Integritet i sigurnost podataka

kolegij: Baze podataka

I. Batistić

`email: ivo@phy.hr`

Fizički odsjek, PMF

Integritet i sigurnost podataka

1. Integritet podataka
2. Istovremeni pristup podacima
3. Oporavak baze podataka
4. Zaštita od neovlaštenog pristupa podacima

1. Integritet baze podataka

Integritet: korektnost (dopuštene zdravorazumske vrijednosti) te konzistencija (međusobna suglasnost) podataka

Pravila za čuvanje integriteta

- Pravila za čuvanje integriteta domene
- Pravila za čuvanje integriteta unutar relacije
- Pravila za referencijski integritet

Integritet domene: Primjer

Promatrajmo bazu podataka o studentima na fakultetu. Ona sadrži niz podataka, pa između ostalog i dob:

STUDENT (**INDEKS**, IME, . . . , DOB)

Zahtijeva se da DOB bude cijeli broj između 10 i 60. Većina DBMSova omogućuje da se DOB definira kao INTEGER, onemogućujući upis:

- stringa (niza slova): ’pet’
- ili decimalnog broja: 15,3045

Ali neće spriječiti upis negativnog broja, npr. -5.

Integritet domene: Primjer 1.

Uglavnom, integritet domene može sačuvati jedino ako se ugrade odgovarajuće kontrole u aplikacijski program.

Tek neki DBMSovi omogućuju zadavanje i dodatnih uvjeta (ograničenja) na vrijednost atributa unutar zadanog tipa (PostgreSQL):

```
CREATE TABLE student (
      prezime VARCHAR(50) CHECK (prezime <> ''),
      ime     VARCHAR(50) CHECK (ime <>''),
      indeks  VARCHAR(10),
      .....
      dob     INTEGER CHECK ((dob >= 10) AND (dob <=60)),
      PRIMARY KEY (indeks)
);
```

Integritet domene: Primjer 2.

Ili:

```
CREATE TABLE student (
       prezime VARCHAR(50),
       ime     VARCHAR(50),
       indeks  VARCHAR(10),
       .....
       dob     INTEGER,
       PRIMARY KEY (indeks),
       CONSTRAINT student_check
                  CHECK ((prezime <> '') AND (ime <>'')
                  AND   (dob >= 10) AND (dob <=60))
);
```

Pravilo za čuvanje integriteta domene

Vrijednost atributa mora biti iz zadane domene.

Također: vrijednost primarnog atributa nije prazno (null).

Primjer: Prije navedeni primjer.

Pravila za integritet unutar relacije

Čuva korektnost veza između atributa unutar iste relacije.

Također:

- funkcionalna ovisnost atributa o ključu
- dvije n-torke unutar relacije ne smiju imati istu vrijednost ključa

Pravila za integritet unutar relacije

Često DBMSi ne pružaju mogućnost da se direktno izrazi jedinstvenost ključa. Npr. neke DBMS dopustit će da se upiše više istovjetnih n-torki u relaciju.

Rješenje: definirati gusti primarni indeks za ključni atribut.

Prilikom svake promijene u relaciju, DBMS automatski ažurira indeks. Narušavanje jedinstvenosti ključa u relaciji došlo bi pojavljivanjem dvaju jednakih vrijednosti ključnog atributa indeksu, što će DBMS automatski spriječiti.

Integritet relacije: Primjer 1.

```
CREATE TABLE kolegij (
        kid     INT,
        naslov  VARCHAR(100),
        ime     VARCHAR(50),
        prezime VARCHAR(50),
        PRIMARY KEY(kid)
);
```

Naredba: PRIMARY KEY(kid) će automatski izgraditi indeks.

Integritet relacije: Primjer 2.

Alternative:

```
CREATE TABLE kolegij (
        kid     INT,
        naslov  VARCHAR(100),
        ime     VARCHAR(50),
        prezime VARCHAR(50),
);

CREATE UNIQUE INDEX kolegij_kid ON kolegij (kid);
```

ili

```
CREATE UNIQUE INDEX kolegij_kid ON kolegij USING BTREE (kid);
```

Alternative za BTREE: RTREE, HASH, GIST (za PostgreSQL)

Integritet relacije: Brisanje indeksa

```
DROP INDEX kolegij_kid; -- opcija RESTRICT
```

ili

```
DROP INDEX kolegij_kid CASCADE;
```

Pravila za referencijski integritet

Čuvaju korektnost i konzistentnost veza među relacijama.

Radi se o pravilima koja se odnose na *strani ključ*, na atribut koji je ujedno atribut i ključ u drugoj relaciji. Svaka vrijednost takvog atributa u prvoj relaciji mora biti prisutna i u drugoj relaciji.

Referencijski integritet: Primjer

Baza podataka za neki fakultet:

STUDENT (**INDEKS**, IME_STUDENTA, STUPANJ)
KOLEGIJ (**KID**, NASLOV, IME_NASTAVNIKA)
IZVJEŠĆE (**KID**, **INDEKS**, OCJENA)

Relacija IZVJEŠĆE sadrži dva strana ključa: KID i INDEKS.

Referencijski integritet: Primjer

Pravila:

- Prije ubacivanja ili promjene n-torke u relaciji IZVJEŠĆE, provjeri da u relaciji STUDENT postoji n-torka s istom vrijednošću za INDEKS, te da u relaciji KOLEGIJ postoji n-torka s istom vrijednošću za KID.
- Prije izbacivanja n-torke iz relacije STUDENT, izbaci sve n-torke iz relacije IZVJEŠĆE s istom vrijednošću za INDEKS.
- Prilikom mijenjanja vrijednosti INDEKS n-torki relacije STUDENT izvrši odgovarajuću promjenu u svim n-torkama relacije IZVJEŠĆE koje imaju istu polaznu vrijednost za INDEKS.

Referencijski integritet: Primjer (nastavak)

Pravila:

- Prije izbacivanja n-torke iz relacije KOLEGIJ, izbaci i sve n-torke u relaciji IZVJEŠĆE s istom vrijednošću KID.
- Prilikom mijenjanja vrijednosti KID u n-torki relacije KOLEGIJ, izvrši odgovarajuće promjene u svim n-torkama relacije IZVJEŠĆE koje imaju istu polaznu vrijednost za KID.

Referencijski integritet: Primjer 1.

```
CREATE TABLE kolegij (
        kid       INT,
        naslov    VARCHAR(100),
        ime       VARCHAR(50),
        prezime   VARCHAR(50),
        PRIMARY KEY(kid)
);
CREATE TABLE student (
        indeks    INT,
        ime       VARCHAR(50),
        prezime   VARCHAR(50),
        stupanj   INT
        PRIMARY KEY(indeks)
);
CREATE TABLE izvjesce (
        kid       INT REFERENCES kolegij (kid),
        indeks    INT REFERENCES student (indeks),
        ocjena    INT
);
```

Referencijski integritet: Primjer 1.

Alternative:

```
CREATE TABLE izvjesce (
       kid      INT,
       indeks   INT,
       ocjena   INT,
       FOREIGN KEY (kid) REFERENCES kolegij (kid),
       FOREIGN KEY (indeks) REFERENCES student (indeks)
);
```

ili

```
CREATE TABLE izvjesce (
       kid      INT REFERENCES kolegij (kid)
                      ON DELETE CASCADE ON UPDATE CASCADE,
       indeks   INT REFERENCES student (indeks)
                      ON DELETE CASCADE ON UPDATE CASCADE,
                      -- default je: RESTRICT
       ocjena   INT
);
```

Napomena:

Rijetki su DBMSi koji mogu automatski izvršiti ova pravila. Obično teret pada na aplikacijske programe.

2. Istovremeni pristup podacima

Velike baze podatak moraju omogućiti većem broju korisnika istovremeni pristup do podataka. DBMS mora pažljivo koordinirati konfliktne radnje tako da svaki korisnik ima dojam kao da **sam** radi s bazom podataka.

Pojam transakcije
Više međusobno vezanih radnji na podacima koje čuvaju konzistenciju i korektnost pohranjenih podataka. Transakcije prevodi bazu iz jednog konzistentnog stanja u drugo konzistentno stanje.

Transakcija

Međustanja koja nastaju nastaju unutar jedne transakcije mogu biti *nekonzistentna*.

Transakcija mora biti u cjelosti izvršena, ili uopće ne smije biti izvršena.

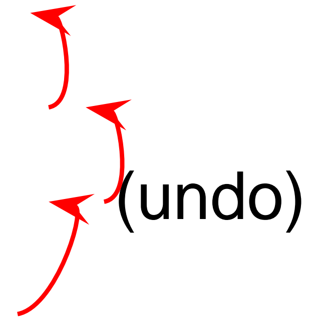
Ako transakcija nije do kraja bila obavljena (možda zbog greške u nekoj naredbi) treba biti neutralizirana - svi podaci koje je ona promijenila do trenutka prekida moraju dobiti natrad polazne vrijednosti.

Transakcija

START_TRANSACTION
    naredna 1;
    naredba 2;
    naredba 3;
    ⋮
    naredba k;
END_TRANSACTION

ili

START_TRANSACTION
    naredba 1;
    naredba 2;
    naredba 3;
    ⋮
    greška!
ABORT_TRANSACTION



Serijabilnost (serijalizabilnost)

Neka se u višekorisničkoj bazi podataka izvodi nekoliko transakcija paralelno tako da se pojedini dijelovi tih transakcija izvode vremenski izmiješano.

Ako je konačni učinak njihovog izvođenja isti kao da su se one izvršavale **serijski** (sekvencijalno), kažemo da se radi o **serijabilnom** (ili **serijalizabilnom**) izvršavanju transakcija.

Serijabilnost se može *a priori* garantirati.

Serijabilnost: Primjer 1.

Baza avionske tvrtke koja sadrži relaciju:

POLASCI (**BR_LETA**, **DATUM**, . . . , SL_SJEDALA)

Dva putnika istovremeno stižu u dvije različite poslovnice tvrtke i traže kartu za isti let istog dana. Neka u tom trenutku postoji 5 slobodnih mjesta na letu. Službenici sa svojih terminala, vezanih na isto računalo, pokreću dvi transakcije T_1 i T_2 koje DBMS *istovremeno* obavlja. Redoslijed operacija unutar transakcije je slijedeći:

Serijabilnost: Primjer 1. - nastavak

1. T_1 učitava odgovarajuću n-torku u svoj buffer.

2. T_1 u učitanoj n-torki smanjuje vrijednost za SL_SJEDALA sa 5 na 4. Promjena je učinjena samo na bufferu.

3. T_2 učitava odgovarajuću n-torku u svoj buffer. Učitana vrijednost za broj slobodnih mjesta, SL_SJEDALA, je još uvijek 5 jer nova vrijednost nije ažurirana.

Serijabilnost: Primjer 1. - nastavak

4. T_2 u učitanoj n-torki smanjuje vrijednost za SL_SJEDALA sa 5 na 4. Promjena je učinjena samo na bufferu, a ne u bazi podataka.

5. Promjenjena n-torka T_1 unosi se u bazu podataka, te vrijednost SL_SJEDALA postaje 4. jednaka

6. Promjenjena n-torka T_2 unosi se u bazu podataka, te vrijednost za SL_SJEDALA postaje 4. Pri tome je došlo do gubitka promjene zbog T_1.

Serijabilnost

Paralelno izvršavanje T_1 *i* T_2 *nije serijabilno !*
Kao rezultat dobiva se prebukirani avionski let.

DBMS mora garantirati serijabilnost. To se postiže posebnom dodatnom kontrolom kod istovremenog izvršavanja transakcija. Postoji više načina:

- Lokoti i dvofazni protokol zaključavanja
- Vremenski žigovi

Serijabilnost: Lokoti

Istovremeni pristup podacima se sprječava s tz. **lokotima** koji zaključavaju pojedine dijelove baze podataka dok jedna transakcija ne završi posao.

Zrnatost zaključavanja (granularity) određuje veličinu dijela baze koja je zaključana drugim transakcijama.

Krupnija zrnatost znači manji stupanj paralelnosti a jednostavniju kontrolu.

(Današnji DBMSi imaju zrnatost reda veličine n-torke ili nekoliko fizičkih blokova.)

Serijabilnost: Lokoti i zaglavljivanje

Primjer:

- T_1 zaključava podatak A.
- T_2 zaključava podatak B.
- T_1 traži lokot za B, ali mora čekati jer je T_2 već zaključao B.
- T_2 traži lokot za A, ali mora čekati jer je T_1 već zaključao A.

Očigledno ni T_1 ni T_2 više ne mogu nastaviti rad - one su se zaglavile.

DBMS koji koristi lokote mora osigurati da ne dođe do **zaglavljivanja** (deadlock).

Serijabilnost: Lokoti i zaglavljivanje

Postoji više rješenja, npr:

1. Zahtijeva se da svaka transakcija odjednom traži sve lokote koji joj trebaju, a ne jedan po jedan. Ako jedan od tih lokota nije dostupan, transakcija ne uzima ostale lokote, već čeka. Mana ove metode je da ona drastično može smanjiti stupanj paralelnosti rada. Naime, efekt je isti kao da se zrnatost povećala.

Serijabilnost: Lokoti i zaglavljivanje

Druga mogućnost:

2. Privremeno se tolerira zaglavljivanje, DBMS povremeno kreće u kontrolu da li ima zaglavljenosti transakcija. Kontrola se provodi gradnjom usmjerenog grafa u kojem čvorovi prestavljaju tekuće transakcije, a grana od čvora T_1 do čvora T_2 znači da T_1 čeka lokot koji drži T_2. Zatvorena petlja u grafu predstavlja skup međusobno zaglavljenih transakcija. DBMS prekida jednu od zaglavljenih transakcija, oslobađajući pri tome njene lokote i neutralizirajući njen dotadašnji učinak. Prekinuta transakcija se ponovo pokreće nakon izvjesnog vremena. Mana ovog pristupa je da dodatni posao kontrole DBMSa.

Serijabilnost: Lokoti i zaglavljivanje

To rješava problem zaglavljivanja u jednostavnim situacijama ali ne i u složenim !

Dvofazni protokol zaključavanja:
Ako u svakoj od transakcija sva zaključavanja slijede prije prvog otključavanja, tada je proizvoljno paralelno izvršavanje takvih transakcija serijabilno.

Zaključavanja i otključavanja zbivaju se u dvije razdvojene faze tijekom izvršavanja transakcije.

Serijabilnost: tri uvjeta

- Korištenje lokota
- Izbjegavanje zaglavljivanja
  (svi lokoti..., usmjereni graf ...)
- Primjena dvofaznog protokola zaključavanja

Serijabilnost: Primjer iz postgreSQL

```
START TRANSACTION ISOLATION LEVEL SERIALIZABLE;
  UPDATE student SET indeks='34342' WHERE indeks='34343';
  UPDATE predavanje SET indeks='34342' WHERE indeks='34343';
ROLLBACK TRANSACTION;
```

Serijabilnost: Vremenski žigovi

Svakoj se transakciji pridružuje **identifikacijski broj**, ili **vremenski žig**. U slučaju jednoprocesorskog sistema, žig može biti trenutak početka transakcije. Istovremeni pristup odvija se tako da se sva čitanja i upisi istog podatka obavljaju u redoslijedu vremenskih žigova prigodnih transakcija. Time se garantira serijabilnost: Ukupni učinak svih transakcija je isti kao da se svaka od njih izvršava trenutačno, u svom posebnom vremenskog trenutku.

Serijabilnost: Vremenski žigovi

Primjer: iz avionske tvrtke
Zahtjev za čitanjem ili upisom transakcije s žigom T_1*, neće se odobriti ako je dotični podatak bio upisan od transakcije sa žigom* T_2*, gdje je* $T_2 > T_1$*. Sličan, zahtijev za upisom će biti odbijen ako je podatak već bio čitan od transakcije sa većim žigom.*

Kad transakcija doživi odbijanje zahtjeva za upisom ili čitanjem, ona se prekida, a njen dotadašnji učinak se neutralizira. Zatim se transakcija ponovo pokreće sa većim vremenskim žigom.

3. Oporavak baze podataka

Baza podataka se može naći u nekonzistentnom stanju zbog prekida transakcije, pogrešnog rada transakcije, ili nekih drugih *hardware*skih ili *software*skih razloga.

Do pogrešnog rada transakcije može doći zbog:

- Logičke greške u samoj transakciji.
- *software*ske greške u DBMSu ili operacijskom sustavu
- *hardware*ske greške u računalu

3. Oporavak baze podataka

Do prekida transakcije može doći:

- Zbog odustajanja od same transakcije
- Kao posljedica kontrole istovremenog izvršavanja više transakcija
- zbog *software*ske greške u DBMSu ili operacijskom sustavu
- *hardware*ske greške u računalu

Oporavak baze podataka

DBMS mora omogućiti oporavak baze, tj. njen povratak iz nekonzistentnog **blisko konzistentno** stanje.

To se postiže:

- rezervnom kopijom baze podataka
- žurnal datotekom

(*blisko konzistentno* = *vremenski najfriškije konzistentno stanje*)

Oporavak: Rezervna kopija

Rezervna kopija baze podataka (backup copy) se dobiva se snimanjem cijele baze podataka na magnetske trake, i to u trenutku kad smatramo da je baza u konzistentnom stanju.

Za vrijeme kopiranja ne smije se obavljati nikakva transakcija koja mijenja podatke.

Stvaranje rezervne kopije je dugotrajna operacija, koja osim toga ometa redovni rad korisniku. Zato se kopiranje ne obavlja suviše često, već periodički u unaprijed predviđenim terminima npr. jednom tjedno.

Oporavak: Žurnal datoteka

U **Žurnal datoteku** (journal file) se ubilježava **povijest** rada svake transakcije.

Za svaku transakciju žurnal datoteka sadrži:

- identifikator transakcije
- adresu svakog podatka kojeg je transakcija stvorila ili promjenila zajedno s prethodnom vrijednošću tog podatka (pre-image) i sa novom vrijednošću (post-image)
- kontrolne točke (checkpoints) u napredovanju transakcije

Oporavak: Žurnal datoteka

Poželjno je da žurnal datoteka sadrži tzv. **točke isporuke** (commintments).

To je kontrolna točka kojom transakcija bilježi u žurnal datoteci da smatra da je sve svoje operacijama uspješno završila.

Oporavak: Dvofazni postupak isporuke

Dvofazni postupak isporuke: (two-phase commitment policy)

- Transakcija upisuje u žurnal sve promjene koje bi trebalo napraviti u bazi podataka, te nakon toga bilježi točku isporuke.
- Čim je točku isporuke ubilježena, DBMS prenosi promjene iz žurnal datoteke u bazu podataka.

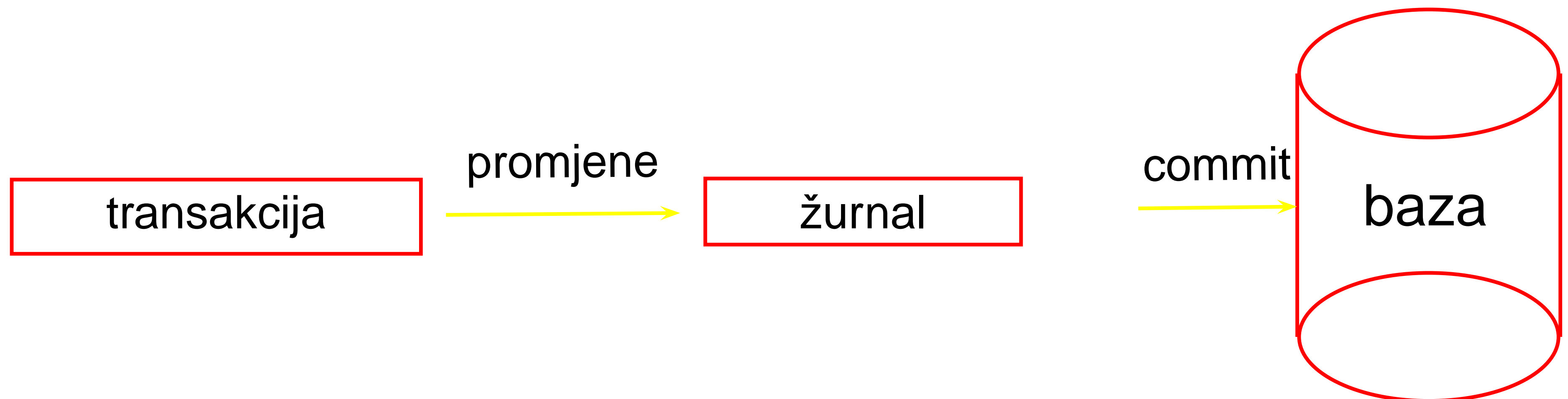
Dakle, promjene podataka, nastale kao rezultat rada transakcije, ne proslijeđuju se u bazu odmah nego tek nakon što transakcija unese u žurnal točku isporuke.

Oporavak: Dvofazni postupak isporuke

Napomena: Ako je transakcija zbog nečega prekinuta, ona ne ubilježava točku isporuke u žurnal datoteku, pa stoga DBMS ne prenosi njene promjene u bazu podataka.

Neutralizacija prekinute transakcije je trivijalna.

Skica dvofaznog postupka isporuke:



Oporavak: Odmotavanje unatrag

Odmotavanje unatrag (roll-back) primjenjuje se kod manjih nekonzistencija. Npr. za neutralizaciju jedne transakcije.

Sastoji od dva koraka:

- Čitamo žurnal i pronalazimo stare vrijednosti (pre-images) podataka koje je transakcija mijenjala
- te stare vrijednosti ponovo uspostavljamo u bazi

Napomena:

Osim toga, ako je neka druga transakcija pročitala u bazi podataka vrijednosti unesene od upravo neutralizirane transakcije, tada treba i tu drugu transakciju na isti način neutralizirati.

Oporavak: Odmotavanje unaprijed

Odmotavanje unaprijed (roll-forward) primjenjuje se kod većih oštećenja baze. Zahtijeva da u žurnal datoteku budu upisane kontrolne točke u kojima smatramo da je baza još bila konzistentna. Obično te kontrolne točke odgovaraju trenucima kad nije bilo aktivnih transakcija.

Procedura:

- Ponovno uspostavimo stanje stare baze zabilježene zadnjom rezervnom kopijom (snimljene na magnetsku traku).
- Odredimo zadnju kontrolnu točku u žurnal datoteci.

Oporavak: Odmotavanje unaprijed

Procedura - nastavak:

- Čitamo dio žurnala od početka do zadnje kontrolne točke; ponovo unosimo u bazu **promjene** podatke (post-images), i to istim redom za svaku isporučenu transakciju u promatranom dijelu žurnala. Time se uspostavlja stanje baze podataka koje odgovaraju zadnjoj kontrolnoj točci.

Napomena:
Primijetimo da odmotavanje unaprijed omogućuje reproduciranje baze koja je bila u potpunosti uništena (zbog kvara na računalu ili kvara diska).