Relacia = mnozina usporiadanych n-tic  
Relacna schema = n-tica, napr: R=(A,B,C,D) / A,B,C,D su atributy relacie  
Atribut = stlpec, pomenovanie stlpca  
Domena = mnozina povolenych hodnot pre dany stlpec(atribut), kazdy atribut ma svoju domenu

Nech mame mnozinu domen D1,D2,…,Dn zodpovedajucu atributom relacnej schemy R=(A1,A2,…An), potom relacia r je podmnozinou r ⊆ D1 x D2 x … x Dn

Nech K ⊆ R => K je superkluc R, ak hodnoty K su dostatocne na identifikovanie kazdej n-tice zo vsetkych moznych relacii r(R).

Relacny model = matematicka formalizacia toho, co bezne nazyvame tabulky databazy.  
Relacna algebra = sada operacii, ktora sluzi na pracu s datami v relacnom modele.

Vstupom kazdej operacie relacnej algebry je jedna alebo viac relacii a vystup je opat relacia.

Zakladne operacie relacnej algebry:

-SELEKCIA (σ): unarna operacia: σp(r)={t | t ∊ r and p(t)}, p je predikat, vrati relaciu s n-ticami(riadkami), ktore splnaju predikat p, predikat moze byt zlozeny z viacerych podmienok prepojenych: and,or,not

-PROJEKCIA (Π): unarna operacia: ΠA1,A2,…,Ak(r), vracia relaciu s len definovanymi atributmi(stlpcami) a automaticky odstranuje duplicitne riadky

-ZJEDNOTENIE (∪): binarna operacia: r ∪ s={t | t ∊ r or t ∊ s }, tato operacia ma 2 podmienky:  
1. Relacie r a s musia mat rovnaku aritu (rovnaky pocet atributov(stlpcov))  
2. Domeny atributov musia byt kompatibilne (napr. 2. Stlpec relacie r musi mat rovnaky typ hodnot ako 2. stlpec relacie s)

-ROZDIEL (-): binarna operacia: r – s={t | t ∊ r and t ∉ s}, ma rovnake podmienky ako zjednotenie

-KARTEZSKY SUCIN: binarna operacia: r x s={t q | t ∊ r and q ∊ s}, dvojice kazdy riadok z prvej relacie  
 s kazdym riadkom z druhej relacie => ak r ma m riadkov a s n riadkov, vysledna relacia ma m \* n riadkov  
 ak R ∩ S ≠0 => je nutne premenovanie atributov

-PREMENOVANIE (ρ): unarna operacia  
 ρx(r): premenuje relaciu r na x  
 ρx(A1,A2,…,An)(r): premenuje relaciu r na x s atributmi premenovanymi na A1,A2,…,An

-PRIENIK (∩): binarna operacia: r ∩ s={t | t ∊ r and t ∊ s}, rovnake podmienky ako zjednotenie  
 r ∩ s=r-(r-s) = s – (s-r)

-PRIRODZENE SPOJENIE: binarna operacia  
 Nech mame dve relacie so schemami R a S => r ⨝ s ma schemu R ∪ S, najprv sa zostroji kartezsky sucin  
 tychto relacii a do vyslednej relacie sa pridaju riadky tr a ts take, kde tr a ts maju rovnake hodnoty v   
 atributoch(stlpcoch) R ∩ S (spolocnych atributoch)

-LAVE VONKAJSIE SPOJENIE: binarna operacia: to iste co PRIRODZENE SPOJENIE, akurat do vyslednej   
 relacie prida este vsetky riadky z prvej relacie (z relacie nalavo), ktore sa nezhodli v spolocnom attribute  
 so ziadnym riadkom z druhej relacie (z relacie napravo)

-PRAVE VONKAJSIE SPOJENIE: binarna operacia: to iste co LAVE VONKAJSIE SPOJENIE, ale do vysledku  
 pridava riadky z druhej relacie (z pravej relacie), ktore sa nezhodli so ziadnym riadkom z prvej relacie  
 v spolocnom attribute

-UPLNE VONKAJSIE SPOJENIE: binarna operacia: spojenie LAVEHO a PRAVEHO VONKAJSIEHO SPOJENIA,  
 do vysledku pridava riadky z oboch relacii, ktore sa v spolocnom attribute nezhodli so ziadnym riadkom  
 z relacie druhej, resp. prvej

NULL hodnota: znamena neznamu hodnotu, alebo ze hodnota neexistuje, vysledok kazdeho   
aritmetickeho vyrazu obsahujuceho null je null, agregacne funkcie hodnotu null ignoruju, pri   
eliminacii duplikacie a pri zgrupovani sa s hodnotou null zachadza ako s kazdou inou, dvakrat spomenuta  
null sa berie ako ta ista hodnota, hodnota null patri do kazdej domeny

Agregacna funkcia: berie subor hodnot a ako vysledok vrati jednu hodnotu

-AGREGACNA OPERACIA: G1,G2,…,GnγF1(A1),F2(A2),…Fn(An) G1,G2,…,Gn – atributy, podla ktorych sa zlucuju riadky(nemusi sa podla ziadneho => moze byt prazdne)  
 F1,F2,…,Fn – agregacna funkcia  
 A1,A2,…,An – nazov atributu

Agregacne funkcie: avg (priemer), min (minimalna hodnota), max (maximalna hodnota), sum (sucet hodnot), count (pocet hodnot)

Modifikacia databazy je mozna pomocou tychto 3 operacii:  
1. Deletion: r ← r – E, kde r je relacia a E je dotaz relacnej algebry, moze odstranit len cele n-tice  
2. Insertion: r ← r ∪ E, E moze byt konstatne zadana n-tica, alebo dotaz relacnej algebry   
3. Updating: r ← Π F1,F2,…,FI(r), kde kazde FI je bud i-ty atribut, ak tento atribut neupdateujeme, alebo  
je to vyraz obsahujuci konstanty a atributy relacie r, cim ziskame nove hodnoty pre dany atribut  
tieto operacie su vyjadrene operatorom: ←

Typy domen v SQL:

-char(n): fixne velky retazec, uzivatelom definovana dlzka -> n  
 -varchar(n): premenlivo velky retazec, s max. dlzkou n  
 -int: cele cisla, obmedzene pocitacom  
 -smallint  
 -numeric(p,d): fixne desatinne cislo, s p miestami pred desatinnou ciarkou a d miestami za nou  
 -real, double precision  
 -float(n): desatinne cislo, s minimalne n miestami

SQL relacia je definovana pomocou vytvorenia tabulky:  
create table r (A1 D1, A2 D2, … An Dn) , kde r je nazov relacie(tabulky), A su atributy, D su ich domeny

drop table r: vymaze tabulku r a jej obsah  
delete from r: vymaze obsah tabulky r ale tabulku zanecha  
alter table r add A D: do tabulky r pride novy atribut(stlpec) A s domenou D, pre kazdy existujuci riadok v tabulke sa prida do noveho stlpcu A hodnota null  
alter table r drop A: zmaze sa atribut A tabulky r

Zakladna stavba SQL dotazu:

select A1,A2,…,An  
from r1,r2,…,rn  
  
kde A su atributy, r su relacie, P je predikat, vysledok SQL dotazu je relacia  
select = projekcia, SQL povoluje duplicitne riadky => pre ich odstranenie je potrebne pouzit distinct za slovom select

Natural Join = PRIRODZENE SPOJENIE  
Pr: select name, course\_id  
 from instructor,teaches  
 where instructor.id=teaches.id;

Sa da zapisat takto pomocou natural join:  
select name,course\_id  
from instructor natural join teaches;

Rename = PREMENOVANIE  
stary nazov **AS** novy nazov: premenuje relaciu stary nazov na novy nazov

Operator LIKE: pouziva sa na porovnavanie retazcov, pouziva dva specialne znaky  
1. % - moze byt akykolvek retazec  
2. \_ - moze byt akykolvek znak  
Napr: %ab% - hocijake slovo obsahujuce ab, \_ab\_ - 4 mieste slovo obsahujuce ab uprostred

Operator ORDER BY: usporiadava hodnoty  
Pr: order by A1,A2,…,An desc – usporiada riadky podla jednotlivych atributov, desc – klesajuco, defaultne je nastavene asc – stupajuco

Where klauzula between:  
Pr: where salary between 90000 and 100000

Duplicity:  
1. σp(r): ak je c1 kopii n-tice(riadku) t1 v relacii r, a ak t1 splna predikat p => vo vysledku selekcie je c1 kopii n-tice(riadku) t1  
2. ΠA(r): projekcia automaticky odstranuje duplicity => vo vysledku nebude viac ako jeden rovnaky riadok  
3. r1 x r2: ak je c1 koppi n-tice(riadku) t1 v relacii r1 a c2 kopii n-tice(riadku) t2 v relacii r2 => vo vysledku kartezskeho sucinu r1 x r2 bude c1\*c2 kopii n-tice(riadku) t1 t2

Mnozinove operacie:  
UNION = ZJEDNOTENIE  
INTERSECT = PRIENIK  
EXCEPT = ROZDIEL  
Kazda z nich automaticky odstranuje duplicity, pre zachovanie duplicit treba pouzit UNION all, INTERSECT all alebo EXCEPT all

Nech n-tica(riadok) sa vyskytuje m krat v relacii r a n krat v relacii s, potom:  
n-tica(riadok) sa vyskytne m+n krat vo vyslednej relacii operacie r union all s  
n-tica(riadok) sa vyskytne min (m,n) krat vo vyslednej relacii operacie r intersect all s  
n-tica(riadok) sa vyskytne max (0, m-n) krat vo vyslednej relacii operacie r except all s

Klauzula some:  
F <compare> r ⬄ ∃t ∊ r take, ze (F <compare> t)  
Pr: (5 < some {0,5,6}) = true, (5 < some {0,5}) = false, (5 = some {0,5}) = true, (5 ≠ some {0,5})=true  
(=some) == in  
(≠some)≠≠ not in

Klauzula all:  
F <compare> all r ⬄ ∀t ∊ r (F <compare> t)  
Pr: (5 < all {0,5,6})=false, (5 < all {6,10})=true, (5 = all {5,6})=false, (5 ≠ all {4,6})=true  
(≠all) == not in  
(=all) ≠≠ in

Test prazdnej relacie pomocou klauzuly exists:  
exists r ⬄ r ≠ {}  
not exists ⬄ r = {}

Databaza:  
1. Subor objektov(entit)  
2. Vztahy medzi objektami(entitami)

Entita = object, ktory existuje a je rozlisitelny od ostatnych, napr: konkretny clovek, spolocnost…  
Entity maju atributy, napr clovek ma atributy: meno, priezvisko, vek…  
Sada entit(entity set): sada entit, rovnakeho typu, s rovnakymi vlastnostami, napr: sada vsetkych ludi…  
Relationship(vztah): spojenie medzi odlisnymi entitami, objektami  
Relationship set(sada vztahov): matematicka relacia medzi n>=2 entitami, kde kazda entita je zo sady  
entit {(e1,e2,…,en) | e1 ∊ E1, e2 ∊ E2, … en ∊ En} => (e1,e2,…,en) je relationship(vztah)  
Entita je reprezentovana sadou atributov, ktore maju popisne vlastnosti a ma ich kazdy clen sady entit.

Typy atributov:  
1. Simple(jednoduche), Composite(zlozene, rozvetvene) napr: atribut name a podatributy first name a last name  
2. Single valued a Multivalued  
3. Derived: mozu byt odvodene z ostatnych atributov, napr: vek, datum narodenia

Mapping Cardinality Constraints: cislo, ktore vyjadruje pocet entit, s ktorymi moze byt spojena ina entita pomocou relationship set

Pre binarne relationship set to musi byt jedna z tychto 4 typov:  
1. One to one  
2. One to many  
3. Many to one  
4. Many to many

Super key(superkluc) sady entit je sada jedneho alebo viacerych atributov ktorych hodnoty jednoznacne  
urcuju kazdu entitu.

Candidate key sady entit je minimalny super key(napr: atribut ID), candidate keys moze byt viac, jeden z kandidatnych klucov je vybrany za primary key.

Relationship set super key: kombinacia primarnych klucov zucastnenych sad entit v relationship set  
Pr: Majme sadu entit student: a sadu entit ucitel:  
 ID Meno ID Meno  
 422 Peter 189 Fero  
 464 Ondrej 874 Juro  
 244 Elena

=>

422(Peter) advisor 874(Juro)  
student entity relationship set ucitel entity

Je priklad spojenia dvoch sad entit, nech pre sadu entit student je primarny kluc=ID a pre sadu entit ucitel je primarny kluc=ID => relationship super key = (student\_ID, ucitel\_ID)

E-R diagram

advisor

**ucitel**

ID  
 meno  
 plat

**student**

ID  
 meno  
 kredity

Obdlzniky reprezentuju sadu entit  
Diamand reprezentuje relationship set  
Atributy entit su vpisane v obdlznikoch  
Podciarknute atributy indikuju primary key

date

advisor

**ucitel**

ID  
 meno  
 plat

**student**

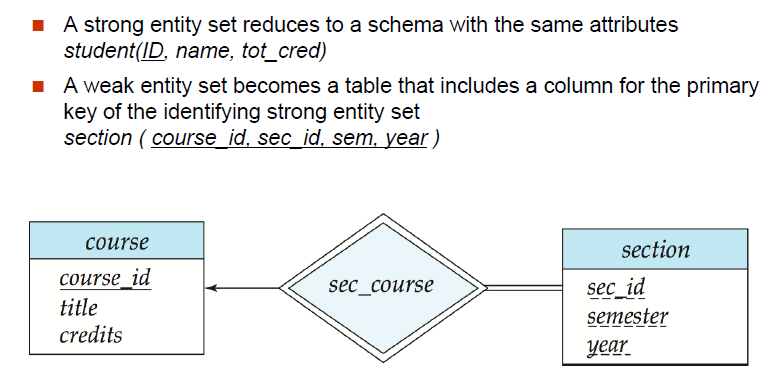
ID  
 meno  
 kredity

Relationship set s atributom

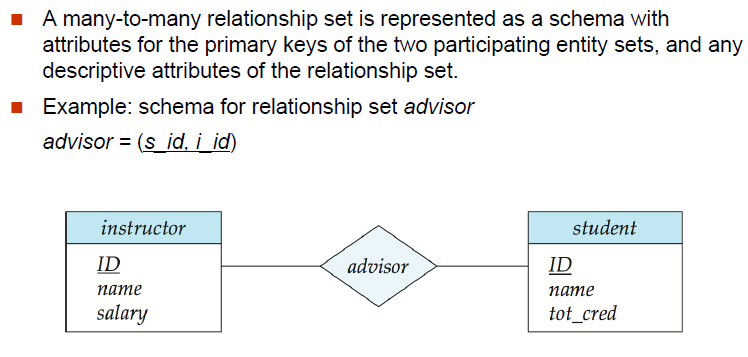
Participation of an Entity Set in a Relationship set:  
1. Total participation: kazda entita v entity set je aspon v jednom spojeni(vztahu) v relationship set  
2. Partial participation: niektore entity niesu v ziadnom spojeni(vztahu) v relationship set

Weak Entity set: sada entit, ktora nema primarny kluc  
Weak entity set musi byt vo vztahu identifying entity set cez totalne one-to-many spojenie z identifying  
do weak entity set  
Identifying relationship je znazornene pomocou dvojiteho diamantu.  
Discriminator of weak entity: subor atributov, ktore sa lisia na vsetkych entitach weak entity set, znacime ho prerusovanym podciarknutim

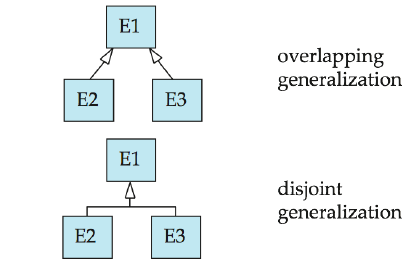
Entity sets a relationship sets mozu byt vyjadrene relacnou schemou, ktora reprezentuje obsah databazy



Reprezentacia Entity sets with simple attributes



Reprezentacia Relationship sets



Bucket – uloziskova jednotka, obsahuje jeden alebo viacero zaznamov (records)  
Hash function – pouziva san a ziskanie bucket zaznamu priamo cez search-key value v hash file organization, hash funkcia h, je funkcia od mnoziny vsetkych search-key hodnot K ku mnozine vsetkych adries bucketov B, hashovacia funkcia sa pouziva na lokalizaciu zaznamov pre pristup, vkladanie a mazanie, zaznamy s roznymi search-key hodnotami mozu byt ulozene do toho isteho bucketu => cely bucket musi byt zaradom prehladany, pre najdenie daneho zaznamu

Najhorsia hashovacia funkcia namapuje vsetky search-key hodnoty do jedneho bucketu => to sposobi  
ze pristupovy cas bude umerny poctu search-key hodnot v subore.  
Idealna hashovacia funkcia je nahodna, takze kazdy bucket bude mat rovnaky pocet search-key hodnot

Bucket Overflows – pretecenie bucketov, moze vzniknut:  
1. Nedostatocnymi bucketmi  
2. Asymetrickym rozdelovanim zaznamov(records)  
 - viacere zaznamy, maju tu istu search-key hodnotu  
 - vybrana hashovacia funkcia, vytvara nerovnomerne rozdelenie klucovych hodnot (key values)

Overflow chaining – overflow buckety daneho bucket su zretazene v linked liste, tato schema sa nazyva closed hashing  
Open hashing – nepouziva overflow buckety, nieje vhodna pre aplikaciu v databazach

Staticke hashovanie – funkcia h mapuje search-key hodnoty na pevne danu mnozinu bucket adries,   
je to nevyhodne, casto je pocet bucketov nedostatocny a dochadza k pretekaniu, alebo pri velkom pocte  
bucketov sa zbytocne plytva miestom

Dynamicke hashovanie – vyhodne pre databazy, ktore rastu a zmensuju sa  
Extendable hashing – forma dynamickeho hashovania, generuje hodnoty vo velkom rozsahu (typicky 32-bitove hodnoty), hocikedy pouzije len prefix hashovacej funkcie pre indexovanie do tabulky adries bucketov, bucket address table size = 2i (spociatku i=0), hodnota I rastie a klesa ako rastie a zmensuje sa databaza, viac riadkov v tabulke adries bucketov moze ukazovat na ten isty bucket, => skutocny pocet bucketov je < 2i, pocet bucketov sa dynamicky meni v zavislosti od spajania a rozdelovania bucketov

Mechanizmus indexovania sa pouziva pre rychlejsi pristup k pozadovanym datam.

Search key – atribut az mnozina atributov, pouzivana pre vyhladanie zaznamov v subore.

Index file – pozostava zo zaznamov zvanych index entries a obsahuje search-key a ukazatel na zaznam v subore:

Indexove subory su zvacsa omnoho mensie ako subory so zaznamami.  
Rozlisujeme dva zakladne druhy indexov v indexovom subore:  
1. Ordered indices – search-key hodnoty su zoradene  
2. Hash indices – search-key hodnoty su rozdelene rovnomerne cez bucket pomocou hash funkcie

V Ordered indicies: 1. Primary index – index, ktoreho search-key urcuje postupne zoradenie zaznamov  
 2. Secondary index – index, ktoreho search-key urcuje ine ako postupne zoradenie  
 zaznamov

Dense index – indexovy zaznam pre kazdu search-key hodnotu v subore, pre kazdy search-key len raz  
Sparse index – indexovy zaznam len pre niektore search-key hodnoty v subore, pouzitelne pri suboroch  
so zoradenymi zaznamami podla search-key hodnot. Pre najdenie zaznamu so search-key hodnotou K, musime: Najst index zaznam s najvacsou search-key hodnotou mensou ako K a od tohto zaznamu postupne prehladavat subor.

Multilevel index – ak sa primary index nezmesti do pamate => pristup k zaznamom sa stane “drahym”, preto primary index san a disku ulozi ako sekvencny subor a nad nim sa zostroji sparse index =>  
outer index = sparse index primary indexu  
inner index = primary index subor  
Ak sa ani outer index nezmesti do pamate, vytvori sa dalsi level sparse indexu, az kym sa nezmesti do pamate, pri vkladani a mazani zaznamov, musia byt aktualizovane indexy na vsetkych urovniach

**Query Processing**

1.Parsing and translation  
-prelozenie dotazu do jeho vnutornej formy, nasledne do relacnej algebry  
-parser skontroluje syntax a overi existenciu relacii

2.Optimization  
-najdenie najlacnejsie planu pre vykonanie dotazu  
-vyraz relacnej algebry moze mat viacero ekvivalentnych vyrazov (foriem)  
-kazda operacia relacnej algebry moze byt vyhodnotena jednym z viacerych algoritmov  
 => vyraz relacnej algebry moze byt vyhodnoteny roznymi sposobmi  
-Komentovany vyraz, opisujuci strategiu vyhodnotenia sa nazyva evaluation-plan  
-zo vsetkych evaluation-plans vybera ten s najnizsou cenou   
-cena je odhadovana na zaklade statistickych udajov z databazoveho katalogu

3.Evaluation  
-stroj pre vyhodnotenie dotazu berie na vstupe evaluation-plan, spusti plan a vrati vysledok dotazu

Cena = uplynuty cas medzi prijatim dotazu a odpovedou na dotaz   
 -na cenu ma vplyv viacero faktorov: pristup na disk, CPU, sietova komunikacia

Pristup na disk(disk accesses)  
-dominantny faktor, urcujuci cenu  
-relativne lahko odhadnutelny, berie do uvahy:  
1.Pocet hladani => priemernu cenu vyhladania  
2.Pocet precitanych blokov => priemerna cena precitania bloku  
3.Pocet zapisanych blokov => priemerna cena zapisania bloku

Cena zapisu bloku je vyssia ako precitania bloku, lebo data su po zapise precitane, ci sa zapisali spravne

tT – cas prenosu jedneho bloku  
tS – cas jedneho vyhladania

Cena prenosu b blokov a S vyhladani = b \* tT + S \* tS

Operacia selekcie: scan suboru, prehladavanie suboru

Algoritmus A1(linearne vyhladavanie)  
- scanuje kazdy blok suboru a testuje vsetky zaznamy, ci splnaju podmienky selekcie  
-odhad ceny = br prenosov bloku + 1 hladanie (br = pocet blokov obsahujucich zaznamy relacie r)  
-ak sa selekcia vykonava na klucovom atribute => cena = (br/2) prenosov bloku + 1 hladanie

Selekcia pomocou indexov  
-vyhladavacie algoritmy, pouzivajuce indexy  
-A2(primary index, equlity on key) - vybere jeden zaznam ktory splna: Cost = (hi + 1) \* (tT + tS)  
-A3(primary index, equality on nonkey)  
 -vybere viacero zaznamov  
 -zaznamy budu na po sebe iducich blokoch  
 -Cost = hi \* (tT + tS) + tS + tT \* b (b = pocet blokov obsahujucich zhodne zaznamy)  
-A4(secondary index, equality on nonkey)  
 -vybere jeden zaznam, ak search-key je kandidatny kluc (candidate key)  
 Cost = (hi + 1) \* (tT + tS)  
 -vybere viacero zaznamov, ak search-key nieje kandidatny kluc (candidate key)  
 -kazdy z n zhodnych zaznamov, moze byt na inom bloku  
 -Cost = (hi + n) \* (tT + tS)

Selekcia zahrnajuca porovnavania  
-moze zahrnat selekcie typu: δA ≤ V(r), δA ≥ V pouzitim linearneho prehladavania, alebo pouzitim indexov:  
-A5(primary index, comparison), kde relacia je zoradena na atribute A  
 -pre δA ≥ V(r) pouzi index pre najdenie prvej ntice ≥ V a scanuj relaciu postupne odtialto  
 -pre δA ≤ V(r) prehladavaj(scanuj) relaciu postupne kym prva ntica > v, nepouziva index  
-A6(secondary index, comparison)  
 -pre δA ≥ V(r) pouzi index, pre najdenie prveho zaznamu ≥ v a scanuj index postupne odtialto  
 pre najdenie pointerov(ukazatelov) na zaznamy  
 -pre δA ≤ V(r) scanuj len leaf pages index finding pointers k zaznamom, kym prvy zaznam > v

1.Conjunction: δ01 ᴧ 02 ᴧ … ᴧ 0n(r)  
-A7(conjunctive selection using one index)  
 -vyber kombinacie 0i a algoritmov A1 az A7, pre najnizsiu cenu vykonania

-A8(conjunctive selection using composite index)  
 -pouzitie zlozenych (multiple-key) indexov ak je to mozne

-A9(conjunctive selection by intersection of identifiers)  
 -vyzaduje indexy s ukazatelmi na zaznamy  
 -pouzitie odpovedajuceho indexu pre kazdu podmienku, a vyber prieniku zo vsetkych ziskanych  
 mnozin ukazatelov na zaznamy, (prienik zaruci vyber zaznamov splnajucich vsetky podmienky)  
 -vyber vsetkych zaznamov zo suboru

2.Disjunction δ01 ᴠ 02 ᴠ … ᴠ 0n(r)  
-A10(disjunctive selection by union of identifiers)  
 -pouzitelny ak vsetky podmienky maju dostupne indexy(indices), inak linearny scan  
 -pouzitie odpovedajuceho indexu pre kazdu podmienku, a vyber zjednotenia zo vsetkych  
 ziskanych mnozin ukazatelov na zaznamy  
 -vyber vsetkych zaznamov zo suboru

3.Negation: δ -,0(r)  
-pouzitie linearneho prehladavania suboru  
-ak malo zaznamov splna -,0 a index je aplikovatelny na 0 => najdi vyhovujuce zaznamy pomocou indexu  
 a vyber ich zo suboru

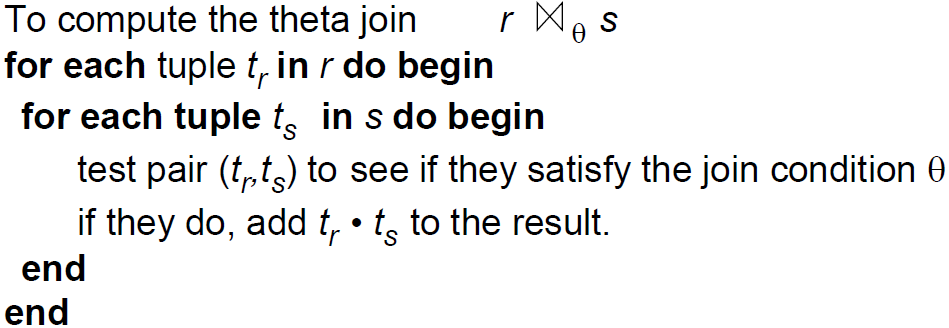
**Sorting (Triedenie)**

-vybudovanie indexov nad relaciou a nasledne pouzitie pre citanie relacie v zoradenom poradi  
-vedie k jednemu pristupu na diskovy blok pre kazdy n-ticu  
-pre relacie, ktore sa zmestia do pamate, moze byt pouzity quicksort  
-pre relacie, ktore sa nezmestia do pamate, sa pouziva external sort-merge

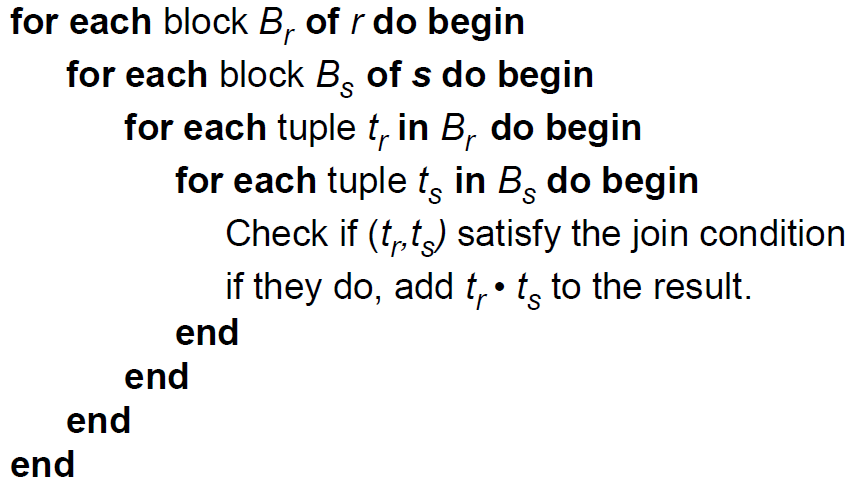
External sort-merge  
Nech M je velkost pameti v strankach (odpovedajuce blokom)  
1.Vytvorenie sorted runs, nech i=0  
 a) Nacitaj M blokov relacie do pamate  
 b) Zotried nacitane bloky  
 c) Uloz zoradene data do davky(run) Ri a zvys i

2.Spoj davky(runs), predpoklad i < M  
 a) Pouzi i blokov pamate pre nacitanie(buffer) vstupnych davok a 1 blok ako vystup,  
 nacitaj prvy blok kazdej davky(runu) do jeho buffer page  
 b) Opakuj  
 1.Vyber prvy zaznam (v zoradenom poradi) zo vsetkych buffer pageov  
 2.Vypis zaznam na vystupny buffer, ak je plny, zapis na disk  
 3. Vymaz zaznam zo vstupneho buffer pageu, ak buffer page je potom prazdny   
 => nacitaj nasledujuci blok davky(runu) do buffer  
 c) az kym vsetky vstupne buffer page su prazdne

Operacia Join  
-Niekolko roznych algoritmov pre implementaciu spojenia (join)  
 -vnorene cykly (nested loop join)  
 -blokovane vnorene cykly (blocked nested loop join)  
 -indexovane vnorene cykly (indexed nested loop join)  
 -zlucovacie spojenie (merge-join)  
 -hesovane spojenie (hash-join)  
-vybrane su na zaklade odhadu ceny

Vnorene cykly (Nested-Loop Join)  


-r je vonkajsia relacia a s je vnutorna relacia joinu  
-nevyzaduje ziadne indexy a moze byt pouzita s akoukolvek spojovacou podmienkou  
-draha metoda, lebo prechadza kazdu dvojicu n-tic z oboch relacii  
-ak sa mensia relacia zmesti do pamate, pouzi ju ako vnutornu relaciu=> cena = br+bs diskovych pristupov  
-v najhorsom pripade, ak sa zmesti do pamate len jeden blok kazdej relacie => cena = nr\*bs+br disk. pr.

Blokovane vnorene cykly (Blocked nested loop join)  


-kazdy blok vnutornej relacie, je sparovany s kazdym blokom vonkajsej relacie  
-najhorsi pripad => cena = br \* bs + br prenosov blokov + 2 \* br hladani  
-najlepsi pripad => cena = br + bs prenosov blokov + 2 hladania

Indexovane vnorene cykly (Indexed Nested-Loop Join)  
-vyhladavanie indexom moze byt pouzite miesto prehladavaniu suboru ak  
 1.spojenie (join) je equi-join alebo natural join  
 2.na vnutornej relacii join atributu je dostupny index  
-pre kazdu n-ticu tr vo vonkajsej relacii r, sa pouzije index pre vyhladanie n-tic vo vnutornej  
 relacii s, ktore splnaju podmienku spojenia s n-ticou tr

Zlucovacie spojenie (Merge-join)  
1.obe relacie zorad na ich join atribute   
2.zlucenie zoradenych relacii  
 1.Krok spojenia je podobny spajaniu pri sort-merge algoritme  
 2.Hlavny rozdiel je v zaobchadzani s duplikatmi v join atributoch, kazdy par, s rovnakou  
 hodnotou na join atribute musi byt vypisany

-kazda n-tica je citana len raz => kazdy blok je citany len raz => cena = br + bs citania blokov

Hesovacie spojenie (Hash-join)  
-aplikovatelne pre equi-joins a natural joins  
-hasovacia funkcia h je pouzita pre rozdelenie n-tic oboch relacii do mnozin, ktore maju rovnaku  
 hodnotu hesovacej funkcie na spojovacich atributoch  
-h mapuje atributy AS na hodnoty {0,1,…n}, kde AS = spolocne atributy relacii r a s pouzite   
 prirodzenym spojenim  
-r0,r1,…,rn - oznacuju oblasti relacie r, kazda n-tica tr ∊ r je vlozena do oblasti ri kde i =h(tr[AS])  
-s0,s1,…,sn – oznacuju oblasti relacie s, kazda n-tica ts ∊ s je vlozena do oblasti si, kde i = h(ts[As])  
-n-tice relacie r z oblasti ri su porovnavane len s n-ticami relacie s z oblasti si  
-nemusia byt porovnavane s n-ticami relacie s z inych oblasti, lebo:  
 -n-tice z relacie r a s splnajuce podmienky spojenia budu mat rovnaku hodnotu pre join atributy  
 -ked je na tuto hodnotu aplikovana hasovacia funkcia h, dostaneme hodnoti i a n-tice z r  
 su ulozene v ri ako aj n-tice z s su ulozene do si

Hash-join algoritmus  
1.Rozdel relaciu s pomocou hesovacej funkcie h, jeden blok pamati je pouzity ako vyrovnavacia  
 pamat (buffer) pre kazde rozdelenie(oblast)  
2.Podobne rozdel relaciu r  
3.Pre kazde i:  
 a) Nacitaj oblast si do pamate a vytvor nad nou hash index pouzitim spojovacieho atributu,  
 tento hash index pouziva inu hashovaciu funkciu ako predoslu h  
 b) Citaj po jednej n-tice z oblasti ri, pre kazdu tr najdi odpovedajucu ts z oblasti si pouzitim  
 hash-indexu, na vystup potom daj spojenie tychto dvoch n-tic

Tranzakcia (Transaction)

Tranzakcia = postupnost operacii, ktora pristupuje a aktualizuje data

Pre zaistenie integrity dat musi databazovy system zarucovat ACID vlastnosti:  
1. Atomickoct (Atomicity): updaty ciastocne vykonanych tranzakcii by sa nemali prejavit v databazi,   
 tj. bud su prevedene vsetky operacie tranzakcie, alebo ziadna

2. Trvanlivost (Durability): akonahle bola cela tranzakcia dokoncena, updaty musia zotrvat aj po vypadku

3. Konzistencia (Consistency): vykonanie jednej (izolovanej) tranzakcie zachovava konzistenciu databazy

4. Izolacia (Isolation): pri sucasnom behu viacerych tranzakcii, musi kazda tranzakcia skryvat svoje  
 medzivysledky pred ostatnymi tranzakciami, inak by bola narusena konzistencia databazy

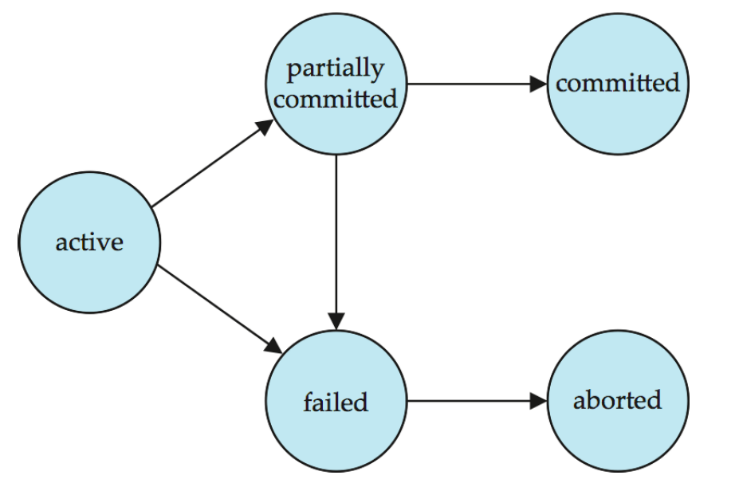
Stavy tranzakcie  
1. Aktivny (Active): pociatocny stav, tranzakcia zostava v tomto stave, pokym je vykonavana

2. Ciastocne potvrdena (Partially Committed): akonahle bola vykonana posledna operacia tranzakcie

3. Chybujuci (Failed): po zisteni, ze normalne vykonavanie tranzakcie nieje dalej mozne

4. Preruseny (Aborted): potom, ako bola tranzakcia vratena (rolled back) a databaza bola  
 vratena do stavu pred spustenim tranzakcie, dve moznosti po preruseni tranzakcie:  
 1.znovu spustit tranzakciu – iba ak nedoslo k logickej chybe  
 2.zamietnut tranzakciu

5. Potvrdeny (Committed): po uspesnom dokonceni



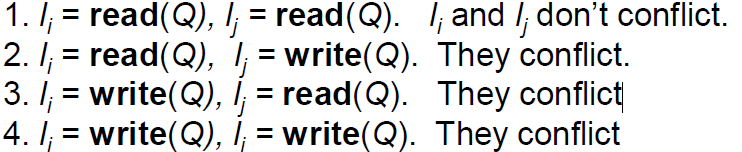
Sucasne vykonavanie tranzakcii  
-viac tranzakcii moze byt sucasne vykonavanych  
-zvysene vyuzitie procesora a disku, co vedie k vyssej tranzakcnej priepustnosti  
 -jedna tranzakcia moze vyuzivat CPU, zatialco druha moze citat alebo zapisovat na disk  
-zmenseny cas odozvy pre tranzakcie: kratke tranzakcie nemusia cakat na dlhe  
-schemata pre riadenie subeznosti (Concurrency control schemes): mechanizmy pre zaistenie  
 izolovanosti => kontrolovanie subeznych tranzakcii, aby nenarusili konzistenciu databazy

Plany (Schedules)  
-postupnost instrukcii, ktora urcuje poradie, v ktorom budu vykonavane instrukcie subeznych tranzakcii  
-plan pre mnozinu tranzakcii, musi pozostavat zo vsetkych instrukcii danych tranzakcii  
-plan musi zachovat poradie instrukcii v kazdej individualnej tranzakcii

Serializovatelnost (Serializability)  
-zakladny predpoklad: kazda tranzakcia zachovava konzistenciu databazy  
-seriove vykonanie mnoziny tranzakcii zachovava konzistenciu databazy  
-plan A je serializovatelny, ak je ekvivalentny seriovemu planu  
-rozdielne formy ekvivalencie planov vedu k nasledujucim pojmom:  
 1.Conflict serializability  
 2.View serializability

Zjednoduseny pohlad tranzakcii (Simplified view of transactions)  
-ignoruju sa operacie ine ako read a write  
-predpoklada sa, ze tranzakcie mozu vykonavat lubovolne vypocty na detach v lokalnych bufferoch,  
 medzi citanim a zapisom  
-zjednodusene plany pozostavaju len z instrukcii read a write

Konfliktne instrukcie  
-instrukcie li a lj tranzakcii Ti a Tj su vkonflikte, len ak existuje polozka Q, ku ktorej pristupuje li aj lj a aspon  
 jedna z tychto instrukcii polozku Q zapisuje



Konfliktna serializovatelnost (Conflict serializability)  
-ak plan S moze byt transformovany na plan S’ seriou prehodeni nekonfliktnych instrukcii => S a S’ su   
 konfliktne ekvivalentne (conflict equivalent)  
-plan S je konfliktne serializovatelny, ak je ekvivalentny podla konfliktu so seriovym planom

Pohladova serializovatelnost (View serializability)  
-nech S a S’ su dva plany s rovnakou mnozinou tranzakcii, S a S’ su ekvivalentne na pohlade (view   
 equivalent), ak tieto podmienky su splnene pre kazdu datovu polozku Q:  
 1. Ak v plane S tranzakcia Ti cita inicialnu hodnotu polozky Q, potom aj v plane S’ musi   
 tranzakcia Ti citat inicialnu hodnotu polozky Q  
 -inicialna hodnota = prva operacia read(Q) v kazdej tranzakcii  
 2. Ked tranzakcia Ti prevedia read(Q) v plane S a hodnota je vysledok tranzakcie Tj, potom  
 tranzakcia Ti v plane S’ musiet tiez citat hodnotu Q, ktora je vysledok tranzakcie Tj 3. Ked tranzakcia Ti v plane S prevedie posledny zapis write(Q), tranzakcia Ti v plane S’ musi  
 tiez previest finalny zapis write(Q)  
-plan S je pohladovo serializovatelny, ak je pohladovo ekvivalentny seriovemu planu  
-kazdy plan, ktory je serializovatelny podla konfliktu, je tiez pohladovo serializovatelny

Realizovatelne plany (Recoverable Schedules)  
-ak tranzakcia Tj cita datovu polozku zapisanu predoslou tranzakciou Ti, potom odovzdanie  
 operacie tranzakcie Ti je realizovane skor, ako odovzdanie operacie tranzakcie Tj

Cascading rollback  
-zlyhanie jednej tranzakcie vedie k serii tranzakcnych “vrateni” (akoby tranzakcie neboli prevedene)

Bezkaskadove plany (Cascadeless plany)  
-kaskadove vratenia nemozu nastat ak:  
 -pre kazdy par tranzakcii Ti a Tj, kde Tj cita datovu polozku zapisanu predoslou tranzakciou Ti,   
 odovzdanie operacie Ti sa uskutocni skor ako read operacia tranzakcie Tj-kazdy bezkaskadovy plan je tiez realizovatelny(recoverable)