SQL DDL - Okidači

Evidencija račun

Potrebno je evidentirati račune izdane u trgovini. Za svaki račun potrebno je pratiti *broj_računa*, *datum_izdavanja* i *zaposlenika* koji je račun izdao, te kupca koji je platio račun. Za zaposlenika se prati: *ime*, *prezime*, *oib*, *datum_zaposlenja*, dok se za kupca prati *ime* i *prezime*. Na svakom računu se nalazi stavke koje u sebi sadrže *artikl* i *količinu*. Artikl se sastoji od *naziva* i *cijene*.

```
kupac(id, ime, prezime)
zaposlenik(id, ime, prezime, oib, datum_zaposlenja)
artikl(id, naziv, cijena)
racun(id, id_zaposlenik, id_kupac, broj, datum_izdavanja)
stavka_racun(id, id_racun, id_artikl, kolicina)
```

Baza podataka & tablice

```
CREATE DATABASE trgovina;
USE trgovina;

CREATE TABLE kupac (
  id INTEGER NOT NULL,
  ime VARCHAR(10) NOT NULL,
  prezime VARCHAR(15) NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE zaposlenik (
  id INTEGER NOT NULL,
  ime VARCHAR(10) NOT NULL,
  prezime VARCHAR(15) NOT NULL,
  oib CHAR(11) NOT NULL,
  datum_zaposlenja DATETIME NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE artikl (
  id INTEGER NOT NULL,
  naziv VARCHAR(20) NOT NULL,
  cijena NUMERIC(10,2) NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE racun (
```

```
  id INTEGER NOT NULL,
  id_zaposlenik INTEGER NOT NULL,
  id_kupac INTEGER NOT NULL,
  broj VARCHAR(100) NOT NULL,
  datum_izdavanja DATETIME NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (id_zaposlenik) REFERENCES zaposlenik (id),
  FOREIGN KEY (id_kupac) REFERENCES kupac (id)
);

CREATE TABLE stavka_racun (
  id INTEGER NOT NULL,
  id_racun INTEGER NOT NULL,
  id_artikl INTEGER NOT NULL,
  kolicina INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (id_racun) REFERENCES racun (id) ON DELETE CASCADE,
  FOREIGN KEY (id_artikl) REFERENCES artikl (id),
  UNIQUE (id_racun, id_artikl)
);
```

Unos podataka

```
INSERT INTO kupac VALUES (1, 'Lea', 'Fabris'),
                         (2, 'David', 'Sirotić'),
                         (3, 'Tea', 'Bibić');

INSERT INTO zaposlenik VALUES
  (11, 'Marko', 'Marić', '123451', STR_TO_DATE('01.10.2020.', '%d.%m.%Y.')),
  (12, 'Toni', 'Milovan', '123452', STR_TO_DATE('02.10.2020.', '%d.%m.%Y.')),
  (13, 'Tea', 'Marić', '123453', STR_TO_DATE('02.10.2020.', '%d.%m.%Y.'));

INSERT INTO artikl VALUES (21, 'Puding', 5.99),
                          (22, 'Milka čokolada', 30.00),
                          (23, 'Čips', 9);

INSERT INTO racun VALUES
  (31, 11, 1, '00001', STR_TO_DATE('05.10.2020.', '%d.%m.%Y.')),
  (32, 12, 2, '00002', STR_TO_DATE('06.10.2020.', '%d.%m.%Y.')),
  (33, 12, 1, '00003', STR_TO_DATE('06.10.2020.', '%d.%m.%Y.'));

INSERT INTO stavka_racun VALUES (41, 31, 21, 2),
                                (42, 31, 22, 5),
                                (43, 32, 22, 1),
                                (44, 32, 23, 1);
```

Definiranje okidača:

```
DELIMITER //
CREATE TRIGGER prefix_table
    {BEFORE|AFTER} {INSERT|UPDATE|DELETE} ON table
    FOR EACH ROW
BEGIN
  ...
END//
DELIMITER ;
```

Zadatak: Kod unosa artikla pretvori naziv u velika slova prije spremanja:

```
DELIMITER //
CREATE TRIGGER bi_artikl
  BEFORE INSERT ON artikl
  FOR EACH ROW
BEGIN
  SET new.naziv = UPPER(new.naziv);
END//
DELIMITER ;

-- Testiranje
INSERT INTO artikl VALUES (24, 'Kruh', 10.0);

-- Ispis
SELECT * FROM artikl;

--  Kod ažuriranja artikla ćemo ipak moći neometano postaviti naziv
UPDATE artikl SET naziv = 'Kruh' WHERE id = 24;
```

Zadatak: Prilikom unosa računa postavi *datum izdavanja* na trenutni datum:

```
DELIMITER //
CREATE TRIGGER bi_racun
  BEFORE INSERT ON racun
  FOR EACH ROW
BEGIN
  SET new.datum_izdavanja = NOW();
END//
DELIMITER ;

-- Testiranje
INSERT INTO racun VALUES (34, 12, 1, '00004', NULL);
-- Ispis
SELECT * FROM racun;
```

Zadatak: osiguraj da će *količina* na stavki računa uvijek biti pozitivna, ako je količina manja od 1 onda ju je potrebno postaviti na 1:

```
DELIMITER //
CREATE TRIGGER bi_stavka_racun
  BEFORE INSERT ON stavka_racun
  FOR EACH ROW
BEGIN
  IF new.kolicina < 1 THEN
    SET new.kolicina = 1;
  END IF;
END//
DELIMITER ;

-- Testiranje
INSERT INTO stavka_racun VALUES (45, 33, 23, -10);
-- Ispis
SELECT * FROM stavka_racun;
-- Kod ažurianja nismo uspjeli ispraviti količinu
UPDATE stavka_racun SET kolicina = -20 WHERE id = 45;


# Trebamo definirati i okidač kod ažuriranja
DELIMITER //
CREATE TRIGGER bu_stavka_racun
  BEFORE UPDATE ON stavka_racun
  FOR EACH ROW
BEGIN
  IF new.kolicina < 1 THEN
    SET new.kolicina = 1;
  END IF;
END//
DELIMITER ;

-- Testiranje
UPDATE stavka_racun SET kolicina = -20 WHERE id = 45;
-- Ispis
SELECT * FROM stavka_racun;
```

Zadatak: osiguraj da će *količina* na stavki računa uvijek biti pozitivna, ako je količina manja od 1 onda ju je potrebno postaviti na 1. Ponavljajući kod izdvojiti u proceduru:

```
# Brisanje okidača i retka iz prethodnog zadatka
DROP TRIGGER bi_stavka_racun;
DROP TRIGGER bu_stavka_racun;
DELETE FROM stavka_racun WHERE id = 45;

DELIMITER //
CREATE PROCEDURE ispravi_kolicinu(INOUT kolicina INTEGER)
BEGIN
  IF kolicina < 1 THEN
    SET kolicina = 2;
  END IF;
END //
DELIMITER ;

# Sada proceduru možemo koristiti u okidačima
DELIMITER //
CREATE TRIGGER bi_stavka_racun
  BEFORE INSERT ON stavka_racun
  FOR EACH ROW
BEGIN
  CALL ispravi_kolicinu(new.kolicina);
END//
DELIMITER ;

DELIMITER //
CREATE TRIGGER bu_stavka_racun
  BEFORE UPDATE ON stavka_racun
  FOR EACH ROW
BEGIN
    CALL ispravi_kolicinu(new.kolicina);
END//
DELIMITER ;

-- Testiranje
INSERT INTO stavka_racun VALUES (45, 33, 23, -10);
-- Ispis
SELECT * FROM stavka_racun;
-- Testiranje
UPDATE stavka_racun SET kolicina = -20 WHERE id = 45;
-- Ispis
SELECT * FROM stavka_racun;
```

Stare i nove vrijednosti kod ažuriranja: Osiguraj da se datum računa ne može nikada izmijeniti:

```
DELIMITER //
CREATE TRIGGER bu_racun
  BEFORE UPDATE ON racun
  FOR EACH ROW
BEGIN
  SET new.datum_izdavanja = old.datum_izdavanja;
END//
DELIMITER ;

-- Ispis
SELECT * FROM racun;
-- Testiranje
UPDATE racun SET datum_izdavanja = NOW() WHERE id = 33;
-- Ispis
SELECT * FROM racun;
```

Javljanje greške: Zabrani izmjenu *datuma izdavanja* računa:

```
# Brisanje okidača iz prethodnog zadatka
DROP TRIGGER bu_racun;

-- Novi okidač
DELIMITER //
CREATE TRIGGER bu_racun
  BEFORE UPDATE ON racun
  FOR EACH ROW
BEGIN
  IF old.datum_izdavanja != new.datum_izdavanja THEN
    SIGNAL SQLSTATE '40000'
    SET MESSAGE_TEXT = 'Ne možeš mijenjati datum izdavanja računa!';
  END IF;
END//
DELIMITER ;

-- Ispis
SELECT * FROM racun;
-- Testiranje
UPDATE racun SET datum_izdavanja = NOW() WHERE id = 33;
-- Ispis
SELECT * FROM racun;
```

Problemi?: Osiguraj da cijena novog artikla ne bude veća od najskupljeg artikla:

```
# Brisanje okidača iz prethodnih zadatka
DROP TRIGGER bi_artikl;

# Napomena: ako želimo to osigurati i prilikom izmjene cijene, onda bismo morali
napisati i BEFORE UPDATE okidač
DELIMITER //
CREATE TRIGGER bi_artikl
  BEFORE INSERT ON artikl
  FOR EACH ROW
BEGIN
  DECLARE max_cijena DECIMAL(10, 2);

  SELECT MAX(cijena) INTO max_cijena
    FROM artikl;

  IF new.cijena >= max_cijena THEN
    SIGNAL SQLSTATE '40000'
    SET MESSAGE_TEXT = 'Cijena novog artikla ne može biti veća od najskupljega
artikla!';
  END IF;
END//
DELIMITER ;

-- Ispis
SELECT * FROM artikl;
-- Testiranje
INSERT INTO artikl VALUES (25, 'Jabuke', 100.0);
-- Ispis
SELECT * FROM artikl;
```

> 💡 Čitanje tablice nad kojom se pokreće okidač će u nekim bazama podataka izazvati grešku (mutacija okidača), dok je u MySQL-u to dopušteno.

Ograničenja AFTER okidača:

```
# Brisanje okidača iz prethodnih zadatka
DROP TRIGGER bi_artikl;

# Ovaj okidač se neće moći kreirati jer nije moguće mijenjati vrijednosti nakon
unosa/izmjene
DELIMITER //
CREATE TRIGGER ai_artikl
  AFTER INSERT ON artikl
  FOR EACH ROW
BEGIN
  SET new.cijena = 100.00;
END//
DELIMITER ;

# Međutim i dalje možemo izazvati grešku koja će poništiti unos/izmjenu/brisanje retka
DELIMITER //
CREATE TRIGGER ai_artikl
  AFTER INSERT ON artikl
  FOR EACH ROW
BEGIN
  IF new.cijena > 100.0 THEN
    SIGNAL SQLSTATE '40000'
    SET MESSAGE_TEXT = 'Cijena artikla ne može biti veća od 100.00!';
  END IF;
END//
DELIMITER ;

-- Ispis
SELECT * FROM artikl;
-- Testiranje
INSERT INTO artikl VALUES (26, 'Sok', 200.0);
-- Ispis
SELECT * FROM artikl;
```

Materijalizirani pogledi

Zadatak: Napraviti ćemo tablicu (implementacija materijaliziranog pogleda u MySQL-u) koja će pratiti izdanu količinu pojedinog artikla (*online* način):

```
CREATE TABLE izdana_kolicina_artikla (
  id_artikl INTEGER REFERENCES artikl,
  kolicina INTEGER
);

# Kreirati ćemo proceduru za ažuriranje tablice
DELIMITER //
CREATE PROCEDURE azuriraj_kolicinu_artikla(IN p_id_artikl INTEGER, IN p_kolicina
INTEGER)
BEGIN
  DECLARE art_postoji INTEGER;

  SELECT COUNT(*) INTO art_postoji
    FROM izdana_kolicina_artikla
    WHERE id_artikl = p_id_artikl;

  IF art_postoji = 0 THEN
    INSERT INTO izdana_kolicina_artikla VALUES (p_id_artikl, p_kolicina);
  ELSE
    UPDATE izdana_kolicina_artikla
      SET kolicina = kolicina + p_kolicina
      WHERE id_artikl = p_id_artikl;
  END IF;

END //
DELIMITER ;
```

Unos: Ažuriranje količine nakon unosa stavke računa

```
DELIMITER //
CREATE TRIGGER ai_stavka_racun
  AFTER INSERT ON stavka_racun
  FOR EACH ROW
BEGIN
  CALL azuriraj_kolicinu_artikla(new.id_artikl, new.kolicina);
END//
DELIMITER ;

-- Testiranje
INSERT INTO stavka_racun VALUES (46, 33, 21, 1);
-- Ispis
SELECT * FROM stavka_racun;
SELECT * FROM izdana_kolicina_artikla;
```

Brisanje: Ažuriranje količine nakon brisanja stavke računa

```
DELIMITER //
CREATE TRIGGER ad_stavka_racun
  AFTER DELETE ON stavka_racun
  FOR EACH ROW
BEGIN
  CALL azuriraj_kolicinu_artikla(old.id_artikl, -old.kolicina);
END//
DELIMITER ;

-- Testiranje
DELETE FROM stavka_racun WHERE id = 46;
-- Ispis
SELECT * FROM stavka_racun;
SELECT * FROM izdana_kolicina_artikla;
```

Ažuriranje: Ažuriranje količine nakon ažuriranje stavke računa

```
DELIMITER //
CREATE TRIGGER au_stavka_racun
  AFTER UPDATE ON stavka_racun
  FOR EACH ROW
BEGIN
  CALL azuriraj_kolicinu_artikla(old.id_artikl, -old.kolicina);
  CALL azuriraj_kolicinu_artikla(new.id_artikl, new.kolicina);
END//
DELIMITER ;

-- Priprema
INSERT INTO stavka_racun VALUES (46, 33, 21, 1);
SELECT * FROM stavka_racun;
SELECT * FROM izdana_kolicina_artikla;

-- Testiranje
UPDATE stavka_racun SET kolicina = 10, id_artikl = 22 WHERE id = 46;
-- Ispis
SELECT * FROM stavka_racun;
SELECT * FROM izdana_kolicina_artikla;
```

Alternativni (*offline/batch*) način: Rješiti ćemo isti zadatak za praćenje izdane količine artikala, ali umjesto da količinu u tablici *izdana_kolicina_artikla* ažuriramo odmah pri unosu stavki računa (*online*), napraviti ćemo to na kraju radnog dana (*offline/batch*) i ažurirati tablicu prema svim računima koji su izdani tog dana:

```
# Brisanje objekata iz prošlog primjera
DROP PROCEDURE azuriraj_kolicinu_artikla;
DROP TABLE izdana_kolicina_artikala;
DROP TRIGGER bi_stavka_racun;
DROP TRIGGER bu_stavka_racun;
DROP TRIGGER bd_stavka_racun;

# Kreiranje tablice i procedure (isto kao u prethodnom primjeru)
CREATE TABLE izdana_kolicina_artikala (
  id_artikl INTEGER REFERENCES artikl,
    kolicina INTEGER
);

DELIMITER //
CREATE PROCEDURE azuriraj_kolicinu_artikla(IN p_id_artikl INTEGER, IN p_kolicina
INTEGER)
BEGIN
  DECLARE l_art_postoji INTEGER;

  SELECT COUNT(*) INTO l_art_postoji
    FROM izdana_kolicina_artikala
    WHERE id_artikl = p_id_artikl;

  IF l_art_postoji = 0 THEN
    INSERT INTO izdana_kolicina_artikala VALUES (p_id_artikl, p_kolicina);
  ELSE
    UPDATE izdana_kolicina_artikala
      SET kolicina = kolicina + p_kolicina
      WHERE id_artikl = p_id_artikl;
  END IF;

END//
DELIMITER ;
```

Procedura koja iterira kroz stavke računa izdanih na određeni datum: Procedura ima ulazni parametar *p_datum* za koji datum ćemo obračunati količinu artikala. U kursoru definiramo da dohvaćamo *id_artikl* i *kolicinu* sa stavki računa, iz svih računa koji su izdani na datum *p_datum*. Iteriramo kroz sve stavke i pozivamo prethodnu proceduru za ažuriranje količine zaliha.

```
DELIMITER //
CREATE PROCEDURE azuriraj_dnevnu_kolicinu(IN p_datum DATE)
BEGIN
  DECLARE l_id_artikl, l_kolicina INTEGER;
  DECLARE finished INTEGER DEFAULT 0;

  DECLARE cur CURSOR FOR
    SELECT id_artikl, kolicina
      FROM stavka_racun
      WHERE id_racun IN (
        SELECT id
          FROM racun
          WHERE DATE(datum_izdavanja) = DATE(p_datum)
      );

  DECLARE CONTINUE HANDLER FOR NOT FOUND
  BEGIN
    SET finished = 1;
  END;

  OPEN cur;

  petlja: LOOP

    FETCH cur INTO l_id_artikl, l_kolicina;

    IF finished = 1 THEN
      LEAVE petlja;
    END IF;

    CALL azuriraj_kolicinu_artikla(l_id_artikl, l_kolicina);

  END LOOP petlja;

  CLOSE cur;

END //
DELIMITER ;
```

Poziv procedure: Procedura za obračun količina se može pozvati ručno:

```
-- Ispis
SELECT * FROM izdana_kolicina_artikala;
SELECT * FROM stavka_racun
  WHERE id_racun IN (SELECT id FROM racun WHERE DATE(datum_izdavanja) =
STR_TO_DATE('05.10.2020.', '%d.%m.%Y.'));

-- Obračun izdanih količina artikala za datum 05.10.2020.
CALL azuriraj_dnevnu_kolicinu( STR_TO_DATE('05.10.2020.', '%d.%m.%Y.') );

-- Obračun izdanih količina artikala za trenutni datum
-- Ovaj poziv procedure se može npr. koristiti u aplikaciji gdje korisnik klikne na
gumb "Dnevni obračun"
CALL azuriraj_dnevnu_kolicinu( CURRENT_DATE() );


-- Također, može se definirati "Job" koji će u bazi automatski pokretati proceduru
svaki dan
CREATE EVENT job_dnevni_obracun
ON SCHEDULE EVERY 1 DAY
DO
   CALL azuriraj_dnevnu_kolicinu(  CURRENT_DATE() );
```

Zadaci:

1. Napiši okidač koji će pretvoriti ime kupca u velika slova samo ako su sva slova mala, npr. ime *Teo* neće pretvarati jer je barem jedno slovo veliko, dok će ime *teo* pretvoriti u velika slova (*TEO*) jer su sva slova malena. Dovoljno je napisati samo jedan okidač po želji (kod unosa ili kod izmjene). **Napomena:** ako se želi napraviti usporedba dvaju izraza bez da se ignoriraju velika i mala slova je potrebno koristiti naredbu BINARY
2. Napiši okidač koji će zabraniti brisanje artikla ako je barem jednom dodan na stavke računa (ista funkcionalnost koju i Foreign Key osigurava). Greška će ispisati poruku *'Ne možeš brisati artikl koji je barem jednom izdan na računu!'*.
3. Napiši okidač koji će zabraniti izmjenu *naziva* artikla koji je izdan na računima u sveukupnoj količini većoj od 4. **Npr.** ne možemo izmijeniti naziv artikla *'Milka čokolada'* jer je na računima izdan u sveukupnoj količini od 6, dok je sve ostale atribute artikla moguće mijenjati.