta
  - Adresa záznamu = ID souboru + číslo bloku + offset v bloku
  - Uložení bloku určuje systém souborů (file system)

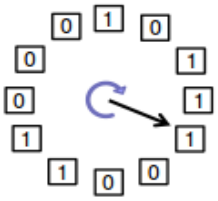
Organizace záznamů v souboru:

- Halda – záznam je uložen kdekoli na volné místo v souboru
- Sekvenční – záznamy jsou v souboru uspořádány podle vyhledávacího atributu
- Hašování – pro výpočet čísla bloku, kde má být záznam uložen, se používá hašovací funkce (toto číslo je vypočítáno na základě hodnot vybraných atributů)
- Shlukování – záznamy různých tabulek mohou být uloženy v jednom bloku (některá data jsou vyžadována současně)

Správa vyrovnávací paměti

Správa vyrovnávací paměti

- Různé strategie
  - LRU, FIFO, pinned blocks, toss-immediate, …
- Problém odkazování na záznamy v paměti
  - Prohazování odkazů (swizzling)
- Specifikum databází
  - Někdy je nutné ponechat bloky v cache „déle“
    - Indexy, vyhodnocování spojení relací, …
- Hodiny („Clock“)
  - Efektivní aproximace LRU
  - Ručka ukazuje na poslední načtený blok.
  - Pro načtení nového bloku se ručka otáčí, dokud nenalezne volné místo (blok s nulou).
  - Pak načte požadovaný blok a nastaví 1
  - Ručka při otáčení snižuje číslo (až na nulu).
- Lze implementovat i *pinned blocks*. Jak?

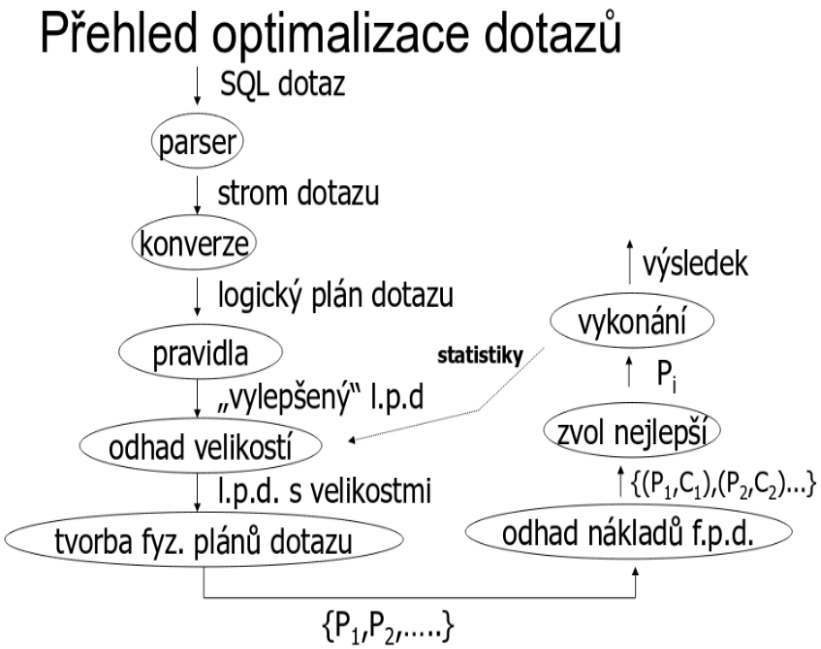


Zpracování dotazu

- Postup vyhodnocení dotazu:
  - **Dotaz** (SQL příkaz)
  - **Syntaktická a sémantická kontrola** (analýza, kontrola a vytvoření stromu dotazu ve formě výrazu v relační algebře)
    Strom dotazu
  - **Logický plán** – to, co se má udělat, postupuje se od listů + úpravy (lze prohodit uzly, některé uzly se spojí, štěpí atd.)
  - **Fyzický plán** – k jednotlivým uzlům (jenž jsou operace) se přiřadí nějaký konkrétní algoritmus (např. SELECT může být sekvenční proběhnutí/dle indexu/vytvoření indexů a použití …)
  - **Vyhodnocení**
- Optimalizace
  Znamená, že se z několika logických plán se vybere jeden, pro který je několik fyzických, ze kterých se vybere jeden finální plán.

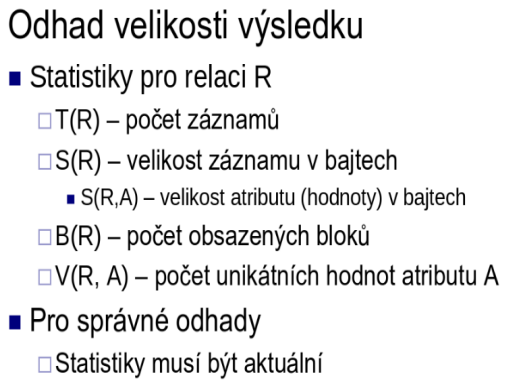
Dotaz se nejprve pomocí **parseru** převede na **syntaktický strom** reprezentující strukturu dotazu.
Ten se po té zpracuje do výrazů relační algebry (**logický plán** dotazu). Pomocí transformačních pravidel (kombinace přirozeného spojení, kartézského součinu, sjednocení, selekce a projekce) dále vznikne **vylepšený logický plán**.
Nyní se za pomocí různých statistik (počet záznamů, velikost záznamů v bajtech, počet obsazených bloků, počet unikátních hodnot daného atributu) odhadnou velikosti výsledků, které ovlivňují odhad ceny provedení. Následně se logický plán transformuje na **fyzický plán**, který určí pořadí operací nutných k

vykonání.
Porovnají se různé fyzické plány, odhadnou se náklady (velikost výsledků, počet V/V operací) a zvolí se **nejlevnější**.
Nakonec se daný plán provede a tím se získá **výsledek**.



Odhad ceny plánu dotazu:

- Odhad velikosti výsledku operace
  - vzorce pro Kartézský součin, SELECT, přirozené spojení, projekce, množinové operace ...
  - Odhad počtu V/V operací

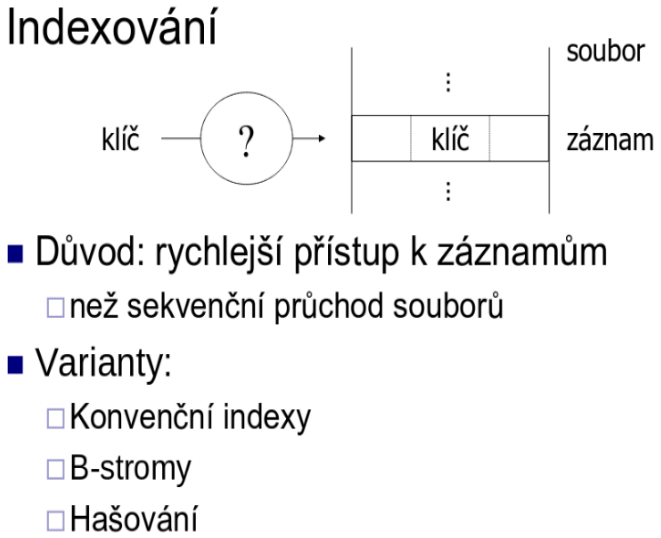


- Odhad velikosti výsledků je „umění".
  Pro korektní odhad potřebujeme korektní statistiky → nutnost udržovat tabulky při modifikacích.
  Jaké jsou náklady takové údržby?

Indexování

- Index je datová struktura také uložena na disku urychlující vyhledávání (sekvenční přístup je pomalý).

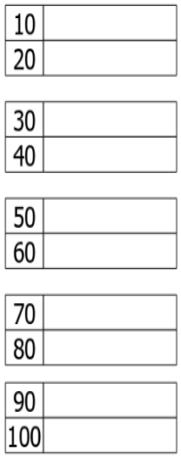
Indexové mechanizmy se používají pro zrychlení přístupu k požadovaným datům. Na druhou stranu dojde ke zpomalení operací (INSERT, UPDATE) měnících obsah indexovaných sloupců.
Vytvořením indexu databázový systém zarezervuje pro požadovaný index určitou část paměťového prostoru a uloží do něj informace o rozmístění hodnot indexovaných sloupců v tabulce. Pokud později dojde k dotazu, který se týká indexovaných sloupců, není tabulka prohledávána podle toho, jak jsou za sebou řádky uloženy, ale pomocí informací uložených v paměťovém prostoru indexu je přistupováno přímo k relevantním řádkům tabulky (něco jako rejstřík v knize).



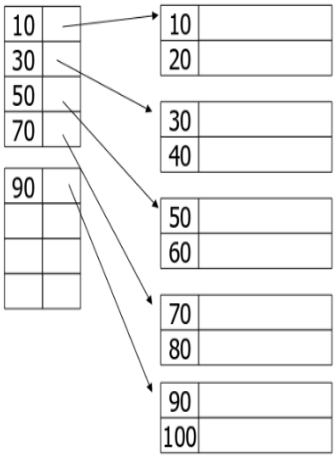
Pojmy:

- **Vyhledávací klíč** – atribut nebo množina atributů používaný pro vyhledávání záznamů v souboru
- **Primární klíč** – atribut nebo množina atributů jednoznačně identifikující každý záznam v tabulce
- **Sekvenční soubor** – záznamy jsou uspořádány podle vyhledávacího klíče
- **Indexový soubor** – skládá se ze záznamů ve tvaru VYHLEDÁVACÍ KLÍČ | UKAZATEL
- **Index -** sekvenční soubor – soubor setříděný podle primárního klíče, ke kterému je vytvořena struktura indexů

Sekvenční soubor

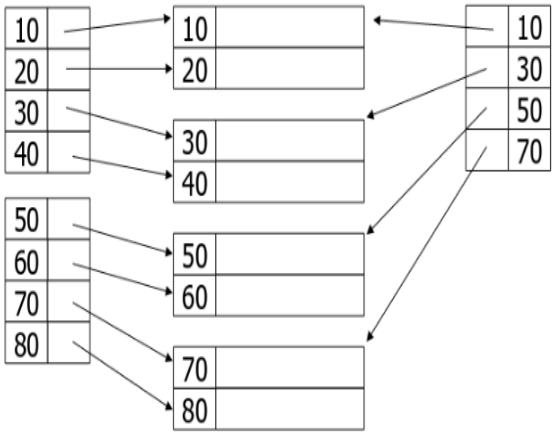


Index-sekvenční soubor



Hustý index

Řídký index

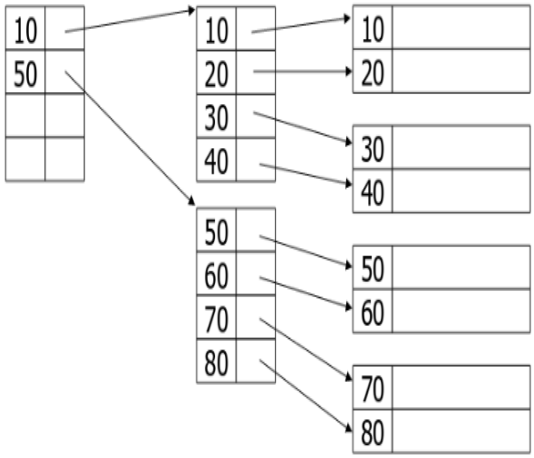


Víceúrovňový index

2. úroveň

1. úroveň

Soubor



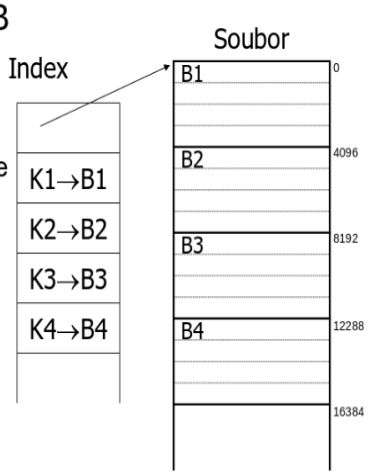
Index a ukazatel:

Index a ukazatele

- Ukazatele v indexech
  - Ukazatel na záznam
    - Adresa bloku + offset záznamu
  - Ukazatel na blok
    - Adresa bloku
    - Obvykle kratší
  - Soubor je spojitý a uspořádaný
    - Ukazatele na bloky se nemusí ukládat
      - tzv. implicitní odkazy
      - „Lze je spočítat“

Implicitní ukazatele na blok

- Blok o velikosti 4KiB
- Hledáme K3
  - Prohledání indexu
    - K3 je ve třetí položce
      - → třetí blok (B3)
  - Offset v souboru
    - (3-1)·4096=8192



Mazání v indexech: [slides04: 15 – 22]

Vkládání v indexech: [slides04: 23 – 27]

Konvenční indexy: shrnutí

- Základní dělení
  - Řídké vs. Husté
- Vkládání / mazání
  - Duplicitní klíče
    - Zejména u sekundárních indexů
- Výhody
  - Jednoduché
  - Index je sekvenční soubor → vhodné pro „full scan“
- Nevýhody
  - Aktualizace drahá
  - Ztráta „sekvenčnosti“ a vyváženosti
    - kvůli přetokovým oblastem

B-stromy:

- Alternativou a zároveň n**ejpoužívanější indexovou strukturou** v databázových systémech jsou B+ stromy.
  Jedná se o víceúrovňový index ve tvaru vyváženého n - árního stromu. Jejich výhodou je, že se při vkládání/mazání provádí automatická reorganizace pouze s malými, lokálními změnami, nevýhodou je ale režie a zvýšené prostorové nároky.

B^+-strom

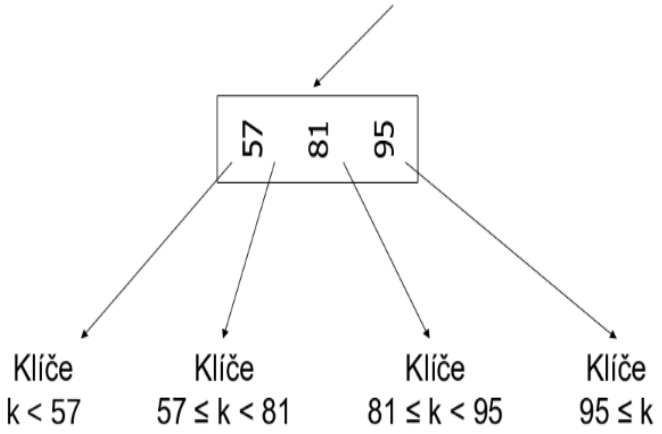
- Parametr *n* (arita stromu) ovlivňuje:
  - Tvar uzlu:

p_1	K_1	p_2	K_2	p_3	…		K_{n-1}	p_n

  - Minimální naplnění
  - Listový uzel
    - Vždy na stejné úrovni
    - p_i odkazuje na záznam s klíčem K_i (data)
    - p_n odkazuje na další list (zřetězení listů)
  - Nelistový uzel
    - p_i odkazuje na uzel, kde jsou klíče K: $K_{i-1} \leq K < K_i$

B^+-strom

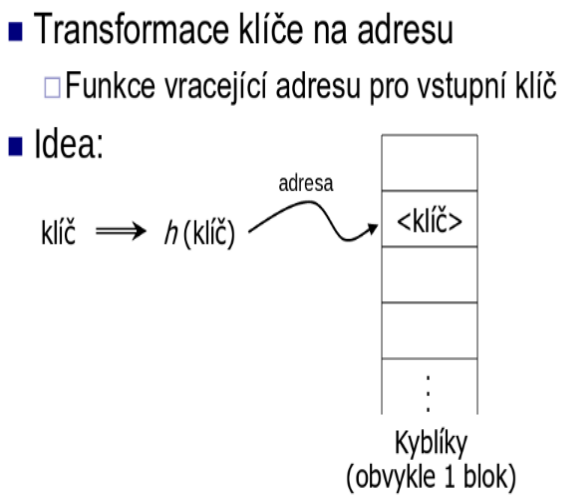
- Vnitřní uzel pro *n=4*



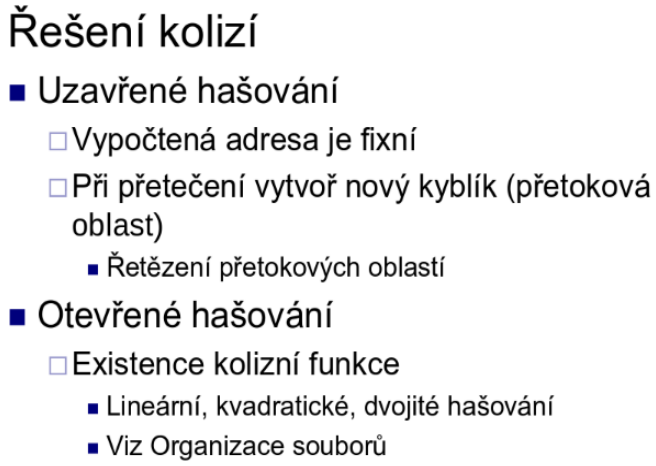
- Vkládána: [slides04: 43 – 50]
- Mazání: [slides04: 51 – 55]

Hashování

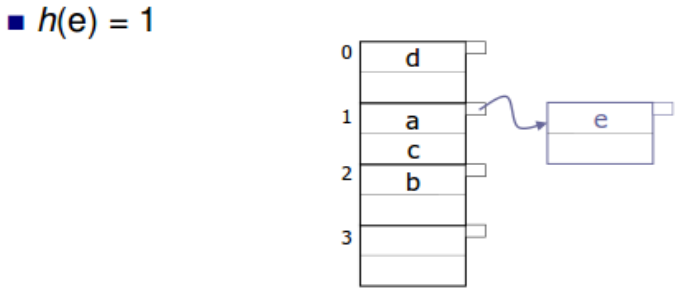
- Hashovací funkce:
  Je matematická funkce (resp. algoritmus) pro převod vstupních dat do (relativně) malého čísla.
  Výstup hašovací funkce se označuje výtah, miniatura, otisk, fingerprint či hash.
  - převede libovolně dlouhý vstup na výstup pevné délky
  - řeší přístup k záznamům s konstantní složitostí
  - ideální hašovací funkce je rovnoměrná a náhodná (vztahuje se na rozložení hodnot)



- **Kolize**:
  Na vypočtené adrese je již něco uloženo. Není problém, pokud lze uložit více klíčů.

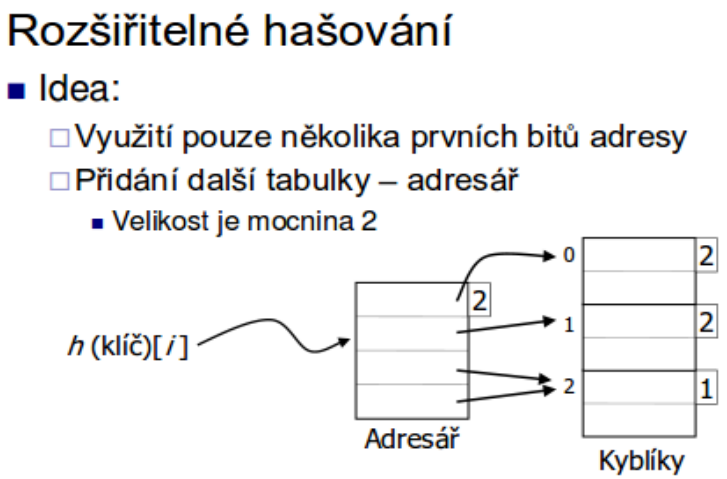


Statické uzavření hashování: – kolize pomocí přetokových oblastí (např. Kapacita kyblíku 2 klíče a poté přetoky)
Při mazání je možné uvolnit přetokvou oblast [slides05:15]
  - Používá se u souborů, které procházejí jen minimem změn
  - Případné změny mohou negativně ovlivnit efektivitu hašování

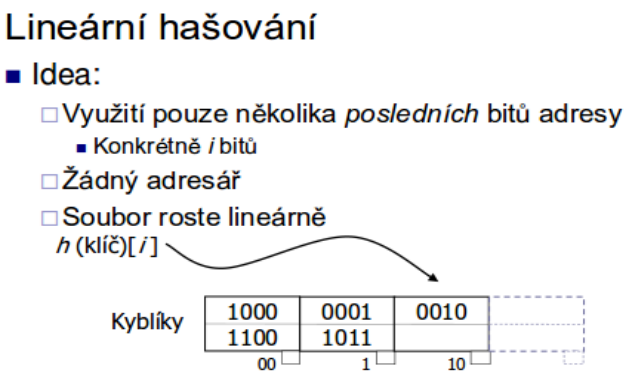


Dynamické hašování:

- K výpočtu adresy se používá pouze prvních *i* bitů z výstupu hašovací funkce; toto *i* se dynamicky mění – pokud je potřeba více adres, tak se zvyšuje, naopak při malém počtu se *i* zmenšuje
- Ppoužívá se u souborů s proměnným počtem záznamů
- Buckety jsou naplněné rovnoměrně – pokud jsou plné, tak se štěpí, pokud jsou prázdné, tak se spojují
- Druhy:
  - **Rozšiřitelné hashování**



    - Hashovací funkce rozmisťuje záznamy do kapes (kapsa = záznamy s jistou podmnožinou klíčů, vymezuje je hašovací funkce)
    - Jako index v adresáři kapes se používá dynamicky určovaný prefix výsledku hašovací funkce h(k)
    - Délku indexu vymezuje globální hloubka adresy kapsy
    - Štěpení
    - Globální hloubka > lokální hloubka – na blok ukazuje více ukazatelů, jednoduše se rozdělí na dva
    - Globální hloubka = lokální hloubka – musí vzniknout nová kapsa a také se zdvojnásobí rozměr adresáře kapes
  - **Lineární hashování**
    - Řeší nedostatky rozšiřitelného hašování za cenu vyšší režie dané manipulací s přetokovými kapsami
    - Počet kapes udržuje tak, aby byly naplněny z např. 80 %
    - Rozdíl oproti rozšiřitelnému hašování: nemusí se vytvářet ani udržovat adresář kapes, přesto počet kapes roste lineárně.



Hašování vs. indexování

- Hašování
  - Vhodné pro přesné dotazy
    SELECT … WHERE a=5
- Indexování
  - Vhodné pro rozsahové dotazy
    SELECT … WHERE a>5

Bitmapový index

- Kolekce bitových polí
  - Pro každou hodnotu je jedno pole
  - Délka pole je počet záznamů
  - = invertovaný soubor
- Vhodné pro atributy s „malým“ počtem hodnot

Bitmapový index: příklad

- Relace R(F,G)

F	*G*
30	foo
30	bar
40	baz
50	foo
40	bar
30	baz

- Bitmapový index pro *G*

Hodnota	*Vektor*
foo	100100
bar	010010
baz	001001

- Nevýhody
  - Paměťová náročnost
    - Pokud je klíč primárním klíčem, pak zabírá $n(n \log_2 n)$ bitů
  - Aktualizace záznamů
    - Vkládání, mazání
    - Nová hodnota → nové bitové pole
- Výhody
  - Rychlé operace na bitech (AND, OR)
  - Lze použít pro rozsahové dotazy
  - Snadné kombinace více indexů dohromady

Datové modelování

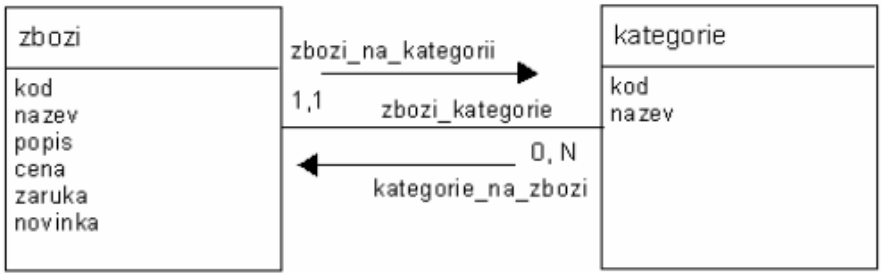
- Cílem datového modelování je navrhnout kvalitní datovou strukturu pro konkrétní aplikaci a databázový systém, který bude tuto aplikace využívat k uložení dat.
- Rozeznáváme:
  - **Databázový model** – definuje **neměnné atributy a strukturu dat** a slouží pro návrh datové struktury.
  - **Konceptuální datový model** – zobecnění konkrétní implementace datové struktury v relační databázi, lze jej přenášet do různých implementačních prostředí, příkladem je entitně - relační model (ER diagramy).

Příklad datového modelu

Datový model slouží pro návrh datové struktury. Jedná se o konceptuální model, který je velice podobný klasickým ER diagramům používaným pro návrh struktury relačních databází nebo UML Class diagramům používaných v objektové analýze a návrhu.

Model je tvořen sadou entit, které jsou mezi sebou provázány pomocí vazeb s příslušnou kardinalitou (násobností).

- Entity - určitou skupinu objektů reálného světa, které se označují jako instance entity. Všechny instance dané entity se nazývají populací a vyznačují se stejnou vnitřní datovou strukturou, která se vyjadřuje množinouatributů.
  Každá entita je popsána svým názvem a sadou atributů, každý atribut má datový typ.
- Vazby mezi entitami - na vazbu můžeme pohlížet jako na dvě vazby v opačných směrech. V tomto smyslu se hovoří o takzvaných rolích, které představují pohled na danou vazbu ve směru od jedné entity ke druhé.
- Kardinalita - představuje omezení v počtu instancí druhé entity, které mají vztah s jakoukoliv instancí první entity.



Transakce

Transakce je posloupnost operací (DML příkazů), které převedou datové schéma z jednoho konzistentního stavu do druhého (zpřístupňuje a aktualizuje data).

- **Atomic** (atomičnost) – transakce se celá provede nebo se celá zruší
- **Consistency** (konzistence) – po dokončení transakce je databáze konzistentní
- **Isolation** (izolovanost) – různé transakce o sobě vzájemně nevědí
- **Durability** (trvanlivost) – po ukončení transakce jsou data trvale uložena

Více transakcí může být spouštěno současně, může však dojít k uváznutí (**deadlocku**).
Chronologické pořadí provádění instrukcí souběžných transakcí je předem určeno pomocí **plánu**.

Každá transakce může nabývat těchto stavů:

- **Aktivní**
- **Částečně potvrzená**
- **Chybující**
- **Zrušená** - je možné ji znovu spustit (nedošlo-li k logické chybě) nebo zamítnout.
- **Potvrzená**.

Transakční zpracování

- Předpoklad
  - Pokud T začíná v konzistentním stavu a T běží samostatně
  - → T končí také v konzistentním stavu
- Správnost
  - Pokud přerušíme běžící transakci, DB zůstane konzistentní
  - Každá transakce vidí konzistentní DB

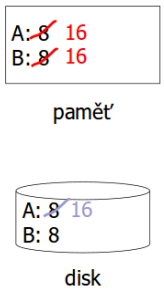
Porušení konzistence

- Příčiny
  - Chyba transakce
  - Chyba DB systému
  - Výpadek hardwaru
    - Např. výpadek disku poruší stav účtu na disku
  - Sdílení dat
    - Např. T1: přidej 10% platu programátorům
      T2: změň programátor → systémový analytik

Příklad transakce T1

T1: Read (A,t);
$t \leftarrow t \cdot 2$;
Write (A,t);
Read (B,t);
$t \leftarrow t \cdot 2$;
Write (B,t);
Output (A);
Output (B);

výpadek!



Transakce

- Základní operace
  - Input (x): blok obsahující x → paměť
  - Output (x): blok obsahující x → disk
  - Read (x,t): a. *Input(x)*, pokud je potřeba
    b. t ← hodnota x v bloku
  - Write (x,t): a. *Input(x)*, pokud je potřeba
    b. hodnota x v bloku ← t

Transakce

- Atomičnost
  - Řešení problému nedokončených transakcí
  - Provedení všech akcí transakce nebo vůbec žádné
- Jak implementovat?
  - Logování provedených změn
    - Tj. vytvoření žurnálu (souboru se záznamy o změnách)

Žurnálování

- Běh transakce produkuje záznamy o změnách do žurnálu
  - Začátek, konec, uložení, aktualizace, …
- Využití
  - Výpadek systému → znovu provedení změn podle žurnálu
  - Obnova dat z archivu → znovu provedení změn podle žurnálu

Žurnálování

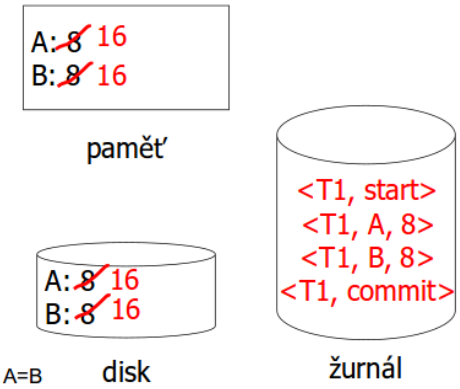
- Při obnově
  - Některé transakce se provedou znovu
    - REDO
  - Některé transakce se zruší
    - UNDO

Undo logging

- „Zrušení podle žurnálu“
- Pokud není 100% jistota uložení změn dokončené transakce
  - Pak se změny podle žurnálu odstraní
    - Tj. obnovení předchozího stavu DB
  - → Transakce nikdy nebyla spuštěna
- Vlastnost
  - Změny prováděné transakcí jsou *ihned ukládány* na disk

Undo logging: Transakce T1

T1: Read (A,t);
$t \leftarrow t \cdot 2$;
Write (A,t);
Read (B,t);
$t \leftarrow t \cdot 2$;
Write (B,t);
Output (A);
Output (B);



Pozn: vyžadujeme platnost A=B

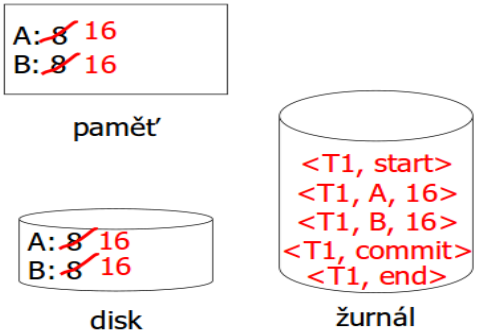
- Pravidla
  1. Pro každou akci vytvoř v žurnálu záznam obsahující starou hodnotu
  2. Před změnou *x* na disku musí být na disku záznamy žurnálu týkající se *x*
     - Tzv. *write ahead logging* (WAL)
  3. Před vytvořením záznamu <commit> v žurnálu musí být všechny zápisy transakce uloženy na disku.

Redo logging

- „Znovu provedení podle žurnálu“
- Vlastnost
  - Změny provedené transakcí jsou *ukládány později*
    - Tj. při potvrzení (commit)
    - Ušetření zápisů na disk
  - Při obnově jsou ignorovány nedokončené transakce
  - Vyžaduje uložení žurnálu před uložením změn v DB.
    - Žurnálují se nové hodnoty

Redo logging: Transakce T1

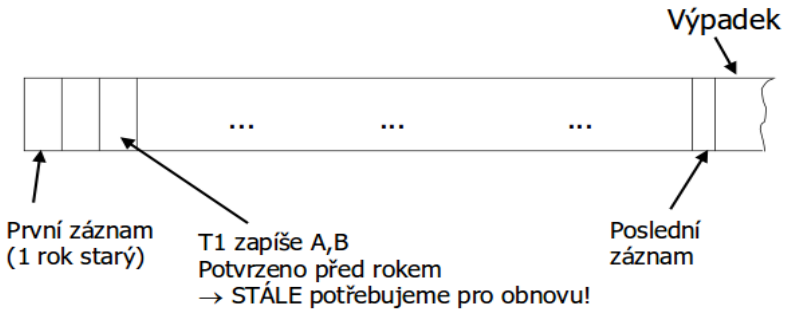
T1: Read (A,t);
t ← t · 2;
Write (A,t);
Read (B,t);
t ← t · 2;
Write (B,t);
Output (A);
Output (B);



- Pravidla
  1. Pro každou akci vytvoř v žurnálu záznam obsahující novou hodnotu
  2. Před změnou *x* na disku (v DB) musí být na disku všechny záznamy žurnálu (včetně commit) pro transakci měnící *x*
     1. Ulož záznamy žurnálu na disk
     2. Ulož změněná data na disk
     3. Zapiš end do žurnálu

Redo logging – obnova po výpadku

- Obnova je velice pomalá

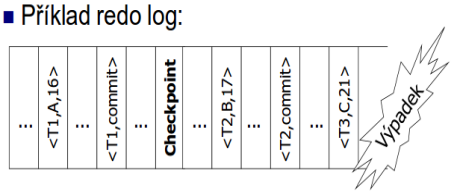


Kontrolní body

- Jednoduchý checkpoint
  - Během vytváření nesmí žádné transakce běžet
  - Významné snížení průchodnosti DB
- Řešení
  - Průběžné kontrolní body (Non-quiescent Checkpoint)
    - Evidence nedokončených transakcí
    - UNDO/REDO logging:
      - všechny změněné záznamy (bloky) jsou uloženy na disk

- Implementace:
  1. Přestaň přijímat nové transakce
  2. Počkej na ukončení všech transakcí
  3. Ulož všechny záznamy žurnálu na disk
  4. Ulož všechny bloky na disk (DB)
  5. Zapiš záznam „checkpoint“ na disk do žurnálu
  6. Pokračuj ve zpracování transakcí

- Postup při obnově
  - Vyhledej poslední kontrolní bod
  - Od něj proveď obnovu
- Příklad redo log:



Žurnálování

- Hlavní nevýhody
  - Undo logging
    - Ze zálohy DB nelze vytvořit aktuální stav DB
  - Redo logging
    - Všechny modifikované bloky musíme držet v paměti až do potvrzení (commit) transakce
  - Zápisy na disk jsou vynuceny pravidly žurnálu a ne přístupem k datům
- Řešení: Undo/Redo logging
  - Záznam v žurnálu obsahuje starou i novou hodnotu: <T_i, x, nová X, původní X>

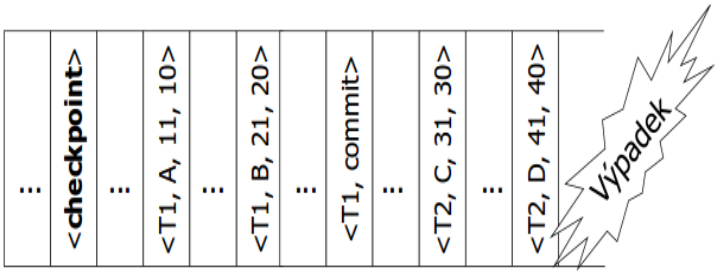
Undo/Redo logging

- Pravidla
  - Hodnota *X* může být uložena před i po potvrzení T_i
  - Před zapsáním hodnoty *X* na disk, musí být na disk zapsán odpovídající záznam žurnálu (WAL)
  - Ulož žurnál při potvrzení transakce

- Obnova
  - Ukončené transakce zopakujeme (redo) od začátku
  - Nedokončené transakce vrátíme (undo) od konce

Undo/Redo logging – obnovení

- Příklad undo/redo log:



3. Metadata

- Metadata (z řeckého meta- = mezi, za + latinského data = to, co je dáno) jsou **strukturovaná data o datech**.
  Příkladem je katalogizační lístek v knihovně, obsahující data o původu a umístění knihy: jsou to data o datech v knize uložené na lístku.
  Metadata mohou sloužit např. **k snadnému vyhledávání**.
- Všechna data v datovém skladu jsou doplněna o metadata. Popisují:
  - tabulky (např. názvy sloupců a datové typy),
  - pohledy
  - procedury
  - uživatele nebo jakýkoliv jiný objekt v databázi.
- Bez metadat by datový sklad nemohl fungovat.

4. Datové sklady

- Datový sklad je zvláštní typ relační databáze, která umožňuje řešit úlohy zaměřené převážně na analytické dotazování nad rozsáhlými soubory dat.
- Jde o centrální úložný prostor, kam jsou ukládána data z různých zdrojů a toto vkládání je logicky organizované.
- Rozdíly oproti relační databázi:
  - **orientace na subjekt** – jasná vnitřní separace funkčních celků, běžná je redundance dat
  - integrovanost – data z různých zdrojů jsou spojená podle logického významu (nikoliv podle původu)
  - **nízká proměnlivost** – data jsou nahrávána zpravidla v dávkách a později nejsou modifikována
  - **historizace** – data mohou být uchovávána v různých časových verzích (nikoliv pouze v aktuálním stavu)
- Technologické charakteristiky:
  - datový sklad musí obsahovat nástroj pro nahrávání dat z různých datových zdrojů, tyto zdroje mohou mít různé datové formáty a různé fyzické umístění
  - datový sklad ukládá data s ohledem na co nejlepší a nejrychlejší provádění složitých dotazů – proto je pro uložení dat používána často technologie OLAP (technologie uložení dat v databázi, která umožňuje uspořádat velké objemy dat tak, aby byla data přístupná a srozumitelná uživatelům zabývajícím se analýzou obchodních trendů a výsledků)
- Data v datovém skladu jsou z logického pohledu členěna do schémat (každé schéma odpovídá jedné analyzované funkční oblasti), jádro každého schématu tvoří jedna nebo několik faktových tabulek (v nich jsou uložena vlastní analyzovaná data) a ty jsou pomocí cizích klíčů spojeny s dimenzemi (tabulky, které obsahují seznamy hodnot sloužících ke kategorizaci a třídění dat ve faktových tabulkách).

Závěr

http://statnice.dqd.cz/mgr-szz:in-ins:4-ins

Slidy k PA152