Relacione baze podataka

Funkcionalne zavisnosti

Normalizacija

Indeksi

Upravljanje transakcijama

Oporavak

Optimizacija

Pogledi

Sigurnost

Funkcionalne zavisnosti

R je relacija, a X i Y su podskupovi atributa iz R, tada kažemo da **Y funkcionalno zavisi od X** odnosno da **X funkcionalno određuje Y**, u oznaci **X** $\rightarrow$ **Y** akko je svakoj važećoj vrednosti torke X u relaciji R pridružena tačno jedna vrednost Y u relaciji R.
To znači da ako su dve torke iz R identične na svim atributima iz X onda moraju biti identične i u svim atributima iz Y.

- Primer:

  {Indeks} $\rightarrow$ {Ime}, {Indeks} $\rightarrow$ {Prezime}, {Indeks} $\rightarrow$ {Mesto_rodjenja}
  {Id_predmeta} $\rightarrow$ {Sifra}, {Id_predmeta} $\rightarrow$ {Naziv}, {Id_predmeta} $\rightarrow$ {Bodovi}
  {Indeks, Id_predmeta, Godina_roka, Oznaka_roka} $\rightarrow$ {Ocena}

Skup atributa X relacije R je **kandidat za ključ** ako važi:

- Za svaki skup atributa Y koji ne pripadaju X važi: X $\rightarrow$ Y
- Ne postoji skup atributa Z iz X i skup atributa W koji ne pripada Z za koje važi:
  Z $\rightarrow$ W

Nadključ (superključ) relacije je skup atributa koji uključuje kao podskup bar jedan kandidat za ključ te relacije.

Napomene:

- Svaka FZ je ograničenje integriteta. Zbog toga svaki atribut relacije funkcionalno zavisi od nekog kandidata za ključ.
- FZ ne zavisi od trenutne vrednosti relacije. Zbog toga ne važi {Naziv} $\rightarrow$ {Oznaka_roka}.
- Dva skupa FZ S i T nad relacijom R su ekvivalntna ako je skup instanci relacije koji zadovoljava S jednak skupu instanci relacije koji zadovoljava T. Dva skupa FZ S i T nad relacijom R su ekvivalentna akko se svaka FZ u S izvodi iz nekih FZ u T i svaka FZ u T izvodi iz nekih FZ u S.

Trivijalna zavisnost je FZ koja je zadovoljena za bilo koji skup vrednosti relacije. FZ koja nije trivijalna je **netrivijalna zavisnost**. Ako je Y $\subseteq$ X onda je X $\rightarrow$ Y trivijalna.

- Primer:

  {Indeks} $\rightarrow$ {Indeks}, {Id_predmeta} $\rightarrow$ {Id_predmeta}
  {Godina_roka, Oznaka_roka} $\rightarrow$ {Godina_roka, Oznaka_roka}, {Godina_roka, Oznaka_roka} $\rightarrow$ {Godina_roka}

Zatvorenje skupa FZ S, u oznaci S^+, je skup svih FZ koje mogu da se izvedu iz S. Dva skupa FZ S i T su ekvivalentni ako važi $S^+ = T^+$.

Armstrongove aksiome:

- **Refleksivnost**: $B \subseteq A \Rightarrow A \rightarrow B$
- **Proširenje**: $A \rightarrow B \Rightarrow AC \rightarrow BC$
- **Tranzitivnost**: $A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C$
- **Samoodređenje**: $A \rightarrow A$
- **Dekompozicija**: $A \rightarrow BC \Rightarrow A \rightarrow B \wedge A \rightarrow C$
- **Unija**: $A \rightarrow B \wedge A \rightarrow C \Rightarrow A \rightarrow BC$
- **Kompozicija**: $A \rightarrow B \wedge C \rightarrow D \Rightarrow AC \rightarrow BD$
- **Opšta teorema unifikacije**: $A \rightarrow B \wedge C \rightarrow D \Rightarrow A \cup (C - B) \rightarrow BD$

Algoritam za određivanje zatvorenja skupa FZ F:

1. $F^+ = F$
2. while(ima promena u F^+) {
    1. za svaku FZ iz F^+ primeniti **refleksivnost** i **proširenje** i dodati dobijene FZ u F^+
    2. za svaki par FZ iz F^+ primeniti **tranzitivnost** ako je moguće i dodati dobijene FZ u F^+
    }

Neka je $A = \{A_1, A_2, \ldots, A_n\}$ skup atributa relacije. **Zatvorenje skupa atributa A**, u oznaci A^+, u odnosu na skup FZ S je skup atributa B takvih da svaka relacija koja zadovoljava sve FZ u S zadovoljava i FZ $A_1, A_2, \ldots, A_n \rightarrow B$. Svi pojedinačni atributi iz A su uvek u A^+. Skup atributa K je **nadključ** relacije akko je K^+ jednako skupu svih atributa relacije.

Algoritam za određivanje zatvorenja skupa atributa A:

1. **Dekompozicija** svih FZ tako da na desnoj strani imaju samo jedan atribut
2. $A^+ = \{A_1, A_2, \ldots, A_n\}$
3. while(ima promena u A^+) {
    1. za svaku FZ $X \rightarrow Y$, ako je $X \subset A$ onda $A^+ = A^+ \cup Y$
    }

Skup FZ S je **pokrivač** skupa FZ T ako je svaka FZ iz T uključena u S, odnosno ako je $T^+ \subset S^+$. Dva skupa FZ su ekvivalentna ako su jedan drugom pokrivač.

FZ $X \rightarrow Y$, skupa FZ S, je **levo-nereducibilna** ako iz X ne može da se ukloni nijedan atribut bez promene zatvorenja S^+. Nadključ K je **kandidat za ključ** relacije akko je njen nereducibilni nadključ.

Skup FZ je **nereducibilan** ako:

- **Singlton zavisna promenljiva**: desna strana svake FZ u S sadrži tačno jedan atribut
- **Nereducibilna determinanta**: leva strana svake FZ u S je levo-nereducibilna

- **Nema redudantnih FZ**: nijedna FZ ne može da se ukloni iz S bez promene S^+

Skup FZ koji zadovoljava prethodna pravila naziva se **kanonički** ili **minimalan**.

Algoritam za određivanje nereducibilnog skupa FZ S:

1. **Dekompozicija** svih FZ tako da na desnoj strani imaju samo jedan atribut.
2. Za svaku FZ proveriti da li se može obrisati bilo koji atribut sa leve strane bez promene S^+ i obrisati ga ako može.
3. Od preostalih FZ obrisati one čijim se brisanjem neće promeniti zatvorenje S^+.

Neka je nad relacijom R izvršena **projekcija** i neka je tada rezultujuća relacija $R_1 = \pi(R)$. Pitanje je kako odrediti koje FZ iz skupa FZ S relacije R će važiti i u R_1.

Algoritam za određivanje važećih FZ u R_1

1. Skup važećih FZ inicijalizujemo na prazan skup, $T = \emptyset$
2. Za svako $X \subseteq R_1$ odrediti zatvorenje skupa atributa X^+ u odnosu na FZ u S
3. Dodati u T sve netrivijalne FZ $X \rightarrow A$ takve da $A \subset X^+ \wedge A \subset R_1$
4. Minimalizovati skup T

Kandidate za ključ možemo odrediti tako što odredimo zatvorenje skupa svih podskupova atributa relacije. Kandidati za ključ će biti oni podskupovi atributa čije zatvorenje sadrži sve atribute relacije, a koji kao podskup ne sadrže neki od već određenih kandidata za ključ.

Još neka pravila:

- Ako se atribut ne javlja na **desnoj strani** niti jedne FZ tada on mora da bude deo ključa.
- Ako se atribut ne javlja na **levoj strani** niti jedne FZ tada on nije deo ključa.

Normalizacija

Normalizacija je proces zamene relacija skupom relacija koje su u pogodnijem obliku. Svrha normalizacije je **izbegavanje redudantnosti** i pojedinih **anomalija ažuriranja**. U procesu normalizacije se operator projekcije koristi tako da je proces reverzibilan, tj. od dobijenog izlaza se može doći do početnog ulaza.

Pretpostavimo da smo atribute *Ime* i *Prezime* premestili u relaciju *Ispit*. Jasno je da će se ime i prezime ponavljati pri svakom zabeleženom polaganju ispita i ovo nepotrebno ponavljanje podataka naziva se **redudantnost**.

Anomalije ažuriranja:

- **Anomalije unošenja**: ne možemo uneti ime i prezime ako student nikada nije polagao nijedan ispit.
- **Anomalije brisanja**: ako obrišemo polaganja ispita, obrisaćemo i informacije o studentu.
- **Anomalije izmene**: ne možemo pojedinačno menjati vrednosti imena i prezimena, već ih moramo promeniti pri svakom polaganju.

Relacija je u **prvoj normalnoj formi (1NF)** akko u svakoj važećoj vrednosti relacije svaka torka sadrži tačno jednu vrednost za svaki atribut, odgovarajućeg tipa i sve torke su jedinstvene.

Relacija je u **drugoj normalnoj formi (2NF)** akko je svaki neključni atribut nereducibilno zavisan od primarnog ključa.

Relacija je u **drugoj normalnoj formi (2NF)** akko za svaku netrivijalnu FZ $X \rightarrow Y$ važi jedan od sledećih uslova:

- X je superključ
- Y je deo ključa
- X nije pravi podskup ključa

Relacija je u **trećoj normalnoj formi (3NF)** akko je u 2NF i svaki neključni atribut je netranzitivno zavisan od primarnog ključa.

Relacija je u **trećoj normalnoj formi (3NF)** akko za svaku netrivijalnu FZ $A_1A_2...A_n \rightarrow B_1B_2...B_m$ važi jedan od sledećih uslova:

- $\{A_1A_2...A_n\}$ je superključ
- svaki atribut $B_1B_2...B_m \notin \{A_1A_2...A_n\}$ je element nekog kandidata za ključ

Ovakva definicija 3NF ne uzima u obzir slučajeve kada:

- relacija ima više od jednog kandidata za ključ
- kandidat za ključ je kompozitan
- kompozitni kandidati za ključeve se preklapaju

Relacija je u **Bojs-Kodovoj normalnoj formi (BCNF)** akko svaka netrivijalna levo-nereducibilna FZ ima kandidat za ključ kao svoju levu stranu.

Relacija je u **Bojs-Kodovoj normalnoj formi (BCNF)** akko svaka netrivijalna FZ ima superključ kao svoju levu stranu.

Poželjne osobine pri redukciji:

- eliminacija anomalija
- mogućnost rekonstrukcije anomalija
- očuvanje FZ, ako se dekompozicija vrši prema FZ tada se informacije sigurno čuvaju

Hitova teorema: Neka je A skup tributa relacije R i neka su $X \subseteq A$, $Y \subseteq A$, $Z \subseteq A$, tako da važi $X \cup Y \cup Z = A$. Neka je $XY = X \cup Y$ i $XZ = X \cup Z$. Ako u relaciji R važi FZ $X \rightarrow Y$, tada je R jednako spajanju projekcija na XY i XZ. Takođe treba da važi:

- Sve FZ polaznog skupa moraju da budu očuvane.
- Ako u novodobijenim projekcijama nastalim razbijanjem osnovne relacije postoji zajednički atribut, on mora da bude ključ u bar jednoj od novodobijenih relacije.

Algoritam za svođenje relacije R na BCNF:

1. Ako je relacija već u BCNF rešenje je R
2. Ako FZ $X \rightarrow Y$ narušava BCNF, razdvojiti relaciju R na relacije $R_1 = \{X\}^+$ i $R_2 = \{X\} - (R - \{X\}^+)$, korak zasnovan na Hitovoj teoremi
3. Odrediti skupove FZ S_1 i S_2 za relacije R_1 i R_2 koristeći algoritam za određivanje skupa FZ u projekciji relacije
4. Primeniti rekurzivno algoritam na relacije R_1 i R_2
5. Za rešenje uzeti skup svih krajnjih dekompozicija

Neka je R relacija i neka su A, B i C podskupovi atributa relacije R. Kažemo da je B **višeznačno zavisno** od A, u oznaci $A \rightarrow\rightarrow B$, akko u svakoj mogućoj važećoj vrednosti relacije R skup vrednosti B koji se uparuje sa parom (vrednost A, vrednost C) zavisi jedino od vrednosti A i nezavisan je od vrednosti C. Višeznačna zavisnost $A \rightarrow\rightarrow B$ je **trivijalna** ako je A nadskup od B ili $A \cup B$ sadrži sve atribute relacije.

Relacija je u **četvrtoj normalnoj formi (4NF)** akko je u BCNF i svaki put kada postoje podskupovi atributa A i B takvi da je zadovoljena netrivijalna višeznačna zavisnost $A \rightarrow\rightarrow B$, tada su svi atributi relacije takođe funkcionalno zavisni od A.

Neka je R relacija i neka su A, B, . . ., Z podskupovi atributa relacije R. Relacija zadovoljava **zavisnost spajanja** *{A, B, . . ., Z} akko je relacija u 4NF i svaka moguća važeća vrednost u relaciji je jednaka spajanju njenih projekcija na A, B, . . ., Z. Zavisnost spajanja *{A, B, . . ., Z} je **trivijalna** ako je najmanje jedan od A, B, . . ., Z skup svih atributa relacije. Zavisnost spajanja *{A, B, . . ., Z} je **posledica kandidata za ključ** relacije R akko je svaki od A, B, . . ., Z nadključ za relaciju R.

Relacija je u **petoj normalnoj formi (5NF)** akko je u 4NF i svaka netrivijalna zavisnost spajanja koja važi u relaciji R je posledica kandidata za ključ u relaciji R.

Ako je relacija u BNCF i nema kompozitne ključeve, tada je ona i u **5NF**.

Relaciona promenljiva R je:

1. **normalizovana** akko je u **1NF** (sve relacije u relacionom modelu su u 1NF)
2. u **2NF** akko za svaki ključ K i svaki neključni atribut A, FZ K $\rightarrow$ {Y} koja važi u R je nereducibilna, tj. K je minimalni ključ
3. u **3NF** akko u svakoj netrivijalnoj FZ X $\rightarrow$ Y koja važi u R je X superključ ili je podskup ključa
4. u **BCNF** akko u svakoj netrivijalnoj FZ X $\rightarrow$ Y koja važi u R je X superključ

Proces normalizacije:

1. Uzeti projekcije originalne relacije u 1NF radi eleminisanja FZ koje nisu nereducibilne. Dobijeni skup relacija je u 2NF.
2. Uzeti projekcije relacija u 2NF radi eliminisanja tranzitivnih zavisnosti. Dobijeni skup relacija je u 3NF.
3. Uzeti projekcije relacija u 3NF radi eliminisanja preostalih FZ u kojima na levoj strani nije kandidat za ključ. Dobijeni skup relacija je u BCNF.
4. Uzeti projekcije relacije u BCNF radi eleminisanja VZ (višeznačnih zavisnosti) koje nisu FZ. Dobijeni skup relacija je u 4NF.
5. Uzeti projekcije relacija koje su u 4NF i eliminisati ZS (zavisnosti spajanja) koje ne slede iz kandidata za ključeve. Dobijeni skup relacija je u 5NF.

U praksi se normalizacija najčešće sprovodi do BCNF, jer puna normalizacija narušava performanse:

- dovodi do velikog broja logički razdvojenih relacija
- to povlači veliki broj razdvojenih datoteka u kojima se te relacije čuvaju
- to dalje povlači veliki broj U/I operacija

Denormalizacija je proces suprotan normalizaciji, gde se u već normalizovanu relaciju dodaju redudantni podaci, sa ciljem da se povećaju performanse upita. Često se primenjuje kada su performanse bitnije od normalizacije podataka, na primer kada imamo mnogo više čitanja podataka nego pisanja (sistem za analizu podataka).

Indeksi

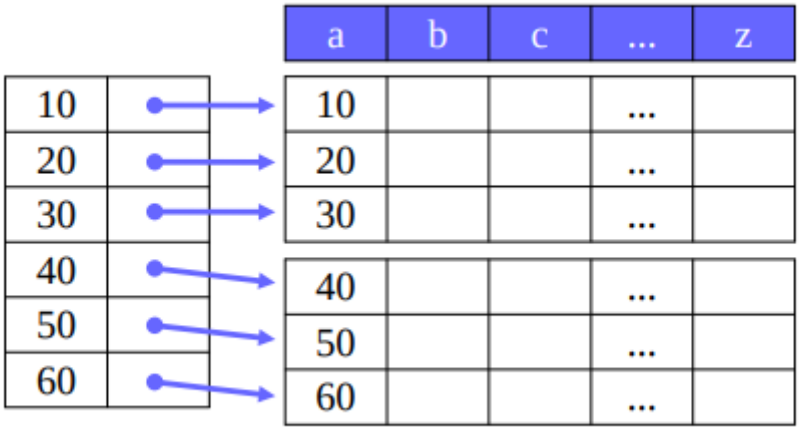
Indeks nad atributom A je struktura podataka koja omogućuje jednostavno pronalaženje svih elemenata koji imaju fiksiranu vrednost atributa A. Svaki indeks se formira nad nekim atributima. Indeks čuva uređenu vrednost ključa pretrage zajedno sa listom pokazivača na odgovarajući slog. Posebno su korisni nad

atributima koji se koriste u where klauzuli ili spajanju. Prilikom promena sadržaja tabele potrebno je promeniti i indekse. Za nalaženje potrebnog ključa čita se više polja u indeksu ali je efikasnost povećana jer:

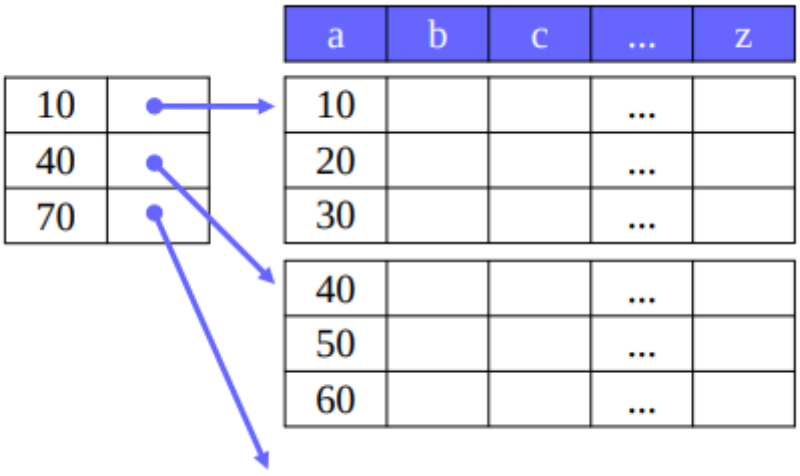
- veličina polja u indeksu je obično znatno manja od veličine sloga
- ključevi su sortirani što omogućuje binarno pretraživanje

Prema upotrebi pokazivača indeksi se dele na:

- **guste** - po jedan upis za svaku vrednost ključa.



- **retke** - po jedan upis za svaki blok podataka. Prilikom pretrage ključa K prvo se nalazi najveći ključ manji ili jednak K. Čita se blok na koji pokazuje nađeno polje, a zatim se taj blok pretražuje i nalazi ključ K.



Prema skupu atributa nad kojima se formiraju indeksi se dele na:

- **primarne** - nad atributima koji su primarni ključ, forsiraju jedinstvenost
- **sekundarne** - mogu ali ne moraju da forsiraju jedinstvenost

- **klasterovane** - tabela se organizuje u redosledu koji odgovara vrednostima atributa indeksa

Indeks može da se proširi na više memorijskih blokova u slučaju velikih podataka. Upotreba **indeksa na više nivoa** može da poveća efikasnost. Može se formirati gust indeks na drugom nivou, ali to nema smisla. Takođe, nema smisla koristiti redak indeks na nesortiranim podacima. Može se korisiti redak indeks nad gustim indeksom nad nesortiranim podacima.

Indeksna datoteka je sekvencijalna datoteka koja mora da se tretira na sličan način kao i datoteka sa sortiranim slogovima:

- korišćenje pomoćnog bloka kod prepunjenosti tekućeg bloka, tj. korišćenje **bloka prekoračenja**
- unos novih blokova
- prebacivanje elemenata u odgovarajući blok

Gust indeks pokazuje na pojedinačni slog pa se:

- modifikuje ako je slog unet, izbrisan ili pomeren
- ne vrši nikakva akcija u slučaju operacija nad blokovima

Redak indeks pokazuje na blok pa:

- može da se modifikuje ako je slog unet, izbrisan ili pomeren
- nema akcije u slučaju uključivanja bloka prekoračenja
- pokazivač mora da se unese/izbriše ako se unosi/briše sekvencijalni blok

Karakteristike indeksa:

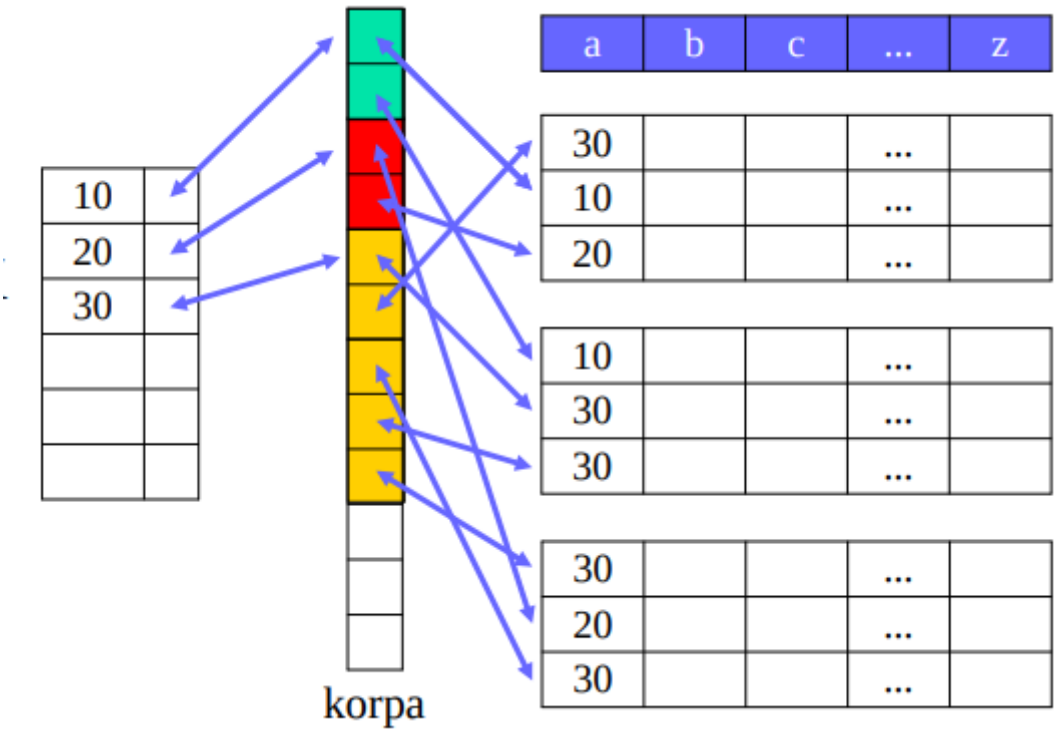
	Gusti indeks	**Redak indeks**
Prostor	jedno polje po slogu	jedno polje po bloku
Pristupi bloku	veliki broj	mali broj
Pristupi slogu	direktan pristup	pretraživanje unutar bloka
Upiti sa exists	koristi samo indeks	uvek mora da pristupi bloku
Upotreba	bilo gde	sortirani elementi
Modifikacija	uvek se ažuriraju ako se menja redosled slogova	ažuriraju se jedino kada se menja prvi slog u bloku

Indeksi nad neključnim atributima mogu da sadrže duplirane vrednosti. Ako su slogovi sortirani po tim vrednostima u pitanju je klasterovani indeks. Problem duplikata se može rešiti na više načina:

1. gusti indeks nad jednim indeksni poljem gde bi postojao jedan pokazivač na sve duplikate, pokazivao bi samo na prvi slog i svi duplikati bi se pronalazili jedinstvenim ključem za pretragu. Prednost je što se lako pronalazi slog i broj duplikata, a mana to što postoji više polja nego što je potrebno pa indeks može da se proširi na više blokova.
2. redak indeks. Prednost je što je indeks manji i pretraživanje je brže, a mana složenije pronalaženje uzastopnih slogova.

Primarni i **klasterovani indeksi** rade sa podacima kod kojih su ključevi sortirani. **Sekundarni indeksi** rade sa nesortiranim vrednostima ključeva, ali se kao i ostale vrste indeksa koriste za brzo nalaženje slogova. Kod njih su dozvoljeni duplikati vrednosti ključeva. Prvi nivo im je uvek gust, dok su svi ostali viši nivoi retki. Sam indeks je sortiran po vrednostima ključa što pojednostavljuje pretraživanje.

Korpe su uobičajen način za izbegavanje dupliranja vrednosti. Prvo se formira korpa datoteka. Vrednost indeksa za K pokazuje na prvi element u korpi za K. Korpa datoteka se obrađuje kao i ostale sortirane datoteke.

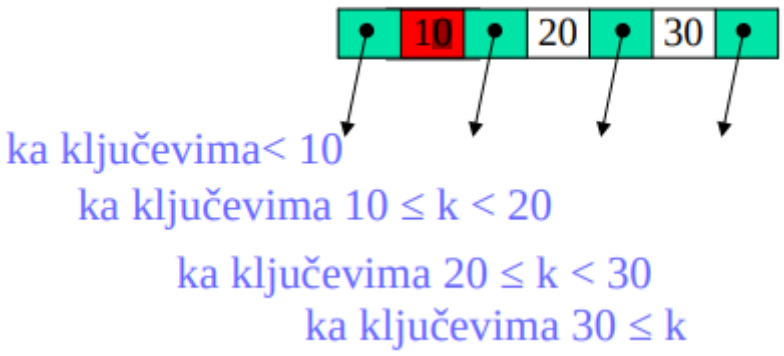


Umesto indeksa često se koristi opšta struktura **B-drveta**:

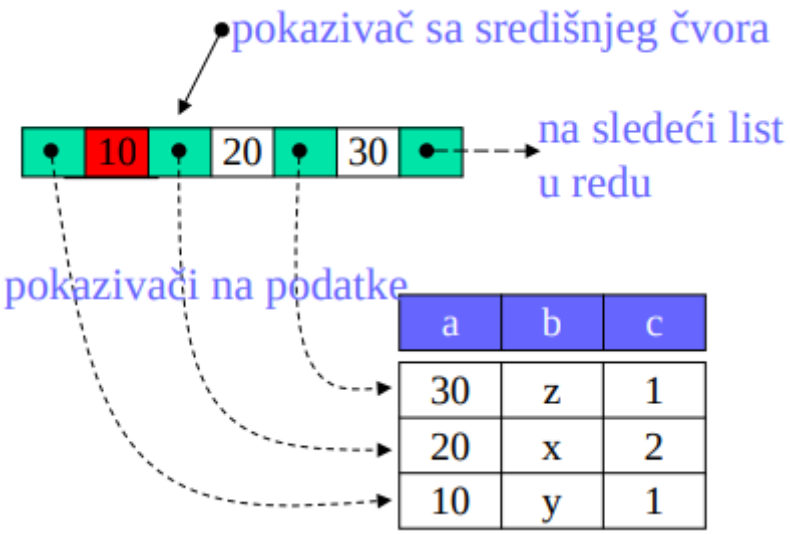
- automatski se održava odgovarajući broj nivoa
- održava se prostor u blokovima
- drvo je balansirano - svi listovi su na istom nivou

Pored B-drveta, postoji i **B$^+$-drvo**. U B-drvetu svi ključevi po kojima se pretražuje i odgovarajući pokazivači na podatke su predstavljeni negde u drvetu, dok su u B$^+$-drvetu svi pokazivači na podatke u listovima sortirani po vrednostima sa leva na desno. Čvor ima n vrednosti (ključeva pretrage) i n+1 pokazivač. Kod čvorova u sredini svi pokazivači pokazuju na čvorove na nižem nivou. Listovi sadrže n pokazivača na podatke i 1 pokazivač na naredni čvor (ponekad i jedan dodatni pokazivač na prethodni čvor).
Čvorovi ne mogu da budu prazni. Čvorovi u sredini koriste najmanje $\lceil \frac{n+1}{2} \rceil$ pokazivača na čvorove na narednom nivou, a listovi koriste najmanje $\lceil \frac{n+1}{2} \rceil$ pokazivača na podatke. Pošto su svi listovi na istom nivou i povezani, jednostavno je sekvencijalno čitanje podataka. Pomoću B$^+$-drveta lako se proverava da li slog sa traženom vrednošću ključa postoji.

- Čvorovi u sredini:



- Listovi:



Algoritam za unošenje podataka u B^+-drvo

1. Odrediti korektan list L.
2. Upisati novu vrednost u list L.
3. Ako postoji prostor unos je završen.
4. Ako ne postoji prostor:
    1. Dodaje se novi list L1 i podaci iz lista L se ravnomerno raspoređuju na L i L1 tako da je svaki od njih pun bar do polovine.
    2. Unosi se bočni pokazivač sa L na L1 i sa L1 na prethodni naredni od L.
    3. Srednji ključ se prepisuje u čvor na nivou iznad.
    4. Unosi se pokazivač na list L1 sa roditelj čvora lista L.
5. Podela čvora na jednom nivou utiče na čvor na višem nivou ako u njega treba da se unese pokazivač na ključ. Po potrebi, rekurzivno ponoviti postupak - ako postoji prostor uneti vrednost/pokazivač, a ako ne postoji podeliti čvor.
6. Ako se deli indeksni čvor, podaci se ravnomerno raspoređuju, ali se srednji ključ pomera naviše.
7. Izuzetak predstavlja koreni čvor - ako treba da se u njega unese podatak, a ne postoji slobodan prostor tada se koreni čvor deli na dva i formira novi koreni čvor na višem nivou sa dva dete čvora. Bez obzira na broj n korenom čvoru je dozvoljeno da ima samo jedan ključ i dva dete čvora.

Algoritam za brisanje podataka iz B^+-drveta

1. Odrediti korektan list L u kome se nalazi ključ koga treba obrisati.
2. Ako posle uklanjanja ključa list ima bar polovinu elemenata, brisanje je završeno.
3. U suprotnom, ako neki od susednih listova ima više od minimalnog broja ključeva i pokazivaca, tada se od njega pozamljuje par ključ/pokazivač, tako da se redosled ključeva ne menja. Pri tome se vodi računa da roditelj ključevi treba da pokazuju na novo stanje.
4. Ako oba susedna lista imaju minimum parova, tada se spajaju dva susedna lista i briše jedan od njih. Broj ključeva/pokazivača u novom listu je manji od maksimuma jer su spojeni list sa minimumom i list sa manje od minimuma broja parova.
5. Ažuriraju se ključevi roditelj čvora spojenih listova i izbriše par ključ/pokazivač u roditelj čvoru koji je pokazivao na izbrisani list.
6. Ako roditelj čvor ima manje od minimalnog broja parova, postupak se rekurzivno ponavlja, u suprotnom brisanje je završeno.

B^+-drveta mogu da se koriste za implementaciju indeksa nad podacima iz baze. Unutrašnji čvorovi služe za ubrzavanje pretraživanja. Listovi mogu da se ponašaju kao:

- Gusti ili retki primarni indeksi
- Gusti ili retki klasterovani indeksi koji dopuštaju duplikate
- Gusti sekundarni indeksi nad nesortiranim slogovima

Mana B^+-drveta je to što pretraživanje uvek mora da počne od korena. Broj blokova koji se onda čitaju je jednak visini drveta plus broj pristupa slogovima.

Prednosti B^+-drveta:

- Broj nivoa je obično realtivno mali, najčešće 3
- Upiti sa intervalima se veoma brzo izvršavaju, pronađe se leva granica i onda se vrši sekvencijalno čitanje
- Ako je n dovoljno veliko, podela i spajanje će biti relativno retki
- U/I operacije sa diska mogu biti smanjene ako se koreni blok stalno drži u memoriji tako da je samo mali broj U/I operacija potreban za pristup slogu

Upravljanje transakcijama

Transakcija je logička jedinica posla pri radu sa podacima. Po izvršenju kompletne transakcije stanje baze treba da je **konzistentno**, tj. da su ispunjeni uslovi integriteta. Konzistentnost može biti narušena u međukoracima transakcije.

ACID svojstva:

- **atomičnost** - transakcija se izvrši u celosti ili se uopšte ne izvrši
- **konzistentnost** - transakcija prevodi jedno konzistentno stanje baze u drugo konzistentno stanje baze
- **izolacija** - efekti izvršenja transaksije su nepoznati drugim transakcijama sve dok se ona uspešno ne završi
- **trajnost** - svi efekti uspešno kompletirane transakcije su trajni, mogu se poništiti samo drugom transakcijom

---

U radu sa transakcijama SUBP koristi usluge upravljača transakcijama. Sistem za upravljanje transakcijama obezbeđuje upis efekata izvršenja svih radnji od kojih se transakcija sastoji ili nijedne. RSUBP DB2 koristi dve radnje upravljača transakcijama:

- **COMMIT** - označava uspešan kraj transakcije i trajan upis efekata svih radnji transakcije u bazu
- **ROLLBACK** - označava neuspešan kraj transakcije i poništenje svih efekata koje su proizvele radnje te transakcije

U sistemu DB2 transakcija je:

- deo aplikacije od početka programa do prve radnje COMMIT ili ROLLBACK
- deo aplikacije između dve susedne takve radnje u programu
- deo aplikacije od poslednje takve radnje do kraja programa

- ako se nijedna od radnji COMMIT ili ROLLBACK ne uključi u program, čitav program predstavlja jednu transakciju i na kraju programa se sistemski generiše jedna COMMIT ili ROLLBACK komanda

---

Konkurentnost predstavlja istovremeno izvršavanje više transakcija. Istovremeno izvršenje može da naruši uslove integriteta, dok **serijsko izvršenje** ispunjava uslove integriteta. Prilikom izvršenja skupa transakcija redosled radnji iz svake transakcije je nepromenjen, ali dve uzastopne radnje ne moraju pripadati istoj transakciji. Pojedinačna radnja izvršava se u jednom koraku.

Osnovne komponente transakcije:

- **objekti** - na primer slogovi
- **čitanje** i **upis**, tj. **ažuriranje**

Dve radnje su **konfliktne** ako se obavljaju nad istim objektom a jedna od njih je radnja upisa, odnosno konfliktne radnje su (čitanje, upis) i (upis, upis). Dva izvršenja datog skupa transakcija su ekvivalentna ako imaju isti redosled izvršavanja svakog para konfliktnih radnji. Za skup transakcija T, skup objekata O nad kojim transakcije obavljaju radnje i neko izvršenje I, redosled izvršavanja konfliktnih radnji opisuje se **relacijom zavisnosti**. Relacija zavisnosti izvršenja I skupa transakcija T = {T1, T2}, u oznaci Z(I), je podskup Dekartovog proizvoda $T \times O \times T$ koji sadrži trojke (T', o, T'') sa svojstvima:

- u i-tom koraku transakcija T' obavlja čitanje ili upis nad objektom o, a u sledećem (j-tom) koraku transakcija T'' obavlja čitanje ili upis nad tim istim objektom
- jedna od dve uočene radnje je upis
- između i-tog i j-tog koraka izvršenja I nema radnje upisa nad objektom o

Analogno se može definisati relacija zavisnosti za n transakcija. Dva **izvršenja skupa transakcija** su ekvivalentna ako su im jednake relacije zavisnosti. Izvršenje skupa transakcija je **linearizovano** ako je ekvivalentno nekom serijskom izvršenju. Za serijsko, tj. linearizovano, izvršenje I skupa transakcija T važi da je relacija zavisnosti Z(I) **antisimetrična**, tj. za proizvoljna dva objekta važi: ako je (T1, o1, T2) iz Z(I), onda (T2, o2, T1) nije iz Z(I). Relacija zavisnosti takvog izvršenja definiše **linearno uređenje <**:

- T1 < T2 akko (T1, o, T2) pripada Z(I) za neki objekat o

Ako jedna transakcija počne po završetku druge (npr. A1, B1, A2, B2) ili ako im se radnje izvršavaju naizmenično (npr. A1, A2, B1, B2), izvršavanje je ili serijsko ili ekvivalentno serijskom i baza će nakon izvršenja biti u konzistentnom stanju. Ako se jedna transakcija kompletno izvrši između prve i druge radnje druge transakcije (npr. A1, A2, B2, B1) baza će se naći u nekonzistentnom stanju. Ovaj problem konkurentnosti se naziva **nekonzistentna analiza**.

Problem **izgubljenog ažuriranja** nastaje kada transakcije vrše čitanje i upis nad istim slogom. Na primer, T1 čita slog, T2 čita slog, T1 menja slog i zatvara datoteku, T2 menja slog i zatvara datoteku. Transakciji T1 se bez obaveštenja poništava efekat upisa i ostaju zapamćene samo promene transakcije T2.

Problem **zavisnosti od poništenog ažuriranja** nastaje kada transakcije istovremeno ažuriraju slogove, a onda neka od njih poništi akcije. Na primer, T2 ažurira slog, T1 čita slog, T2 poništi efekte ažuriranja ili T2 ažurira slog, T1 ažurira slog, T2 poništi efekte ažuriranja.

Prva dva problema konkurentnosti vezana su za redosled konkurentnog izvršavanja dve ili više transakcija. Treći problem je osim za konkurentnost vezan i za oporavak baze pri poništavanju transakcije.

Zaključavanje objekta je postupak koji obezbeđuje transakciji pristup objektu i istovremeno sprečava druge transakcije da mu pristupe. Svaka transakcija na kraju svog izvršavanja otključava sve objekte koje je zaključala. **Granularnost** predstavlja veličinu objekta zaključavanja - cela relacija, pojedinačne torke, delovi torke, grupe atributa itd. Kod jednostavnih modela postoji samo jedna vrsta katanca i pri izvršenju skupa transakcija nijedna transakcija ne može zaključati već zaključani objekat. U realnim sistemima postoje **deljivi** i **ekskluzivni katanci**. Više transakcija može da postavi deljivi katanac na jedan objekat, ali najviše jedna može da postavi ekskluzivni.

Transakcija je **dvofazna** ako se sastoji od dve serijske faze - **faze zaključavanja objekata** i **faze otključavanja objekata**. Faza otključavanja objekata počinje prvom radnjom otključavanja i u njoj nema više radnji zaključavanja. U fazi zaključavanja nema nijedne radnje otključavanja. Dovoljan uslov za linearizovanost izvršenja I skupa transakcija {T1, T2} je da su one dvofazne. Dvofaznost skupa transakcija nije potreban uslov, a može proizvesti i neželjeni efekat poznat kao **uzajamno blokiranje transakcija**.

Najviši **nivo izolovanosti** aplikacije i njenih transakcija u sistemu DB2 je **nivo ponovljivog čitanja (RR)**. Na ovom nivou se zaključavaju sve vrste kojima se transakcija obraća, a ne samo vrste koje zadovoljavaju neki uslov. Na taj način se rezultat izvršenja jednog upita transakcije ne može promeniti od strane druge transakcije pre nego što se ta transakcija završi. Ovaj nivo obezbeđuje i da transakcije nemaju uvid u nepotvrđene promene drugih transakcija.

Sledeći nivo izolovanosti je **nivo stabilnosti čitanja (RS)**. On obezbeđuje zaključavanje samo onih vrsta koje ispunjavaju uslov upita. Te vrste ne mogu biti promenjene od strane drugih transakcija dok se trenutna ne završi. Rezultat ponovljenih izvršenja istog upita od strane transakcije može da se razlikuje jer može doći do pojave **fantomskih redova** (npr. druga transakcija može izmeniti vrste tako da nove vrste zadovoljavaju uslov upita).

Nivo stabilnosti kursora (CS) obezbeđuje zaključavanje samo one vrste koju transakcija trenutno čita, dok su sve koje je prethodno pročitala otključane. Transakcije i dalje nemaju uvid u promene drugih transakcija pre njihovog okončanja.

Najslabiji nivo izolovanosti transakcije je **nivo nepotrvrđenog čitanja (UR)**. On obezbeđuje transakciji da vidi nepotvrđene promene drugih transakcija, kao i da druge transakcije pristupe podacima koje trenutna transakcija čita.

Objekat koji transakcija sistemski zaključava zavisi od nivoa izolovanosti transakcije i operacije i može biti različite granularnosti. Jedini objekat koji korisnik može eksplicitno da zaključa **deljivim (S)** ili **ekskluzivnim**

(X) katancem jeste tabela, pomoću **lock table iskaz**. Održavanje jednog ključa nad celom tabelom je efikasnije i adekvatno je ako transakcija obrađuje veliki broj slogova, ali smanjuje konkurentnost obrade u odnosu na zaključavanje pojedinačnih vrsta.

Deljivi katanac, tj. S-katanac, se postavlja pri čitanju objekta, a ekskluzivni katanac, tj. X-katanac, se postavlja pri upisu u objekat. Postoji i **katanac ažuriranja**, tj. U-katanac, koji omogućava menjanje objekta uz dopuštenje drugim transakcijama da čitaju podatke. Postoje i druge vrste katanaca u DB2 (**IS - intent share**, **IX - intent exclusive**, **SIX - shared with intent exclusive**). Katanci se drže do kraja transakcije, osim u slučaju CS i UR nivoa. Pojedinačne vrste mogu biti zaključane samo S, U ili X katancima, dok se katanci namere (intent) mogu postaviti samo na tabele.

Transakcija dobija katance automatski od sistema:

- kad otvori kursor za čitanje, dobija IS-katanac nad tabelom
- kad uspešno obavi čitanje vrste, dobija S-katanac na toj vrsti
- kad otvori kursor za ažuriranje, dobija IX-katanac ili U-katanac nad tabelom
- kad uspešno obavi ažuriranje vrste, dobija X-katanac na vrsti

Oporavak

Oporavak se odnosi na restaurisanje podataka baze u slučaju da dođe do greške u transakciji, sistemu ili disku (mediju). Razlog za neuspešno obavljanje transakcija može biti:

- **pad transakcije**, npr. prekoračenje neke od dozvoljenih vrednosti
- **pad sistema**, npr. prestanak električnog napajanja
- **pad medija**, npr. oštećenje glava diska na kom se nalazi baza podataka

Log datoteka (sistemski log) predstavlja duplirane podatke i informacije koji se čuvaju na različitim mestima ili medijima. Na osnovu nje je moguća rekonstrukcija baze podataka. U njoj su pokazivačima povezani slogovi koji se odnose na jednu transakciju. Ona veoma retko ili nikada ne pada i uvek treba da bude dostupna. Moguće je i dupliranje, tripliranje, itd... podataka na različitom broju medija. Komponenta SUBP - **upravljač oporavkom** preduzima aktivnosti da bi oporavljeni sadržaj baze zadovoljio uslove integriteta baze koji su prethodni bili narušeni. On periodično prepisuje celu bazu na medij za arhiviranje. Takođe, pri svakoj promeni baze upisuje slog promene u log datoteku. Upisuje tip promene, novu (pri unošenju) i staru (pri brisanju) vrednost pri ažuriranju.

U slučaju pada transakcije ili sistema, upravljač oporavka koristi informacije iz log datoteke, poništava dejstva parcijalno izvršenih transakcija i ponovo izvrši neke kompletirane transakcije. U slučaju pada medija, upravljač oporavka zadnju arhiviranu kopiju baze prepisuje na ispravni medij i koristi informacije iz log datoteke da bi ponovo izvršio transakcije koje su kompletirane posle tog zadnjeg arhiviranja a pre pada medija.

Baza podataka se menja operacijama ažuriranja, unošenja ili brisanja podataka, što predstavlja **DO-logiku**. Oporavak se bazira na **UNDO-logici** i **REDO-logici**. Pad sistema ili medija može da se dogodi i u fazi

oporavka od prethodnog pada, pa može doći do ponovnog poništavanja već poništenih radnji, ili ponovnog izvršavanja već izvršenih radnji. Zbog toga UNDO i REDO logika imaju **svojstvo idempotentnosti**, tj. undo(undo(x)) = undo(x) i redo(redo(x)) = redo(x).

Izvršenje transakcije može da se završi **planirano** ili **neplanirano**. Do planiranog završetka dolazi izvršenjem COMMIT operacije ili eksplicitne ROLLBACK operacije (greška za koju postoji programska provera, program nastavlja sa sledećom transakcijom). Do neplaniranog završetka transakcije dolazi pri grešci za koju ne postoji sistemska provera, izvršenjem implicitne (sistemske) ROLLBACK operacije (radnje transakcije se poništavaju i program prekida sa radom).

COMMIT operacijom efekti transakcije postaju trajni i ne mogu se poništiti opravkom. U log datoteku se upisuje COMMIT slog, a svi katanci koje je transakcija držala se oslobađaju. Ova operacija ne podrazumeva uvek završetak upisa svih podataka u bazu, ali se garantuje da će se u nekom trenutku upisati (kada se napuni bafer onda se upisuju podaci). U slučaju pada sistema posle COMMIT operacije upis se garantuje REDO-logikom.

Prilikom pada transakcije, tj. implicitne ROLLBACK operacije, sprovodi se aktivnost oporavka od pada transakcije. Oporavak je obezbeđen upisom svih potrebnih informacija u log datoteku kada se izvrši BEGIN TRANSACTION operacija. Operacija ROLLBACK podrazumeva poništavanje svih promena nad bazom. To se izvršava čitanjem unazad svih logova koji pripadaju toj transakciji (do BEGIN TRANSACTION loga) i za svaki slog promena se poništava UNDO-logikom. Opravak od pada transakcije ne uključuje REDO-logiku.

U slučaju pada sistema, sadržaj unutrašnje memorije je izgubljen. Pomoću log datoteke poništavaju se efekti transakcija koje su bile aktivne u trenutku pada sistema. Te transakcije mogu da se pronađu kao transakcije za koje postoji BEGIN TRANSACTION slog ali ne postoji COMMIT slog.

Da bi se smanjila količina posla vezana za oporavak od pada sistema, **tačke pamćenja** trebaju biti u pravilnim intervalima. Slog tačke pamćenja se upisuje u log datoteku. On sadrži informacije o svim aktivnim transakcijama u tom trenutku, adrese poslednjih slogova tih transakcija u log datoteci i druge informacije. Adresa sloga tačke pamćenja upisuje se u **datoteku ponovnog startovanja (restart file)**. **Protokol unaprednog upisivanja u log** obezbeđuje da se u toku tačke pamćenja odgovarajući slog prvo upisuje u log datoteku, pa se zatim podaci upisuju iz bafera podataka u bazu, bez obzira da li su baferi puni. Ako do pada sistema dođe posle upisa u log datoteku, a pre nego što sadržaj bafera bude upisan u bazu restaurisanje se vrši REDO-logikom. Upravljač oporavka podrazumeva da transakcija oslobađa sve X-katance i S-katance pri izvršenju COMMIT operacije.

Pseudokod oporavka od pada sistema:

1. Naći slog poslednje tačke pamćenja u log datoteci, iz datoteke ponovnog startovanja
2. Ako u poslednjoj tački pamćenja nema aktivnih transakcija i slog tačke pamćenja je poslednji slog u log datoteci oporavak je završen
3. Inače:
    1. Formiramo dve prazne liste - listu uspelih i listu neuspelih transakcija
    2. Sve transakcije aktivne u poslednjoj tački pamćenja se stavljaju u listu neuspelih
    3. Čitamo redom log datoteku od tačke pamćenja do kraja
        1. Kada naiđemo na slog "početak transakcije" dodajemo je u listu neuspelih
        2. Kada naiđemo na slog "kompletirana transakcija" premestimo je u listu uspelih
    4. Čitamo unazad log datoteku i poništavamo akcije neuspelih transakcija
    5. Čitamo log datoteku od poslednje tačke pamćenja i ponovo izvršavamo uspele transakcije

Postoji niz poboljšanja postupka oporavka od pada sistema:

- Moguće je sva upisivanja jedne transakcije u bazu ostaviti za trenutak izvršenja COMMIT operacije te transakcije. Time se eliminiše potreba za UNDO logikom.
- Moguće je oslabiti zahtev za izolacijom transakcije, tj. zahtev za dvofaznošću. To nosi opasnost nelinerarizovanog izvršenja.

Moguće je poboljšati i aktivnosti vezane za tačke pamćenja. Moguće je eliminisati fizičko upisivanje bafera podataka u bazu podataka u tački pamćenja. Prilikom oporavka će onda biti poništene one radnje neuspelih transakcija čiji su efekti upisani u bazu, a ponovo se izvršavaju samo one radnje uspelih transakcija čiji efekti nisu upisani u bazu. U vreme upisa sloga u log datoteku, uvodi se jedinstvena oznaka sloga - **serijski broj**. Serijski brojevi su u rastućem poretku. Kadgod se ažurirana stranica upisuje u bazu podataka, u stranicu se upisuje i serijski broj sloga sistemskog loga. Ako je serijski broj upisan u stranicu veći ili jednak serijskom broju sloga loga, efekat ažuriranja je upisan u bazu, a ako je manji onda efekat nije upisan. Pri registovanju tačke pamćenja eliminiše se fizički upis bafera podataka u bazu. Takođe, dodaje se upis, u slog tačke pamćenja, vrednosti serijskog broja m najstarije stranice bafera podataka. Slog sistemskog loga sa serijskim brojem m onda odgovara tački u sistemskom logu od koje treba ponovo izvršavati uspele transakcije pri oporavku. Tačka m prethodi tački pamćenja. Može se desiti da neka transakcija koja je uspešno kompletirana pre tačke pamćenja upadne u skup aktivnih transakcija, pa se i na nju primenjuje procedura oporavka. Pseudokod poboljšanog oporavka od pada sistema:

1. Formiramo dve prazne liste - listu uspelih i listu neuspelih transakcija
2. Sve aktivne transakcije u tački m stavljamo u listu neuspelih
3. Čitamo redom log datoteku od tačke m do kraja:
    1. Kada naiđemo na slog "početak transakcije" dodajemo je u listu neuspelih
    2. Kada naiđemo na slog "kompletirana transakcija" premestimo je u listu uspelih
4. Čitamo unazad log datoteku od kraja i poništavamo efekte radnji neuspelih transakcija, za koje važi da je serijski broj tekućeg sluga veći ili jednak serijskog broju odgovarajuće stranice baze podataka
5. Čitamo redom log datoteku od tačke m do kraja i ponovo izvršavamo radnje uspelih transakcija, za koje važi da je serijski broj tekućeg sluga manji serijskog broju odgovarajuće stranice baze podataka

Optimizacija

Optimizacija se vrši automatski. Optimizator će bolje uraditi optimizaciju od korisnika relacionog sistema jer:

- ima na raspolaganju statističke informacije iz sistemskog kataloga
- ako se statistika promeni moguće je izvršiti reoptimizaciju
- on je program i strpljiviji je i sposobniji u odnosu na čoveka
- u njega su uključene veštine i znanje najboljih programera, koje su kroz program svima na raspolaganju

---

Osnovne **faze obrade upita**:

- Parsiranje upita i provera semantike
- Konverzija upita u kanonički oblik
- Analiza i izbor kandidata za procedure niskog nivoa
- Formiranje planova upita i izbor najjeftinijeg

U prvoj fazi vrši se provera sintaksne ispravnosti upita i prevođenje upita na interni zapis koji je pogodniji za obradu u računari. Obično se koristi drvo upita ili drvo apstraktne sintakse. Proverava se i korektnost tipova argumenata, funkcija, korelacija, podupita...

U drugoj fazi se upit konvertuje u kanonički oblik koji je efikasniji, jer je jedan isti upit moguće zapisati na više načina. Za konverziju optimizator koristi različita pravila za transformaciju. Dva upita su ekvivalentna akko se pri njihovom izvršavanju u svim slučajevima dobija isti rezultat. Za podskup C datog skupa upita Q kažemo da je u kanoničkoj formi za Q akko je svaki upit q iz Q ekvivalentan nekom upitu c iz C. Upit c predstavlja **kanonički oblik** upita q.

U sledećoj fazi optimizator odlučuje na koji način će izvršavati transformisani upit. Osnovna strategija je posmatranje upita kao niza operacija niskog nivoa (spajanje, restrikcija, ...) između kojih postoje određene zavisnosti. Optimizator razmatra i postojanje indeksa, pristupnih puteva, fizičku distribuciju podataka itd. Za svaku od operacija niskog nivoa optimizator ima na raspolaganju skup predefinisanih procedura za njihovu implementaciju. Svaka procedura ima pridruženu formulu za određivanje cene implementacija, obično u zavisnosti od broja U/I operacija, CPU vremena, veličine međurezultata i slično.

U poslednjoj fazi se formiraju planovi upita i bira se onaj najbolji. Svaki plan je kombinacija kandidata za implementaciju procedura za operacije niskog nivoa. Ukupna cena upita je jednaka zbiru cena pojedinačnih procedura. Često upiti uključuju formiranje međurezultata pri izvršavanju. Optimizator na osnovu svega toga pravi procene cene koštanja upita.

Faze procesa optimizacije koriste statistiku baze podataka koja se čuva u **katalogu**. Postoji statistika o osnovnim tabelama, o kolonama u osnovnim tabelama, o indeksima itd. Statistika se ne sakuplja automatski već na zahtev korisnika. U DB2 se koristi naredba runstats.

Neke transformacije izraza:

- (a where restrikcija1) where restrikcija 2 $\iff$ a where restrikcija1 and restrikcija2
- (a [atributi1]) [atributi2] $\iff$ a[atributi2]
- (a [atributi]) where restrikcija $\iff$ (a where restrikcija) [atributi] - u opštem slučaju je bolje primeniti restrikciju pre projekcije jer se time smanjuje veličina ulaza za projekciju

Za unarni operator f se kaže da je **distributivan** preko binarnog operator O akko važi f(A O B) $\equiv$ f(A) O f(B). Restrikcija je distributivna preko unije, preseka i razlike. Restrikcija je distributivna preko spajanja akko se uslov restrikcije sastoji od najviše dva odvojena uslova koja su spojena konjunkcijom. Projekcija nije distributivna preko razlike, ali jeste preko unije i preseka:

- (a union b) [c] $\equiv$ a[c] union b[c]
- (a intersect b) [c] $\equiv$ a[c] intersect b[c]

Projekcija je distributivna preko spajanja akko važi da je AC unija atributa koji su zajednički za A i B i onih atributa C koji se pojavljuju samo u A i da je BC unija atributa koji su zajednički za A i B i onih atributa C koji se pojavljuju samo u B:

- (a join b) [c] $\equiv$ (a [ac]) join (b [bc])

Binarni operator O je **komutativan** akko važi A O B $\equiv$ B O A. Unija, presek i spajanje su komutativni, a razlika i deljenje nisu. Binarni operator O je **idempotentan** akko važi A O A $\equiv$ A. Unija, presek i spajanje su idempotentni, a razlika i deljenje nisu.

Optimizator mora da vodi računa i o transformacijama aritmetičkih izraza jer se i oni mogu naći u okviru upita. Na primer, svaki logički izraz se može transformisati u ekvivalentni izraz u **konjuktivnoj normalnoj formi (KNF)**. KNF je izraz oblika C_1 and C_2 and . . . and C_n, pri čemu nijedan od izraza C_i ne sadrži konjunkciju. Prednost KNF je to što je ceo izraz netačan ako je bar jedan od od podizraza netačan. Optimizator može i da bira redosled izvršavanja tako da se prvo izvršavaju jednostavniji izrazi.

Semantičke transformacije su značajne, ali se retko sreću kod komercijalnih optimizatora jer su složeni. Na primer, u izrazu (dosije join ispit) [ocena], svaka torka iz tabele ispit se spaja sa nekom torkom tabele dosije (zbog veze primarnog i stranog ključa). Zbog toga će svaka torka tabele ispit biti u rezultatu, pa je spajanje nepotrebno i dovoljno je napisati ispit [ocena].

U najvećem broju slučajeva sistem vrši grupisanje torki prema zajedničkim vrednostima u određenim atributima. Na primer, kod spajanja se koriste:

- gruba sila, prave se sve moguće kombinacije torki u spajanju
- indeksi, direktan pristup uparenim torkama unutrašnje relacije spajanja
- heš, direktan pristup uparenim torkama unutrašnje relacije spajanja

---

Saveti za poboljšanje efikasnosti upita:

- u select klauzuli navoditi samo atribute koji su neophodni, izbegavati korišćenje *
- koristiti predikate koji prave restrikciju samo na one slučajeve koji su potrebni
- koristiti **optimize for** klauzulu ako je potreban značajno manji broj slogova od broja postojećih slogova u tabeli
- koristiti **for read only** and **for fetch only** klauzule
- isključiti **distinct** and **order by** gde nisu neophodni
- koristiti **union all** umesto union ako je moguće
- izbegavati konverziju numeričkih tipova
- atributi koji se porede treba da budu istog tipa
- ako je moguće koristiti sledeće tipove podataka:
  - **char** umesto varchar
  - **integer** umesto float, decimal ili decfloat
  - **decfloat** umesto decimal
  - **date**/**time** umesto karaktera
  - brojčane tipove umesto karaktera
- izbeći korišćenje **or** pri spajanju ako je moguće
- koristiti **in**
- ako je moguće izbegavati korišćenje skalarnih i matematičkih funkcija nad atributima u predikatu
- koristiti **group by** umesto distinct
- ne tražiti podatke koji su već poznati (ako je uslov da je godina_roka = 2020, ne izdvajati godinu_roka u select klauzuli)
- ako je moguće koristiti **case** umesto union
- davati prednost **SARGable (Search ARGument) atributima** - atributima po kojima može da se vrši pretraživanje

Pogledi

Pogledi predstavljaju virtuelne tabele. Moguće je kreiranje pogleda nad tabelama ili drugim pogledima. SQL kompilator upit nad pogledom konvertuje u ekvivalentan upit nad tabelom nad kojom je pogled napravljen, bilo da se radi o ažuriranju, unošenju ili brisanju.

Kreiranje i brisanje pogleda:

- **create view ime [spisak_kolona]?**
  as upit [with opcija check option]?
- **drop view ime**

Kroz spisak kolona se eksplicitno mogu imenovati kolone pogleda, ali to nije obavezno. Ako koristimo **with opcija check option** klauzulu, redovi koje želimo da ubacimo u tabelu preko pogleda moraju da zadovoljavaju uslove upita koji je korišćen za kreiranje pogleda. Ako je pogled kreiran nad drugim pogledom

onda je bitno koju opciju stavljamo, a inače ne. Opcija **local** znači da nove torke moraju da zadovolje sve uslove trenutnog pogleda, ali i pogleda nad kojim je on definisan. Opcija **cascaded** znači da nove torke moraju da zadovolje sve uslove trenutnog pogleda, ali i sve uslove svih pogleda od kojih on zavisi.

Ograničenja prilikom korišćenja pogleda:

- pogled se definiše nad jednom tabelom, u from klauzuli ne sme da se navede više od jedne tabele
- nije dozvoljeno koristiti agregatne funkcije, group by, having ili distinct u upitu
- ako želimo da unosimo podatke u tabelu pomoću pogleda, pogled mora da sadrži sve atribute koji su u toj tabeli obavezni

Prednosti pogleda su:

- nezavisnost podataka
- viđenje baze prilagođeno specifičnim grupama korisnika
- pojednostavljen rad sa podacima
- bezbednost (skrivanje podataka)

U novijem pristupu kreiranje pogleda uključuje relacione izraze. Više o tome ovde.

Sigurnost

Integritet predstavlja zaštitu podataka protiv autorizovanih korisnika, tj. obezbeđuje ispravnost i korektnost podataka. **Sigurnost** predstavlja zaštitu podataka od neautorizovanih korisnika, tj. zaštitu protiv neautorizovanih pristupa, promene ili uništenja baze. Sličnosti između sigurnosti i integriteta:

- sistem mora da bude svestan ograničenja koje korisnici ne smeju da prekrše
- ograničenja moraju da budu zadata od strane DBA u nekom jeziku
- ograničenja moraju da budu evidentirana u sistemskom katalogu
- SUBP mora da vrši nadzor nad operacijama korisnika

Aspekti sigurnosti su:

- pravni, socijalni i etički (uvid u stanje računa korisnika)
- fizička kontrola (fizičko obezbeđenje računarske sale)
- politička pitanja (odlučivanje ko i čemu sme da pristupi)
- operativni problemi (kako obezbediti tajnost lozinki)
- hardverska kontrola (da li CPU ima mogućnost hardverske zaštite programa)
- podrška OS (da li OS briše sadržaj memorije i diskova po završetku rada programa)
- problemi vezani za baze podataka (da li postoji koncept vlasnika podataka)

Jedinica podataka koja se osigurava može biti baza podataka, relacija, pojedinačna torka, šeme, indeksi i slično. Sigurnost se obično zapisuje preko kontrolne matrice pristupa. Predstavljanje se vrši po korisnicima,

po objektima i po dozvoli za pristup.

U sistem zaštite su uključena dva nezavisna mehanizma:

- **pogledi**, mogu da se koriste za sakrivanje osetljivih podataka od neautorizovanih korisnika
- **podsistem za autorizaciju**, dopušta korisniku sa određenim pravima pristupa da ta prava prenosi na druge korisnike ili da preneta prava povuče

---

Pogledi su imenovane relacije. Vrednost pogleda je rezultat izvršavanja relacionog izraza koji se navodi pri formiranju pogleda. Sistem pretvara taj upit u ekvivalentan upit nad osnovnim relacijama. Pogled na ovaj način može biti definisan i nad više tabela, a takođe kao i pre može biti definisan nad drugim pogledom. Kreiranje i brisanje pogleda:

- **var ime view relacioni_izraz [lista_kandidata_za_kljuceve]? [restrict | cascade]?**
- **drop var ime**

Lista kandidata za ključeve može biti prazna ako pogled može da nasledi kandidate za ključeve. Opcije **restrict** i **cascade** ne moraju da se navedu.

Definicija pogleda kombinuje spoljašnju šemu, spoljašnje/konceptualno preslikavanje i spoljašnje/spoljašnje preslikavanje. Funkcije pogleda:

- obezbeđuju zaštitu za skrivene podatke
- omogućavaju da različiti korisnici vide iste podatke na različite načine
- uprošćavaju složene operacije
- omogućavaju logičku nezavisnost podataka

Logička nezavisnost podataka podrazumeva da proširenje relacija baze i restruktuiranje baze ne smeju imati efekat na izvršavanje aplikativnih programa. Na primer, ako se relacija dosije razbije na dve relacije dosije1 i dosije2, može se formirati pogled nad te dve tabele i aplikativni programi bi trebalo bez problema da rade sa tim pogledom kao što su radili sa prvobitnom relacijom dosije.

Zlatno pravilo važi i za poglede, odnosno ažuriranje pogleda ne sme da naruši ograničenja integriteta nad pogledima. Ograničenja integriteta nad pogledima su izvedena iz ograničenja integriteta osnovnih relacija nad kojima su pogledi definisani. Ako ažuriramo neki pogled, kakve vrste ažuriranja treba izvesti nad osnovnim relacijama da bi se implementiralo originalno ažuriranje pogleda?

- **Codd-ov pristup**: definisanje pogleda koji mogu da se ažuriraju
- **Date-in pristup**: svi pogledi mogu da se ažuriraju, a ažuriranje se izvodi kao brisanje postojećih i unošenje novih podataka

U SQL-u pogled može da se ažurira ako važi:

- izraz kojim se definiše pogled je **upit** koji ne sadrži join, union, intersect, except, group by ili having
- **select** klauzula ne sadrži ključnu reč distinct i svaka njena stavka sadrži ime koje predstavlja atribut osnovne tabele
- **from** klauzula sadrži tačno jednu osnovnu tabelu ili tačno jedan pogled koji može da se ažurira
- **where** klauzula ne sadrži podupit u kom from klauzula referiše na istu tabelu kao from klauzula nadupita

---

Da bi korisnik izvršio bilo koju operaciju nad objektom on mora da posetuje dozvolu (autorizaciju) za tu operaciju nad tim objektom. **Sistemski administrator**, odnosno **SYSADM nivo autorizacije**, je inicijalni vlasnik svih dozvola. On može da daje dozvole, **grant naredbom**, kao i da ih povlači, **revoke naredbom**.

Davanje dozvole nad tabelama ili pogledima:

- **grant dozvola [on [tip]? objekat]? to korisnik [with grant option]?**

dozvola predstavlja jednu dozvolu ili listu dozvola razdvojenih zarezom koje želimo da dodelimo. Može se iskoristiti i fraza **all** ili **all privileges** ako želimo da dodelimo sve dozvole. Dozvola može biti **control**, **delete**, **insert**, **select**, **update**, **alter**, **index** (dozvola za kreiranje indeksa), **references** (dozvola za formiranje/brisanje spoljašnjeg ključa). Klauzula **on [tip]? objekat** se ne koristi kada dajemo sistemske privilegije. Ako dodeljujemo dozvole nad pojedinačnim objektima unosimo ih razdvojene zarezom. Navode se tip i ime objekta, a ako se tip ne navede podrazumeva se da je u pitanju tabela. Na kraju se navodi lista korisnika kojima želimo da dodelimo privilegije ili **public** ako želimo da ih dodelimo svim korisnicima. Ukoliko na kraju navedemo opciju **with grant option** onda će korisnici kojima smo dodelili dozvolu sami moći da dodeljuju tu dozvolu.

Povlačenje dozvole nad tabelama ili pogledima:

- **revoke dozvola [on [tip]? objekat]? from korisnik**

Primeri:

- grant select on table dosije to public
- grant delete on ispitni_rok to korisnik01
- grant all on table dosije, predmet, ispit to korisnik02, korisnik03
- grant select, update (sifra, naziv) on table predmet to korisnik04
- revoke select, update on table predmet from korisnik04 (nije moguće ukinuti update dozvolu samo za pojedine kolone, već za sve)