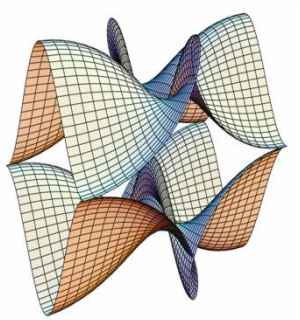
Sveučilište u Zagrebu
PMF – Matematički odsjek

BAZE PODATAKA
Predavanja 2019/2020



Poglavlje 7: Integritet i sigurnost baze podataka

Sastavio: Robert Manger
10.05.2020

Općenito o integritetu i sigurnosti

- Integritet i sigurnost važni su za ispravan rad baze nakon što je ona ušla u redovitu uporabu.
  - U prvom redu su briga administratora baze.
  - Donekle su i briga projektanta.
- Upoznat ćemo se s nizom mehanizama za čuvanje integriteta i sigurnosti.
  - neke od njih automatski pokreće DBMS,
  - drugi se implementiraju onda ako ih projektant ugradi u svoju fizičku shemu,
  - treći stoje na raspolaganju administratoru koji ih pokreće po potrebi.

Sadržaj Poglavlja 7

Općenito o čuvanju integriteta (1)

- Čuvati *integritet* baze znači čuvati korektnost i konzistentnost podataka.
  - *Korektnost*: svaki podatak ima ispravnu vrijednost.
  - *Konzistentnost*: različiti podaci međusobno su usklađeni, dakle ne protuslove jedan drugome.
- Integritet baze lako bi se mogao narušiti na primjer pogrešnim radom aplikacija.
- Voljeli bismo kad bi se baza podataka sama mogla braniti od narušavanja integriteta.
- Zato suvremeni DBMS-i dozvoljavaju projektantu da definira takozvana *ograničenja* (*constraints*).

Općenito o čuvanju integriteta (2)

- *Ograničenja* su uvjeti (pravila) koje korektni i konzistentni podaci moraju zadovoljavati.
  - Projektant uvodi ograničenja tako da ih upiše u fizičku shemu baze.
  - Uvedena ograničenja DBMS će uključiti u konačnu realizaciju baze.
  - U kasnijem radu kod svake promjene podataka DBMS će provjeravati jesu li sva ograničenja zadovoljena.
  - Ako neko ograničenje nije zadovoljeno, tada DBMS neće izvršiti traženu promjenu, već će dotičnoj aplikaciji poslati poruku o greški.
- Dalje opisujemo tri vrste ograničenja, te načine njihovog uvođenja u fizičku shemu.

Ograničenja za integritet domene (1)

- *Ograničenja za integritet domene* izražavaju činjenicu da vrijednost atributa mora biti iz odgovarajuće domene.
  - Zahtjev da vrijednost primarnog atributa ne smije biti prazna također spada u ovu kategoriju.
- Ograničenje na integritet domene najčešće se uvodi tako da se u naredbi CREATE TABLE atributu pridruži odgovarajući tip, uz eventualnu klauzulu NOT NULL.
  - No popis podržanih tipova obično je siromašan, tako da pridruživanjem tipa često ne uspijevamo u potpunosti izraziti potrebno ograničenje.

Ograničenja za integritet dom (2)

- Gledamo početnu fizičku shemu za bazu o fakultetu.
- Za neke atribute uspjeli smo precizno odrediti tipove i osigurati integritet domene.
- Npr, GODINA_STUDIJA može poprimati samo vrijednosti iz navedene liste.

```
CREATE TABLE STUDENT
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   GODINA_STUDIJA ENUM('1','2','3','4','5'),
   PRIMARY KEY(JMBAG));

CREATE TABLE PREDMET
  (SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   NASLOV CHAR(80),
   IME_ZAVODA CHAR(40),
   OIB_NASTAVNIKA NUMERIC(11) UNSIGNED,
   SEMESTAR ENUM('Z','L'),
   ECTS_BODOVI NUMERIC(2) UNSIGNED,
   PRIMARY KEY(SIFRA_PREDMETA));

CREATE TABLE NASTAVNIK
  (OIB NUMERIC(11) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   IME_ZAVODA CHAR(40),
   BROJ_SOBE NUMERIC(3) UNSIGNED,
   PLACA NUMERIC(5) UNSIGNED,
   PRIMARY KEY(OIB));

CREATE TABLE ZAVOD
  (IME_ZAVODA CHAR(40) NOT NULL,
   OIB_PROCELNIKA NUMERIC(11) UNSIGNED,
   OPIS_DJELATNOSTI CHAR(160),
   PRIMARY KEY(IME_ZAVODA));

CREATE TABLE UPISAO
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   DATUM_UPISA DATE,
   OCJENA ENUM('2','3','4','5'),
   PRIMARY KEY(JMBAG,SIFRA_PREDMETA));
```

Ograničenja za integritet domene (3)

- Zamislimo da se naš fakultet nalazi u zgradi koja ima tri etaže, označene s 1, 2 i 3.
- Pretpostavimo da su brojevi soba tako oblikovani da prva znamenka odgovara etaži.
- Tada tip NUMERIC(3) UNSIGNED kojeg smo pridružili atributu BROJ_SOBE ne čuva integritet domene.
- Mnogi DBMS-i dozvoljavaju da se u shemu ugradi i precizniji uvjet koji vrijednosti atributa moraju zadovoljavati.
- Takav uvjet može se uklopiti u naredbu CREATE TABLE ili se može pojaviti kao zasebna naredba.

Ograničenja za integritet domene (4)

- Evo jednog načina kako se uvođenjem dodatnog uvjeta može osigurati integritet domene za naš atribut BROJ_SOBE u zgradi s tri etaže.
  - Uvjet je zadan unutar naredbe CREATE TABLE NASTAVNIK.
  - Iznimno rabimo sintaksu MS SQL Server-a. Slično rješenje postoji i u Oracle-u.

```
CREATE TABLE NASTAVNIK
  (OIB NUMERIC(11) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   IME_ZAVODA CHAR(40),
   BROJ_SOBE NUMERIC(3)
     CHECK (BROJ_SOBE BETWEEN 101 AND 399),
   PLACA NUMERIC(5) UNSIGNED,
   PRIMARY KEY(OIB));
```

Ograničenja za integritet unutar rel (1)

- *Ograničenja za čuvanje integriteta unutar relacije* čuvaju korektnost veza između atributa.
  - Najvažniji primjer je ograničenje koje traži da dvije *n*-torke unutar iste relacije ne smiju imati jednaku vrijednost primarnog ključa.
  - Slično ograničenje može se postaviti i za atribut koji je kandidat za ključ.
- Ograničenje za svojstvo ključa uvodi se:
  - tako da se u naredbu CREATE TABLE stavi klauzula PRIMARY KEY odnosno UNIQUE,
  - ili eksplicitnim stvaranjem primarnih indeksa naredbom CREATE UNIQUE INDEX.

Ograničenja za integritet unutar rel (2)

- U našoj fizičkoj shemi za bazu o fakultetu sve naredbe CREATE TABLE imaju odgovarajuće klauzule PRIMARY KEY.
  - Znači, već u polaznoj verziji sheme osigurali smo da se čuva svojstvo ključa.
  - Umjesto klauzule PRIMARY KEY mogli smo uporabiti naredbu CREATE UNIQUE INDEX.

- Zamislimo da na našem fakultetu vrijedi pravilo da svaki nastavnik mora imati zasebnu sobu.
- Tada atribut BROJ_SOBE postaje kandidat za ključ unutar relacije NASTAVNIK.

Ograničenja za integritet unutar rel (3)

- Dva načina kako se u MySQL-u uvođenjem dodatnog ograničenja može osigurati svojstvo ključa za BROJ_SOBE:
  - Ubacivanjem klauzule UNIQUE u postojeću naredbu CREATE TABLE NASTAVNIK.
  - Dodavanjem nove naredbe CREATE UNIQUE INDEX, naredba CREATE TABLE ostaje nepromijenjena.

```
CREATE TABLE NASTAVNIK
  (OIB NUMERIC(11) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   IME_ZAVODA CHAR(40),
   BROJ_SOBE NUMERIC(3) UNSIGNED,
   PLACA NUMERIC(5) UNSIGNED,
   PRIMARY KEY(OIB),
   UNIQUE(BROJ_SOBE));
```

```
CREATE UNIQUE INDEX NB_IND ON NASTAVNIK (BROJ_SOBE);
```

Ograničenja za referencijalni integ(1)

- *Ograničenja za čuvanje referencijalnog integriteta* služe zato da se očuva konzistentnost veza između relacija.
  - Najčešće je riječ o ograničenjima koja se odnose na strani ključ, dakle na atribut u jednoj relaciji koji je ujedno primarni ključ u drugoj relaciji.
  - Svaka vrijednost takvog atributa u prvoj relaciji mora biti prisutna i u drugoj relaciji.

- U današnjim DBMS-ima ograničenje kojim se čuva svojstvo stranog ključa uvodi se tako da se u odgovarajuću naredbu CREATE TABLE stavi klauzula FOREIGN KEY … REFERENCES ….

Ograničenja za ref integritet (2)

- Nova verzija fizičke sheme za bazu o fakultetu, u sintaksi MySQL-a.
- Klauzule FOREIGN KEY uvode ograničenja za čuvanje integriteta svih stranih ključeva osim OIB PROČELNIKA u relaciji ZAVOD.

```
CREATE TABLE STUDENT
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   GODINA_STUDIJA ENUM('1','2','3','4','5'),
   PRIMARY KEY(JMBAG))
   ENGINE=INNODB;

CREATE TABLE ZAVOD
  (IME_ZAVODA CHAR(40) NOT NULL,
   OIB_PROCELNIKA NUMERIC(11) UNSIGNED,
   OPIS_DJELATNOSTI CHAR(160),
   PRIMARY KEY(IME_ZAVODA))
   ENGINE=INNODB;

CREATE TABLE NASTAVNIK
  (OIB NUMERIC(11) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   IME_ZAVODA CHAR(40),
   BROJ_SOBE NUMERIC(3) UNSIGNED,
   PLACA NUMERIC(5) UNSIGNED,
   PRIMARY KEY(OIB),
   INDEX NI_IND (IME_ZAVODA),
   FOREIGN KEY (IME_ZAVODA) REFERENCES ZAVOD(IME_ZAVODA))
   ENGINE=INNODB;

CREATE TABLE PREDMET
  (SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   NASLOV CHAR(80),
   IME_ZAVODA CHAR(40),
   OIB_NASTAVNIKA NUMERIC(11) UNSIGNED,
   SEMESTAR ENUM('Z','L'),
   ECTS_BODOVI NUMERIC(2) UNSIGNED,
   PRIMARY KEY(SIFRA_PREDMETA),
   INDEX PI_IND (IME_ZAVODA),
   INDEX PO_IND (OIB_NASTAVNIKA),
   FOREIGN KEY (IME_ZAVODA) REFERENCES ZAVOD(IME_ZAVODA),
   FOREIGN KEY (OIB_NASTAVNIKA) REFERENCES NASTAVNIK(OIB))
   ENGINE=INNODB;

CREATE TABLE UPISAO
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   DATUM_UPISA DATE,
   OCJENA ENUM('2','3','4','5'),
   PRIMARY KEY(JMBAG, SIFRA_PREDMETA),
   INDEX UJ_IND (JMBAG),
   INDEX US_IND (SIFRA_PREDMETA),
   FOREIGN KEY (JMBAG) REFERENCES STUDENT(JMBAG),
   FOREIGN KEY (SIFRA_PREDMETA) REFERENCES
     PREDMET(SIFRA_PREDMETA))
   ENGINE=INNODB;
```

Ograničenja za referencijalni integ(3)

- Nakon realizacije prethodne sheme, DBMS na primjer neće dozvoliti:
  - da se u relaciju UPISAO ubaci *n*-torka s vrijednošću JMBAG koje nema u relaciji STUDENT.
  - da se iz relacije PREDMET briše *n*-torka sa ŠIFROM PREDMETA koja se pojavljuje u relaciji UPISAO.
- U prethodnoj shemi važan je redoslijed naredbi CREATE TABLE.
  - Relacija sa stranim ključem može stvoriti tek nakon što već postoji ona relacija koju taj strani ključ referencira.

Ograničenja za referencijalni integ(4)

- Npr. relacija NASTAVNIK se zbog stranog ključa IME ZAVODA mora stvarati nakon relacije ZAVOD.
- Pritom nije moguće u relaciji ZAVOD istovremeno zadati i strani ključ OIB PROČELNIKA jer bi taj strani ključ zahtijevao suprotni redoslijed stvaranja.

- U zadnjoj shemi vidimo neke specifičnosti MySQL:
  - Za sve datoteke umjesto standardne indeks-sekvencijalne organizacije rabi se organizacija InnoDB koja jedina omogućuje provjeru stranog ključa.
  - Uz svaki strani ključ eksplicitno se deklarira sekundarni indeks koji će služiti za traženje po tom ključu.

Ograničenja za referencijalni integ(5)

- Provjera ograničenja za referencijalni integritet predstavlja teret za DBMS i usporava rad.
  - Projektant treba odmjeriti koja ograničenja su stvarno potrebna, a koja se mogu zanemariti ili zamijeniti provjerama u aplikacijama.
  - Prethodna shema je pretjerivanje jer smo za 5 relacija uveli 5 referencijalnih integriteta.
  - Vjerojatno bi bilo dovoljno provjeravati samo sekundarne ključeve u relaciji UPISAO, budući da se jedino tu radi o nešto većoj količini podataka.

Sadržaj Poglavlja 7

7.1. Čuvanje integriteta

7.2. Sigurnost baze

7.3. Istovremeni pristup

Općenito o sigurnosti baze (1)

- Baza podataka predstavlja dragocjen resurs.
  - Njezin nastanak i održavanje iziskuju goleme količine ljudskog rada.
- Zato se od DBMS-a očekuje da on u što većoj mjeri garantira sigurnost podataka.
  - Ne smije desiti da podaci budu uništeni ili oštećeni zbog tehničkog kvara, pogrešnih transakcija, nepažnje korisnika ili zlonamjernih radnji.
- Današnji DBMS-i raspolažu djelotvornim mehanizmima za sigurnost.

Općenito o sigurnosti baze (2)

- Objasnit ćemo kako projektant ili administrator baze mogu postići da se mehanizmi za sigurnost aktiviraju tijekom rada baze.
  - Prvi odjeljak bavi se tehničkim aspektom sigurnosti, dakle oporavkom baze u slučaju njezinog većeg ili manjeg oštećenja.
  - Ostali odjeljci bave se suptilnijim aspektom koji se tiče zaštite baze od neovlaštenih radnji korisnika.

Stvaranje pretpostavki za oporavak (1)

- Rad s bazom svodi se na pokretanje *transakcija*.
  - Jedna transakcija s korisničkog stanovišta predstavlja jednu nedjeljivu cjelinu.
  - No ona se realizira kao niz od nekoliko elementarnih zahvata u samoj bazi.
- Primjer transakcije je bankovna transakcija.
  - Zadani novčani iznos prebacuje se s jednog bankovnog računa na drugi.
  - Takvo prebacivanje predstavlja jednu logičku cjelinu.
  - No ono se realizira kroz dvije zasebne promjene u bazi: smanjivanje salda na jednom računu, povećanje salda na drugom.

Stvaranje pretpostavki za oporavak (2)

- Osnovna svojstva transakcije:
  - Prevodi bazu iz jednog konzistentnog stanja u drugo.
  - Među-stanja koja nastaju nakon pojedinih operacija unutar transakcije mogu biti nekonzistentna.
  - Da bi se očuvao integritet baze, transakcija mora u cijelosti biti izvršena ili uopće ne smije biti izvršena.
  - Transakcija koja iz bilo kojeg razloga nije do kraja bila obavljena morala bi biti *neutralizirana*.
  - Svi podaci koje je ona do trenutka prekida promijenila morali bi natrag dobiti svoje polazne vrijednosti.

Stvaranje pretpostavki za oporavak (3)

- Baza podataka se u toku svog rada može naći u neispravnom stanju.
  - Najčešći razlog je baš neispravno izvedena transakcija, dakle transakcija koja se počela izvršavati, nije bila obavljena do kraja, no nije bila ni neutralizirana.
  - Mnogo rjeđi razlozi su: pogrešno sastavljena transakcija, softverske greške u DBMS-u ili operacijskom sustavu te hardverske greške.
- Od DBMS-a se očekuje da u svim slučajevima oštećenja baze omogući njezin *oporavak*:
  - povratak u stanje koje je što ažurnije i pritom još uvijek konzistentno.

Stvaranje pretpostavki za oporavak (4)

- Da bi oporavak zaista bio moguć, mora biti ispunjena bar neka od sljedećih pretpostavki.
  - **Uključen je DBMS-ov mehanizam za upravljanje transakcijama.**
    - DBMS očekuje od aplikacije da ona eksplicitno najavi početak transakcije, te da objavi njezin završetak.
    - Pritom završetak može biti potvrda (*commit*) da je transakcija ispravno obavljena, ili njezin opoziv (*rollback*).
    - Za vrijeme izvršavanja transakcije DBMS samo privremeno pamti promjene u bazi.
    - U slučaju potvrde, te promjene se trajno upisuju u bazu, a u slučaju opoziva one se neutraliziraju odnosno zaboravljaju.

Stvaranje pretpostavki za oporavak (5)

- **Povremeno se stvara rezervna kopija baze.**
  - Ona se dobiva snimanjem cijele baze na drugi medij, i to u trenutku kad smatramo da je baza u konzistentnom stanju.
  - Stvaranje kopije je dugotrajna operacija koja može ometati redovni rad korisnika.
  - Zato se kopiranje ne obavlja suviše često, već periodički u unaprijed predviđenim terminima – na primjer jednom tjedno.
- **Održava se žurnal-datoteka.**
  - To je datoteka gdje je ubilježena „povijest“ svake transakcije koja je mijenjala bazu nakon zadnjeg stvaranja rezervne kopije.
  - Za jednu transakciju žurnal evidentira adresu svakog zapisa kojeg je transakcija promijenila, zajedno s prethodnom vrijednošću tog zapisa i novom vrijednošću.

Stvaranje pretpostavki za oporavak (6)

- Navedene pretpostavke za oporavak baze zaista omogućuju razne oblike oporavka.
  - DBMS-ovom kontrolom transakcija uz mogućnost opoziva znatno se smanjuje broj transakcija koje će oštetiti bazu svojim djelomičnim izvođenjem.
    - Kad aplikacija ustanovi da se transakcija ne može izvršiti do kraja, ona će je opozvati, a DBMS će neutralizirati promjene.
  - Žurnal datoteka omogućuje neutralizaciju transakcije koja je došla do kraja i trajno je promijenila bazu, ali se naknadno utvrdilo da je bila pogrešna ili nepotrebna.
    - Rabi se postupak odmotavanja unatrag (*roll-back*).

Stvaranje pretpostavki za oporavak (7)

- Rezervna kopija baze omogućuje ponovno uspostavljanje baze nakon znatnijeg oštećenja.
  - Postupak se svodi na presnimavanje svih podataka iz rezervne kopije natrag u bazu.
  - Time se uspostavlja stanje zabilježeno zadnjom rezervnom kopijom (možda od prošlog tjedna).
- Rezervna kopija i žurnal datoteka zajedno omogućuju još bolji oporavak baze koja je pretrpjela znatnije oštećenje.
  - Najprije se uspostavlja stanje iz zadnje rezervne kopije.
  - Zatim se rabi postupak odmotavanja unaprijed (*roll-forward*).

Stvaranje pretpostavki za oporavak (8)

- Mehanizmi i sredstva za oporavak baze mogu se naknadno uključivati i dodavati po potrebi.
- Ipak, odluka koja će se sredstva rabiti zapravo spada u nadležnost projektanta.
  - Uključivanje pojedinog mehanizma donosi veću sigurnost ali opterećuje rad i smanjuje performanse.
  - Projektant treba procijeniti u kojoj mjeri se isplati žrtvovati performanse zbog veće sigurnosti.
  - U mnogim DBMS-ima uporaba određenih mehanizama zaštite moguća je samo ako su datoteke organizirane na odgovarajući način.
  - Takva ograničenja projektant mora uzeti u obzir kod oblikovanja fizičke građe.

Stvaranje pretpostavki za oporavak (9)

- Pokazujemo kako se mehanizmi i sredstva za oporavak baze realiziraju u MySQL-u.
  - Upravljanje transakcijama u MySQL-u.
    - MySQL po defaultu ne upravlja transakcijama, to se zove *autocommit mode*.
    - No taj default može se promijeniti tako da aplikacija ili korisnik pošalju naredbu SET AUTOCOMMIT = 0;
    - Pod uvjetom da su sve datoteke tipa InnoDB, moguće je dalje upravljati transakcijama.
    - Početak transakcije mora se eksplicitno označiti naredbom BEGIN;
    - Potvrda da je transakcija uredno završila postiže se naredbom COMMIT;
    - Opoziv neuspjele transakcije postiže se naredbom ROLLBACK;

Stvaranje pretpostavki za oporav (10)

- Stvaranje rezervne kopije MySQL baze.
  - Obavlja se pozivom uslužnog programa mysqldump.
  - Program se poziva iz komandne ljuske operacijskog sustava i prima opcije koje određuju koja baza će se presnimiti kamo.
- Žurnal datoteke u MySQL-u.
  - MySQL po defaultu održava žurnal datoteku.
  - Naime, sam DBMS realiziran je kao program mysqld . Taj program prilikom pokretanja čita komandnu datoteku koja u svojem standardnom obliku sadrži opciju oblika

    --log-update=*ime_datoteke* .
  - Time je određeno ime datoteke u koju će se upisivati sve promjene baze.
  - Ako želimo isključiti žurnal datoteku, tada treba zaustaviti mysqld i ponovo ga pokrenuti bez navedene opcije.

Stvaranje pretpostavki za oporav (11)

- Primjer upravljanja transakcijama u MySQL-u: plaća nastavnika Turinga smanjuje se za 1000, a plaća nastavnika Pascala povećava za 1000.

```
SET AUTOCOMMIT = 0;

BEGIN;

UPDATE NASTAVNIK
   SET PLACA = PLACA – 1000  WHERE PREZIME = 'Turing';

UPDATE NASTAVNIK
   SET PLACA = PLACA + 1000  WHERE PREZIME = 'Pascal';

COMMIT; (ili ROLLBACK;)

SET AUTOCOMMIT = 1;
```

Davanje ovlaštenja korisnicima (1)

- Zaštita podataka od neovlaštene uporabe postiže se tako da se SQL naredbama GRANT i REVOKE korisnicima dodjeljuju ili uskraćuju ovlaštenja.
- Uvođenje korisnika i upravljanje ovlaštenjima obično je posao administratora baze.
  - Ipak, dobro je da projektant već predvidi nekoliko tipičnih korisnika, te predloži njihova ovlaštenja.
  - Projektantovi naputci mogu kasnije služiti administratoru kao obrazac za uvođenje daljnjih korisnika.

Davanje ovlaštenja korisnicima (2)

- Ovlaštenja u MySQL-u:
  - Ovlaštenje je pridruženo „e-mail adresi“ ime_korisnika@ime_racunala. Ista osoba može imati drukčija ovlaštenja kad se prijavljuje s drugog računala.
  - Uobičajena ovlaštenja su: SELECT, INSERT, DELETE, UPDATE, ALTER, CREATE, DROP, ALL.
  - Doseg ovlaštenja može biti:
    - globalno za sve baze koje kontrolira dotična instalacija MySQL (*.*),
    - za jednu takvu bazu (ime_baze.*),
    - za jednu relaciju jedne baze (ime_baze.ime_relacije),
    - ili čak za pojedine atribute unutar jedne relacije.

Davanje ovlaštenja korisnicima (3)

- Prvi primjer davanja ovlaštenja u MySQL-u:
  - administrator neregistriranom (anonymous) korisniku dozvoljava pretraživanje fakultetske baze.

    > GRANT SELECT on fakultet.* TO ''@localhost;
  - Nakon ove naredbe GRANT, moguća je ovakva sesija neregistriranog korisnika:

    $ mysql –u nepoznato_ime
    > SELECT USER( );
    (ispisuje se: nepoznato_ime@local host)
    > USE fakultet;
    > SELECT * FROM PREDMET;
    …*(ispis)* ...
    > INSERT INTO PREDMET VALUES
    (33333, 'Novi naslov' , 'Zavod za racunarstvo', 33571209458, 'L', 5));
    …*(poruka o greški)* ...

Davanje ovlaštenja korisnicima (4)

- Drugi primjer davanja ovlaštenja u MySQL-u:
  - Administrator stvara korisnika someuser s lozinkom loz000, koji se prijavljuje s lokalnog računala, može raditi sve što hoće u bazi s imenom someuserbase te smije pretraživati fakultetsku bazu fakultet.

```
> CREATE DATABASE someuserbase;
> GRANT ALL ON someuserbase.* TO
    someuser@localhost IDENTIFIED BY 'loz000';
> GRANT SELECT ON fakultet.* TO someuser@localhost;
```

  - Jedna moguća sesija korisnika someuser.

```
$ mysql –u someuser –p
  Enter password: ****** (loz000)
```

Davanje ovlaštenja korisnicima (5)

```
> SELECT USER( );
   (ispisuje se: someuser@localhost)
> USE tudja_baza;
> SHOW TABLES;
   … (poruka o greški) ...
> USE someuserbase;
 > CREATE TABLE PROBA
     (ID NUMERIC(2) UNSIGNED NOT NULL,
     NAZIV CHAR(20),
     PRIMARY KEY(ID));
 > INSERT INTO PROBA VALUES (44, 'Novi naziv');
 > USE fakultet;
 > SELECT * FROM PREDMET;
   …  (ispis)  ...
> INSERT INTO PREDMET VALUES
      (33333, 'Novi naslov', 'Zavod za racunarstvo', 33571209458, 'L', 5));
   … (poruka o greški) ...
```

Davanje ovlaštenja korisnicima (6)

- Primjer oduzimanja ovlaštenja u MySQL-u:
  - naredbe kojima administrator korisniku someuser oduzima pravo čitanja relacija PREDMET, ZAVOD i UPISAO u fakultetskoj bazi fakultet.

```
> REVOKE SELECT ON fakultet.* FROM ''@localhost;
> REVOKE SELECT on fakultet.* FROM someuser@localhost;
> GRANT SELECT ON fakultet.STUDENT TO someuser@localhost;
> GRANT SELECT ON fakultet.NASTAVNIK TO someuser@localhost;
```

  - Zatim je moguća sljedeća sesija korisnika someuser:

```
$ mysql –u someuser -p
  Enter password: ****** (loz000)
 > USE fakultet;
 > SELECT * FROM PREDMET;
    …(poruka o greški) ...
 > SELECT * FROM NASTAVNIK;
   …(ispis) ...
```

Pogledi kao mehanizam zaštite (1)

- Osim za postizavanje logičke nezavisnosti, pogledi mogu služiti i za zaštitu podataka.
  - Projektant ili administrator mogu određenom korisniku pridružiti njegov pogled na bazu.
  - Korisnik tada „vidi“ samo dio baze, pa su mu time bitno ograničene njegove mogućnosti zlouporabe podataka.
- U relacijskom modelu:
  - globalna shema i pogled (pod-shema) zadaju se kao skup relacija.
  - Pritom se virtualne relacije koje čine pogled izvode iz stvarnih relacija koje čine globalnu shemu.

Pogledi kao mehanizam zaštite (2)

- U SQL-u:
  - relacija-pogled zadaje se naredbom CREATE VIEW,
  - izvođenje pogleda iz globalnih relacija opisuje se naredbom SELECT unutar CREATE VIEW.
- Da bi zaštita preko pogleda zaista funkcionirala, potrebno je još korisniku regulirati ovlaštenja.
  - Naredbe GRANT i REVOKE primjenjive su ne samo na stvarne relacije nego i na poglede.
  - GRANT i REVOKE moraju osigurati da korisnik nema pristupa do stvarnih relacija ali da može koristiti poglede.
  - Korisnik je prisiljen raditi samo s podacima koje smo obuhvatili pogledima, a ostali podaci su mu nedostupni.

Pogledi kao mehanizam zaštite (3)

- Definiranje pogleda za pojedine vrste korisnika:
  - Moglo bi se prepustiti administratoru baze, no bolje je da se time bavi projektant,
  - Naredbe CREATE VIEW projektant može uključiti u svoju fizičku shemu.
- Tri primjera uporabe pogleda kao zaštite.
  - Pogled na podatke o nastavnicima gdje se skriva povjerljivi podatak o plaći.
  - Pogled na podatke o nastavnicima pripremljen za tajnicu Zavoda za računarstvo.
  - Pogled na podatke o upisanim predmetima za studenta s JMBAG-om 0036398757.

Pogledi kao mehanizam zaštite (4)

```
CREATE VIEW NAST_VIEW1
  AS SELECT OIB, PREZIME, IME, IME_ZAVODA, BROJ_SOBE
  FROM NASTAVNIK;

CREATE VIEW NAST_VIEW2
  AS SELECT * FROM NASTAVNIK
  WHERE IME_ZAVODA = 'Zavod za računarstvo';

CREATE VIEW UPISAO_VIEW
  AS SELECT  PREDMET.NASLOV, UPISAO.DATUM_UPISA, UPISAO.OCJENA
  FROM UPISAO, PREDMET
  WHERE UPISAO.JMBAG = 0036398757
  AND UPISAO.SIFRA_PREDMETA = PREDMET.SIFRA_PREDMETA;
```

Sadržaj Poglavlja 7

Općenito o istovremenom pristupu (1)

- Većina baza podataka je *višekorisnička*.
  - Jedan te isti podatak potreban je raznim osobama i raznim aplikacijama, možda čak u isto vrijeme.
- Od suvremenog DBMS-a traži se da korisnicima omogući *istovremeni pristup* do podataka.
  - Svaki korisnik treba imati dojam da sam radi s bazom.
  - Obično je riječ o prividnoj istovremenosti (dijeljenje vremena istog računala).
  - Ipak, DBMS i u tom slučaju mora pažljivo koordinirati radnje korisnika.
  - Ne smije se dopustiti da istovremene radnje, koje mogu biti u konfliktu, naruše integritet baze.

Općenito o istovremenom pristupu (2)

- Opisat ćemo mehanizme i metode za čuvanje integriteta baze u uvjetima istovremenog rada korisnika.
  - Riječ je o mehanizmima koji su ugrađeni u DBMS i aktiviraju se automatski,
  - ili o metodama koje moraju rabiti programeri prilikom pisanja svojih aplikacija.

- Zahtjev da se integritet baze ne smije narušiti zbog istovremenog pristupa preciznije se izražava kao svojstvo *serijalizabilnosti*.

Serijalizabilnost paralelnih transakc (1)

- Rad korisnika s bazom svodi se na pokretanje unaprijed definiranih *transakcija*.
  - Jedna transakcija predstavlja nedjeljivu cjelinu, no realizira se kao niz osnovnih operacija u samoj bazi.
- U višekorisničkoj bazi dešavat će se da se nekoliko transakcija izvodi paralelno.
  - Osnovne operacije koje pripadaju raznim transakcijama tada će se vremenski ispreplesti.
- Da bismo održali integritet baze, zahtijevamo da učinak paralelnih transakcija mora biti isti kao da su se one izvršavale sekvencijalno u nekom (bilo kojem) redoslijedu.

Serijalizabilnost paralelnih transakc (2)

- Svojstvo da učinak istovremenog izvršavanja transakcija mora biti ekvivalentan nekom sekvencijalnom izvršavanju, naziva se *serijalizabilnost*.
- Serijalizabilnost se ne može a priori garantirati.
  - Nekontrolirano paralelno izvršavanje transakcija može dovesti do neželjenih efekata.
- Da bismo to pokazali, zamislimo da:
  - Studenti organiziraju ekskurziju autobusom.
  - Broj studenata koji će ići na ekskurziju ograničen je brojem sjedala u autobusu.
  - Fakultetska baza bilježi koji student je zauzeo koje sjedalo, te koliko ima slobodnih sjedala.
  - Studenti na *on-line* način rezerviraju mjesto u autobusu.

Serijalizabilnost paralelnih transakc (3)

- Tada je moguć sljedeći scenarij uporabe baze.

  Dva studenta istovremeno pokušavaju dobiti kartu za ekskurziju. U tom trenutku postoji samo jedno slobodno sjedalo u autobusu. Pokreću se dvije transakcije, T_1 i T_2, koje DBMS „istovremeno“ obavlja. Dolazi do slijedećeg vremenskog redoslijeda izvršavanja osnovnih operacija.

  1. T_1 učitava iz baze broj slobodnih sjedala.
  2. T_1 smanjuje pročitanu vrijednost s 1 na 0. Promjena je za sada učinjena samo u radnoj memoriji računala.
  3. T_2 učitava iz baze broj slobodnih sjedala. Stanje baze je još uvijek nepromijenjeno, pa je učitani broj neažuran, dakle 1.
  4. T_2 smanjuje pročitanu vrijednost s 1 na 0. Promjena je opet učinjena samo u radnoj memoriji.
  5. T_1 unosi u bazu promijenjenu vrijednost za broj slobodnih sjedala. Dakle u bazi sada piše da nema slobodnih sjedala.

Serijalizabilnost paralelnih transakc (4)

6. Slično i T_2 unosi u bazu svoju promijenjenu vrijednost za broj slobodnih sjedala. Dakle u bazu se ponovo upisuje da nema slobodnih sjedala.
7. Budući da je na početku svog rada naišla na broj slobodnih sjedala veći od 0, T_1 izdaje studentu kartu.
8. Iz istih razloga i T_2 izdaje drugom studentu kartu.

Vidimo da se ovaj način stvorila jedna karta previše. Dva studenta morala bi sjediti na istom sjedalu.

- Ozbiljni DBMS ne smije dopustiti ovakav slijed događaja, već mora garantirati serijalizabilnost.
- Potrebna je kontrola istovremenog izvršavanja transakcija. Uobičajena tehnika koju rabi i MySQL zasniva se na lokotima.

Lokoti i zaključavanja (1)

- *Lokoti* su pomoćni podaci koji služe za koordinaciju konfliktnih radnji.
  - Baza je podijeljena na više dijelova, tako da jednom dijelu odgovara točno jedan lokot.
  - Transakcija koja želi pristupiti nekom podatku najprije mora „uzeti“ lokot i time *zaključati* dotični dio baze.
  - Čim je obavila svoju operaciju, transakcija treba „vratiti“ lokot i time *otključati* podatke.
  - Kad transakcija naiđe na podatke koji su već zaključani, ona mora čekati dok ih prethodna transakcija ne otključa.
  - Time se zapravo izbjegava (sasvim) istovremeni pristup istom podatku.

Lokoti i zaključavanja (2)

- Ovaj mehanizam dovoljan je da otkloni probleme iz prethodnog primjera.
- Veličina dijela baze kojem je pridružen jedan lokot određuje *zrnatost* zaključavanja (*granularity*).
  - Što je zrno krupnije, to je kontrola zaključavanja jednostavnija, no stupanj paralelnosti rada je manji.
  - Zrnatost suvremenih DBMS-a je obično reda veličine n-torke ili bar fizičkog bloka.
- Uporaba lokota krije u opasnost od međusobne *blokade* dviju ili više transakcija (*deadlock*).

Lokoti i zaključavanja (3)

- Da bismo točnije objasnili pojam blokade, opet se služimo zamišljenim scenarijem vezanim uz našu studentsku ekskurziju autobusom.

  Uzmimo da za svako sjedalo u autobusu baza podataka pohranjuje ime putnika (studenta) koji sjedi na tom sjedalu. Pretpostavimo da postoji transakcija kojom dva putnika mogu zamijeniti sjedala (na primjer pušač i nepušač). Zamislimo sada da su istovremeno pokrenute dvije transakcije ovakve vrste, nazovimo ih T_1 i T_2. Neka T_1 mijenja imena putnika na sjedalima 1 i 2, a T_2 obavlja istu zamjenu ali u suprotnom redoslijedu. Tada je moguć slijedeći način obavljanja elementarnih operacija.

Lokoti i zaključavanja (4)

1. T_1 zaključa podatak o sjedalu 1 jer mu misli pristupiti.
2. Iz istih razloga T_2 zaključa podatak o sjedalu 2.
3. T_1 traži lokot za sjedalo 2, ali mora čekati jer je T_2 već zaključala dotični podatak.
4. Slično, T_2 traži lokot za sjedalo 1, ali mora čekati jer je T_1 već zaključala taj podatak.

Očigledno ni T_1 ni T_2 više ne mogu nastaviti rad.

- DBMS koji rabi lokote mora se brinuti da se eventualna blokada spriječi ili prekine. Rješenje:
  - Privremeno se dopušta blokada, no povremeno se kontrolira ima li blokiranih transakcija.
  - Ako takve transakcije postoje, jedna od njih se prekida, neutralizira se njezin dotadašnji učinak te se ona ponovo starta u nekom kasnijem trenutku.

Lokoti i zaključavanja (5)

- Način uporabe lokota u MySQL-u.
  - Kod jednostavnijih tipova datoteka, na primjer MyISAM, rabe se lokoti na razini cijele relacije.
    - MySQL na početku izvršavanja transakcije automatski uzima lokote za sve relacije koje sudjeluju u toj transakciji.
    - Kod svih transakcija lokoti se uzimaju u istom redoslijedu, čime se izbjegava blokada.
  - Kod složenijih tipova datoteki, na primjer InnoDB, rabe se lokoti na razini *n*-torke.
    - Lokoti se opet uzimaju automatski, dakle bez eksplicitne naredbe u transakciji.
    - Blokada je moguća, no ona se automatski otklanja prekidanjem i neutralizacijom jedne transakcije.

Dvofazni protokol zaključavanja (1)

- Na osnovi do sada pokazanih primjera, moglo bi se povjerovati da uporaba lokota (uz izbjegavanje blokade) daje garanciju za serijalizabilnost.
- To nažalost nije točno, a pogrešni utisak stekao se zato što su primjeri bili suviše jednostavni.
  - Moguće je neispravno izvršavanje istovremenih transakcija makar se podaci zaključavaju.
- Da bismo to pokazali, još jednom ćemo se poslužiti zamišljenim „scenarijem“ vezanim uz našu studentsku ekskurziju autobusom.

Dvofazni protokol zaključavanja (2)

Opet promatramo transakciju kojom u autobusu dva putnika (studenta) mijenjaju sjedala. Uzmimo da su istovremeno pokrenute dvije identične transakcije, T_1 i T_2, koje obje mijenjaju imena putnika na istim sjedalima 1 i 2. Neka na početku na tim sjedalima sjede studenti Ivan i Marko. Tada je moguć sljedeći redoslijed obavljanja elementarnih operacija.

1. T_1 zaključa sjedalo 1, čita ime Ivan i pamti ga kao prvo ime, te otključa sjedalo 1.
2. T_1 zaključa sjedalo 2, čita ime Marko i pamti ga kao drugo ime, te otključa sjedalo 2.
3. T_1 ponovo zaključa sjedalo 1, upisuje mu zapamćeno drugo ime (dakle Marko), te otključa sjedalo 1.
4. T_2 zaključa sjedalo 1, čita ime Marko i pamti ga kao svoje prvo ime, te otključa sjedalo 1.

Dvofazni protokol zaključavanja (3)

5. T_2 zaključa sjedalo 2, čita ime Marko i pamti ga kao svoje drugo ime, te otključa sjedalo 2.
6. T_1 ponovo zaključa sjedalo 2, upisuje mu zapamćeno prvo ime (dakle Ivan), te otključa sjedalo 2.
7. T_2 ponovo zaključa sjedalo 1, upisuje mu svoje zapamćeno drugo ime (dakle Marko), te otključa sjedalo 1.
8. T_2 ponovo zaključa sjedalo 2, upisuje mu svoje zapamćeno prvo ime (dakle opet Marko), te otključa sjedalo 2.

Vidimo da sada na oba sjedala sjedi Marko, a Ivana nema nigdje. Opisani način izvršavanja transakcija T_1 i T_2 nije serijalizabilan. Naime, bilo koji sekvencijalni redoslijed izvršavanja proizveo bi dvostruku zamjenu, to jest Ivan bi opet bio na sjedalu 1, a Marko na sjedalu 2.

Dvofazni protokol zaključavanja (4)

- Dakle zaključavanje podataka samo po sebi nije garancija za serijalizabilnost.
- Srećom, stvar se lagano može spasiti. Dovoljno je od transakcija zahtijevati da se pokoravaju jednom strožem „pravilu ponašanja“.
- Može se dokazati da vrijedi tvrdnja.
  - Ako u svakoj od transakcija sva zaključavanja slijede prije prvog otključavanja, tada proizvoljno istovremeno izvršavanje tih transakcija mora biti serijalizabilno.

Pravilo se zove *dvofazni protokol zaključavanja*.

Dvofazni protokol zaključavanja (5)

- Dvofazni protokol zaključavanja objašnjavamo na zadnjem primjeru s dvostrukom zamjenom sjedala.
  - Razlog zašto transakcije T_1 i T_2 nisu korektno radile je baš taj što se one nisu pokoravale protokolu.
  - Da bi bila u skladu sa protokolom, transakcija mora samo jednom zaključati podatak o sjedalu (prije čitanja), te ga samo jednom otključati (nakon ažuriranja).
  - Uz ovakvu promjenu „ponašanja“, prije opisani redoslijed događaja više nije moguć.
  - U najmanju ruku, koraci 5 i 6 morat će zamijeniti redoslijed, jer T_2 neće moći pristupiti sjedalu 2 dok ga T_1 nije ažurirala i otključala.
  - Obje transakcije tada će korektno obaviti svoj posao.

Vremenski žigovi (1)

- Postoje i druge metode za koordinaciju paralelnog izvršavanja transakcija koje ne rabe lokote.
- Kao primjer, spominjemo tehniku zasnovanu na *vremenskim žigovima* (*time stamps*).
  - Svakoj transakciji pridružuje se identifikacijski broj, takozvani vremenski žig.
  - Čitanja i promjene istog podatka dozvoljavaju se samo ako se one odvijaju u redoslijedu vremenskih žigova.
  - U slučaju narušavanja tog redoslijeda, jedna od transakcija mora se prekinuti, neutralizirati i ponovo startati s većim vremenskim žigom.
  - Na primjer, ako transakcija T_1 ima žig t_1, T_2 ima žig $t_2 > t_1$, T_1 želi pročitati podatak x, a T_2 je već mijenjala taj isti x, tada se T_1 mora neutralizirati.

Vremenski žigovi (2)

- Izvršavanje transakcija u skladu s vremenskim žigovima garantira serijalizabilnost. Naime:
  - Ukupni učinak svih transakcija je isti kao da se svaka od njih izvršavala trenutačno, u svom posebnom trenutku koji je određen vremenskim žigom.
- Usporedba dvofaznog protokola zaključavanja i vremenskih žigova:
  - Obje metode garantiraju serijalizabilnost, no obje stvaraju dodatno opterećenje za DBMS.
  - Metoda s vremenskim žigovima je efikasnija ako transakcije najčešće pristupaju različitim podacima i ne smetaju jedna drugoj.
  - Uporaba lokota se više isplati ako postoje zajednički podaci kojima sve transakcije moraju pristupiti.