Integritate-Murriztapenak eta Abiarazleak Ariketak

1. Bitez PRODUKTUA(kodPro, izenPro, erosPrezioa, salmentaPrezioa, mota) eta KONPETENTZIAPRODUKTUA(kodPro, izenPro, erosPrezioa, salmentaPrezioa, mota) taulak. Lehenengoak biltegian ditugun produktuak gordetzen ditu: produktuen erosketa-prezioa eta salmentako prezioa gordetzen dira, eta motak produktuaren kalitateari buruzko informazioa ematen du. Konpetentziakoen produktuen taulak ez du erosketa-prezioaren zutaberik, bere ordez hornitzailearen kodea gordetzen da. Taula horien ondorengo integritate- murriztapenak idatz itzazu:

   a) Produktuen erosketa-prezioa 100 baino handiagoa izan behar da.

```
ALTER TABLE PRODUKTUA
  ADD CONSTRAINT 1A CHECK (erosPrezioa > 100)
```

   b) Biltegiko produktuen salmenta-prezioa konpetentziakoen produkturen batena baino txikiagoa izan behar da.

```
ALTER TABLE PRODUKTUA
  ADD CONSTRAINT 1B CHECK (salmentaPrezioa <ANY (SELECT salmentaPrezioa FROM
  KONPETENTZIAPRODUKTUA)

CREATE ASSERTION 1B
   CHECK (NOT EXISTS (SELECT *
              FROM PRODUKTUA
              WHERE salmentaPrezioa >= ALL
                                (SELECT salmentaPrezioa FROM KONPETENTZIAPRODUKTUA)
```

Oharra: Taulabarneko integritate-murriztapena definitzean kontuan hartu behar dugu KONPETENTZIAPRODUKTUA taula aldatuz gero, zehazki, orain arteko produktu garestiena ezabatuz gero, murriztapena ez dela egiaztatuko eta PRODUKTUAko tuplaren batentzat baldintza ez betetzea gerta daitekeela. Horregatik taularteko murriztapena bezala definitu behar dugu.
Adibidez, KONPETENTZIA taulan (....5..) eta (...2...) tuplak eta PRODUKTUA taulan (... 4...) tupla (balioak salmentaPrezioa dira). Egoera hori onargarria da, orain *delete from KONPETENTZIA KONPETENTZIA where ...* egiten bada, eta (... 5...) tupla ezabatzen bada, geratzen den egoera ez dago ondo, eta tupla barneko murriztapenak ez du detektatuko.

   c) Biltegiko produktuen salmenta-prezioa erosketa-prezioa baino %30 handiagoa izan behar da.

```
ALTER TABLE PRODUKTUA
  ADD CONSTRAINT 1C CHECK (salmentaPrezioa > 1.30*erosPrezioa)
```

   d) 'A' motako produktuek gutxienez 200ko salmenta prezioa izan behar dute.

$$\forall x\,(\text{produktua}(x) \wedge x.\text{mota}='A' \rightarrow x.\text{salmentaPrezioa} > 200) \;\Rightarrow\; \forall x\,(\text{produktua}(x) \wedge x.\text{mota}<>A \vee x.\text{salmentaPrezioa} > 200)$$

```
ALTER TABLE PRODUKTUA
  ADD CONSTRAINT 1D CHECK (mota <> 'A' OR salmentaPrezioa > 200)
```

e) Bere erosketa-prezioa 10.000 baino handiagoa den produkturik ezin da eduki.

```
ALTER TABLE PRODUKTUA
  ADD CONSTRAINT 1E CHECK (erosPrezioa < 10.000)
```

f) Salmenta-prezioa konpetentziakoek eskaintzen duten minimoa baino txikiagoa duten hiru produktu baino gehiago ezin dira egon.

```
CREATE ASSERTION 1F
   CHECK (3 >= SELECT COUNT(*)
         FROM PRODUKTUA
         WHERE salmentaPrezioa < (SELECT MIN(salmentaPrezioa) FROM KONPETENTZIAPRODUKTUA))
```

g) Mota, hauetako hizki bat da: A, B, C, X, Z.

```
ALTER TABLE PRODUKTUA
  ADD CONSTRAINT 1H CHECK (mota IN ('A', 'B', 'C', 'X', 'Z')
```

h) Gure biltegian dauden produktuen kalitatea konpetentziakoena baino hobea da. Kalitatearen ordena honakoa da A>B>C>X>Z.

```
ALTER TABLE PRODUKTUA
  ADD CONSTRAINT 1I CHECK (mota > (SELECT mota
                                   FROM KONPETENTZIAPRODUKTUA
                                          WHERE kodPro = PRODUKTUA.kodPro)
```

i) A motako produktuen erosketa-prezioa eta salmenta-prezioaren arteko diferentzia 200koa da, gure alde.

$\forall x$ (produktua(x) $\wedge$ x.mota=A $\rightarrow$ x.sp-x.ep=200) $\Rightarrow$ $\forall x$ (produktua(x) $\wedge$ x.mota<>A $\vee$ x.sp-x.ep=200)

```
ALTER TABLE PRODUKTUA
  ADD CONSTRAINT 1J CHECK (mota<>'A' OR salmentaPrezioa-erosPrezioa=200)
```

2. Bedi honako eskema erlazionala. Tauletan pelikulen eta horiek eman diren zine-salen inguruko datuak gordetzen dira (estreinaldiak edo erreposizioak). Sala bakoitza bere zenbakia eta kokatzen den zinearen izenaren bidez identifikatzen da (adibidez, Astoria zineko 3. sala). Pelikula bakoitza sala berean denboraldi batean baino gehiagotan ematea posible izan daitekeenez, *PROIEKTATUA* taulan pelikula ikusgai jarri zeneko data gordetzen da, eta atributu horrek taularen gakoan parte hartzen du. SQLren definizio-lengoaia erabiliz, ondorengo integritate- murriztapenak idatz itzazu.

   PELIKULA (titulua, zuzendaria, produktorea)
   PROIEKZIOSALA (salaZenb, zineIzena, edukiera, airegirotua?, estereo?)
   PROIEKTATUA (*titulua*, salaZenb, zineIzena, proiekzioData, ikusleKop)

a) Salaren zenbakiak 1 eta 10en arteko zenbaki oso bat izan behar du

```
CREATE TABLE PROIEKZIOSALA
  salaZenb NUMBER,
  CONSTRAINT  zenbmurriz CHECK (salaZenb BETWEEN 1 AND 10)
```

b) 100 ikusle baino gehiagorako edukiera duten sala guztiek aire girotua eduki behar dute.

$\forall x \in$ proiekziosala (x.edukiera > 100 $\rightarrow$ x.airegirotua=bai) $\Rightarrow$
$\forall x \in$ proiekziosala (x.edukiera <= 100 $\vee$ x.airegirotua=bai)

```
ALTER TABLE PROIEKZIOSALA
  ADD CONSTRAINT  airegirotumurriz CHECK (edukiera <= 100 OR airegirotua? = TRUE)
```

c) Pelikula baten ikusle-kopurua ezin da pelikula hori proiektatzen den salaren edukiera baino handiagoa izan.

$\forall x \in$ proiektatua ($\exists y \in$ sala (x.salaZenb=y.salaZenb $\wedge$ x.ikusleKop < y.edukiera)) << Taulabarneko adierazpena idazteko
$\Rightarrow$ $\neg\exists x \in$ proiektatua $\neg$ ($\exists y \in$ sala (x.salaZenb=y.salaZenb $\wedge$ x.ikusleKop < y.edukiera))
$\Rightarrow$ $\neg\exists x \in$ proiektatua ($\neg\exists y \in$ sala (x.salaZenb=y.salaZenb $\wedge$ x.ikusleKop < y.edukiera)) << Taularteko adierazpena idazteko

Taulabarneko murriztapena:

```
CREATE TABLE Proiektatua
   ADD CONSTRAINT 1C
   CHECK (ikusleKop < (SELECT edukiera
                       FROM PROIEKZIOSALA
                       WHERE zineIzena = Proiektatua.zineIzena AND salaZenb=Proiektatua.salaZenb))
```

Taularteko murriztapena:

```
CREATE ASSERTION salaEdukieraMurriz
   CHECK (NOT EXISTS (SELECT *
                      FROM PROIEKTATUA AS A
                      WHERE NOT EXISTS (SELECT *
                                        FROM PROIEKZIOSALA AS B
                                        WHERE B.zineIzena = A.zineIzena AND B.salaZenb = A.salaZenb
                                              AND A.ikusleKop < B.edukiera)))

CREATE ASSERTION salaEdukieraMurriz
   CHECK (NOT EXISTS (SELECT *
                      FROM PROIEKTATUA NATURAL JOIN PROIEKZIOSALA WHERE ikusleKop > edukiera)

CREATE ASSERTION salaEdukieraMurriz
   CHECK (NOT EXISTS (SELECT *
                      FROM PROIEKTATUA AS A
                      WHERE ikusleKop > (SELECT edukiera
                                         FROM PROIEKZIOSALA
                                         WHERE zineIzena = A.zineIzena AND salaZenb = A.salaZenb))
```

3. Bedi pelikulei buruzko datu-base sistema. Datu-basean pelikulak, hauek protagonizatu zituzten aktoreak eta ekoiztu zituzten exekutiboak gordetzen dira. Estudioei eta hauen zuzendariei buruzko datuak ere gordetzen dira. Ekoizleak eta Zuzendariak pelikula-exekutiboak dira.

PELIKULA(titulua, urtea, iraupena, koloretan, *estudioa*, *ekoizleKodea*)
PELIKULAPROTAGONISTA(*pelTitulua*, *pelUrtea*, *aktoreIzena*)
AKTOREA(izena, helbidea, sexua, jaiotzeData)
EXEKUTIBOA(kodea, izena, helbidea, ondarea)
ESTUDIOA(izena, helbidea, *zuzendariKodea*)

Ondoren adierazten diren integritate-murriztapenak idatz itzazu. Kasuren batean, baldintzan bi taula badaude, bi murriztapen definitu behar izatea gerta daiteke.

a) Inork ezin du estudio baten zuzendaria izatera iritsi 10.000.000 euroko ondarerik ez badu.

$\forall x \in$ estudioa ($\exists y \in$ exekutiboa (y.kodea=x.zuzendaria $\wedge$ y.ondarea>10.000.000)) << Taulabarneko adierazpena idazteko
$\Rightarrow \neg\exists x \in$ estudioa $\neg$ ($\exists y \in$ exekutiboa (y.kodea=x.zuzendaria $\wedge$ y.ondarea>10.000.000)) << Taularteko adierazpena idazteko

Taulabarneko murriztapena:

```
ALTER TABLE ESTUDIOA ( ...
  ADD CONSTRAINT  im_a1
        CHECK (zuzendariKodea IN (SELECT kodea FROM EXEKUTIBOA WHERE ondarea >= 10.000.000)))

ALTER TABLE EXEKUTIBOA ( ...
  ADD CONSTRAINT  im_a2
        CHECK (kodea NOT IN (SELECT zuzendariKodea FROM ESTUDIOA) OR ondarea >= 10.000.000))
```

Taularteko murriztapena:

```
CREATE ASSERTION im_a
   CHECK (NOT EXISTS (SELECT *
                      FROM ESTUDIOA
                      WHERE zuzendariKodea NOT IN (SELECT kodea
                                                   FROM EXEKUTIBOA
                                                   WHERE ondarea >= 10.000.000)))
```

Oharra: Taulabarneko integritate-murriztapena definitzerakoan kontuan hartu behar dugu EXEKUTIBOA taula aldatuz gero, zehazki, zuzendariren baten ondarea jaitsiz gero, murriztapena ez dela egiaztatuko eta tuplaren batentzat baldintza ez betetzea gerta daitekeela. Horregatik EXEKUTIBOA taulan ere beste taulabarneko murriztapena definitu behar dugu.

Beste era bat:

$\forall x \in$ estudioa,$y \in$ exekutiboa (y.kodea=x.zuzendariKodea $\rightarrow$ y.ondarea>10.000.000)
$\Rightarrow \neg\exists x \in$ estudioa, $y \in$ exekutiboa $\neg$ (y.kodea=x.zuzendariKodea $\rightarrow$ y.ondarea>10.000.000))
$\Rightarrow \neg\exists x \in$ estudioa, $y \in$ exekutiboa (y.kodea=x.zuzendariKodea $\wedge$ y.ondarea<=10.000.000)) << Taularteko adierazpena idazteko

```
CREATE ASSERTION im_a
   CHECK (NOT EXISTS (SELECT *
                      FROM ESTUDIOA INNER JOIN EXEKUTIBOA ON zuzendariKodea = kodea
                      WHERE ondarea <= 10.000.000))
```

Oharra: EXEKUTIBOA taulan ezabaketaren bat badago, ESTUDIOA taulan, dagokion tuplan, 'null' jarriko da. Murriztapenaren egiaztapena egiterakoan, INNER JOIN egiten denean multzo hutsa aterako da eta not exists(∅) = TRUE dela badakigu, beraz murriztapena beteko da.

b) Pelikula bat ezin da koloretakoa izan 1939 aurrekoa bada.

$\forall x \in$ pelikula (x.urtea<1939 $\rightarrow$ x.koloretan=false) $\Rightarrow$ $\forall x \in$ pelikula ($\neg$(x.urtea<1939) $\vee$ x.koloretan=false)

```
ALTER TABLE PELIKULA
  ADD CONSTRAINT  im_b
    CHECK (urtea >= 1939 OR koloretan = false))
```

Oharra: Honako baldintzak ez dira zuzenak:

urtea < 1939 **AND** koloretan = false; pelikula guztiak 1939 aurrekoak eta zuri-beltzean izatera behartzen dugulako.

urtea < 1939 **OR** koloretan = false; pelikulak 1939 aurrekoak izan arren koloretan nahiz zuri-beltzean izan daitezkeelako eta bestela, 1939 ondorengokoa izanik, pelikula derrigorrez zuri-beltzean izatera behartzen dugulako.

c) Aktore bat ezin da bera jaio aurretik filmatutako pelikula batean azaldu.

$\forall x \in$ aktorea ($\forall y \in$ pelikulaprotagonista (y.aktoreizena=x.izena $\rightarrow$ y.pelurtea>x.jaiotzeData))

<<<< Taulabarneko adierazpena idazteko

$\Rightarrow \forall x \in$ aktorea, $y \in$ pelikulaprotagonista (y.aktoreizena=x.izena $\rightarrow$ y.pelurtea>x.jaiotzeData))

$\Rightarrow \neg \exists x \in$ aktorea, $y \in$ pelikulaprotagonista $\neg$ (y.aktoreizena=x.izena $\rightarrow$ y.pelurtea>x.jaiotzeData)

$\Rightarrow \neg \exists x \in$ aktorea, $y \in$ pelikulaprotagonista (y.aktoreizena=x.izena $\wedge \neg$ y.pelurtea>x.jaiotzeData)

<<<< Taularteko adierazpena idazteko

Taulabarneko murriztapena:

```
ALTER TABLE AKTOREA
  ADD CONSTRAINT  im_c
         CHECK (jaiotzeData < ALL (SELECT pelUrtea
                                   FROM PELIKULAPROTAGONISTA
                                   WHERE aktoreIzena = AKTOREA.izena))
ALTER TABLE PELIKULAPROTAGONISTA
  ADD CONSTRAINT  im_c
        CHECK (pelUrtea > (SELECT jaiotzeData
                           FROM AKTOREA
                           WHERE izena = PELIKULAPROTAGONISTA.izena))
```

Taularteko murriztapena:

```
CREATE ASSERTION im_c
   CHECK (NOT EXISTS (SELECT *
                      FROM AKTOREA INNER JOIN PELIKULAPROTAGONISTA ON aktoreIzena = izena
                      WHERE jaiotzeData <= pelUrtea))
```

Beste era bat:

```
CREATE ASSERTION im_c
   CHECK (NOT EXISTS (SELECT *
                      FROM AKTOREA AS A
                      WHERE jaiotzeData >ANY (SELECT pelUrtea FROM PELIKULAPROTAGONISTA
                                              WHERE aktoreIzena = A.izena))
```

Oharra: AKTOREAren taulabarneko integritate-murriztapena definitzean kontuan hartu behar dugu PELIKULAPROTAGONISTA taula aldatuz gero, zehazki, aktorearen pelikula berri bat sartuz gero edo aurrekoren bati data aldatuz gero, murriztapena ez dela egiaztatuko eta aktore tuplaren batentzat baldintza ez betetzea gerta daitekeela. Horregatik PELIKULAPROTAGONISTA taulan ere taulabarneko murriztapena definitzen dugu, murriztapen osagarria, taula hori aldatzen denean egiaztatuko dena.

Hortaz, zuzenena taularteko murriztapena definitzea izango litzateke

d) Bi estudiok ezin dute zuzendari bera eduki.

```
CREATE TABLE ESTUDIOA ( ... zuzendariKodea NUMBER UNIQUE)
```

e) Aktoreen artean azaltzen den izen bat ezin da exekutiboen taulan ere egon.

$\forall x \in$ aktorea (x.izena $\notin$ exekutiboa)

Taulabarneko murriztapena:

```
ALTER TABLE AKTOREA ( …
       ADD CONSTRAINT  im_e1
            CHECK (izena NOT IN (SELECT izena FROM EXEKUTIBOA)));

ALTER TABLE EXEKUTIBOA ( ...
     ADD CONSTRAINT
          im_e2
     CHECK (izena NOT IN (SELECT izena FROM AKTOREA)))
```

Taularteko murriztapena:

```
CREATE ASSERTION im_e
   CHECK (NOT EXISTS (SELECT * FROM AKTOREA WHERE izena IN (SELECT izena FROM EXEKUTIBOA)))
```

Oharra: Taulabarneko integritate-murriztapenak definituz gero bi tauletan egin behar dugu, AKTOREA edota EXEKUTIBOA tauletan sarrera edo eguneraketa burutuz gero egiazta daitezen. Esaterako, murriztapena AKTOREAn bakarrik jarriz gero, exekutiboen taulan aktore baten izena sartzean ez litzateke egiaztatuko. Beste aukera taularteko murriztapena definitzea izango litzateke.

f) *Estudioa* taulan azaltzen den estudio baten izena, *Pelikula* taulako tuplaren batean azaldu behar da.

```
ALTER TABLE ESTUDIOA
  ADD CONSTRAINT  im_f
         CHECK (izena IN (SELECT estudioa FROM PELIKULA))
```

Oharra: Enuntziatuan aipatzen denaz aparte, PELIKULA taulako *estudioa* atributuak ESTUDIOAko *izena* atributua erreferentziatzen duela suposa dezakegu, hots, ESTUDIOA taulakoa ez den baliorik ezin duela eduki. Horrela, honako murriztapena ere egon beharko da:

```
CREATE TABLE PELIKULA ( ...
 CONSTRAINT  im_f1  FOREIGN KEY  estudioa  REFERENCES  ESTUDIOA(izena))
```

Hala ere, aurreko komentario horretaz aparte, eta im_f murriztapenarekin jarraituz, PELIKULA taulan ezabaketaren bat eginez gero estudio baten izena PELIKULA taulatik desagertzea gerta daiteke, taulabarneko murriztapenarekin egiaztatuko ez den egoera sortuz. DELETE eragiketa ez dagoela suposatzen badugu, im_f murriztapena horrela utz dezakegu, bestela hobe izango litzateke taularteko murriztapena definitzea (eragiketa mota guztiekin egiaztatzen dena).

```
CREATE ASSERTION im_f
   CHECK (NOT EXISTS (SELECT *
                      FROM ESTUDIOA
                      WHERE izena NOT IN (SELECT estudioa FROM PELIKULA)))
```

g) Pelikula baten ekoizlea estudio baten zuzendaria ere izanez gero, orduan pelikula ekoiztu zuen estudioaren zuzendaria izan behar da, hain zuzen ere.

$\forall x \in$ pelikula ($\exists$y (estudioa(y) $\wedge$ x.ekoizleKodea=y.zuzendariKodea) $\rightarrow \exists$z (estudioa(z) $\wedge$ x.ekoizleKodea=z.zuzendariKodea $\wedge$ x.estudioa=z.izena))
$\Rightarrow \forall x \in$ pelikula ($\neg \exists$y (estudioa(y) $\wedge$ x.ekoizleKodea=y.zuzendariKodea) $\vee \exists$z (estudioa(z) $\wedge$ x.ekoizleKodea=z.zuzendariKodea $\wedge$ x.estudioa=z.izena))
$\Rightarrow \neg \exists x \in$ pelikula $\neg$ ($\neg \exists$y (estudioa(y) $\wedge$ x.ekoizleKodea=y.zuzendariKodea) $\vee \exists$z (estudioa(z) $\wedge$ x.ekoizleKodea=z.zuzendariKodea $\wedge$ x.estudioa=z.izena))
$\Rightarrow \neg \exists x \in$ pelikula ($\exists$y (estudioa(y) $\wedge$ x.ekoizleKodea=y.zuzendariKodea) $\wedge \neg \exists$z (estudioa(z) $\wedge$ x.ekoizleKodea=z.zuzendariKodea $\wedge$ x.estudioa=z.izena))

```
CREATE ASSERTION im_g
   CHECK (NOT EXISTS (SELECT *
                      FROM PELIKULA AS A
                      WHERE ekoizleKodea IN (SELECT zuzendariKodea FROM ESTUDIOA)
                        AND NOT EXISTS(SELECT *
                                       FROM ESTUDIOA
                                       WHERE A.ekoizleKodea=zuzendariKodea AND izena=A.estudioa)))
```

Beste era bat:

```
CREATE ASSERTION im_g
   CHECK (NOT EXISTS (SELECT *
                      FROM PELIKULA AS A
                      WHERE ekoizleKodea IN (SELECT zuzendariKodea FROM ESTUDIOA)
                                    AND ekoizleKodea <> (SELECT zuzendariKodea
                                                         FROM ESTUDIOA
                                                         WHERE izena = A.estudioa)))
```

Oharra: Batetik, murriztapena PELIKULA eta ESTUDIOA taulen arteko erlazioari dagokio, eta bestetik, ESTUDIOA taulan aldaketaren bat egiten bada PELIKULAren baten ekoizlearengan eragina izan dezake eta hori egiaztatzea interesatzen zaigu (taulabarneko murriztapenak ez digu ziurtatzen). Arrazoi horiek direla eta, taularteko murriztapena definitu dugu.

Beste era bat: Honek ez du funtzionatzen

```
CREATE ASSERTION im_g
   CHECK(NOT EXISTS (SELECT *
                     FROM PELIKULA AS A INNER JOIN ESTUDIOA AS B ON A.ekoizleKodea=B.zuzendariKodea
                     WHERE A.estudioa <> B.izena))
```

Oharra: Honek ez du funtzionatzen, bere adierazpen logikoa azpian jartzen duguna da (eta ikusten den bezala, ez dator bat enuntziatuan adierazitakoarekin). Beste era batera esanda, demagun PELIKULA taulan (p1, ..., e1, ek1) tupla dugula, eta ESTUDIOA taulan bi tupla hauek: (e1...ek1) eta (2 ...ek1). Beraz ek1 ekoizlea bi estudioren zuzendaria da, e1 eta e2, eta p1 pelikularen estudioa eta ekoizlea, hurrenez hurren e1 eta ek1 dira. Enuntziatuan jarritako baldintza betetzen da, beraz, baina check-ean jarritako adierazpenarekin (not exists ... A.estudioa <> B.izena) ez da onartzen. Adierazpen horrekin Pelikularen estudioko zuzendaria izatea besterik ez da onartzen, eta hori ez da enuntziatuan esaten dena.

$\forall x \in$ pelikula ($\forall y \in$ estudioa (x.ekoizleKodea=y.zuzendariKodea $\rightarrow$ x.estudioa = y.izena))
$\Rightarrow$ ...
$\Rightarrow \neg \exists x \in$ pelikula, $y \in$ estudioa (x.ekoizleKodea=y.zuzendariKodea $\wedge \neg$ x.estudioa = y.izena)

h) Pelikularen oinarrizko gakoa titulua eta urtea.

```
ALTER TABLE PELIKULA
  ADD CONSTRAINT im_h PRIMARY KEY  (titulua, urtea)
```

i) Pelikula guztien ekoizleak *Exekutiboa* taulan azaldu behar dira.

```
ALTER TABLE PELIKULA
  ADD CONSTRAINT im_i FOREIGN KEY (ekoizleKodea) REFERENCES EXEKUTIBOA(kodea)
```

j) Pelikulek ezin dute 60 minutu baino gutxiago eta 250 minutu baino gehiago iraun.

```
ALTER TABLE PELIKULA
  ADD CONSTRAINT im_j CHECK (iraupena > 60  AND  iraupena < 250)
```

k) Ezin da izenik egon aktore bezala eta pelikula-exekutibo bezala azaltzen denik.

```
ALTER TABLE AKTOREA
  ADD CONSTRAINT im_k1
        CHECK (izena NOT IN (SELECT izena FROM EXEKUTIBOA));


ALTER TABLE EXEKUTIBOA
  ADD CONSTRAINT im_k2
        CHECK (izena NOT IN (SELECT izena FROM AKTOREA))
```

l) Bi estudiok ezin dute helbide bera eduki.

```
ALTER TABLE ESTUDIOA
  ADD CONSTRAINT im_l (helbidea UNIQUE)
```

Ondorengoko egiaztapen bakoitza egiteko **abiarazle (trigger)** bat eta **taularteko murriztapen** bat (edo gehiago) idatz itzazu. DB aldatzen saiatu baino lehen baldintza betetzen dela suposa dezakezu. Abiarazlearen kasuan, eskatutako aldaketa bertan behera utzi ordez, ahal bada, aldatu egin ezazu (horretarako null balioak (edo lehenetsitako balioak) dituzten tuplak sartu behar baldin badituzu ere). Abiarazlearen gorputzeko kodearen sintaxia algoritmo moduan utz daiteke, eta kasu guzti-guztiak aztertzea ez da beharrezkoa, helburua abiarazleen ezaugarrien (altxarazte unea, baldintza egokia, ekintza-moduak) inguruan hausnartzea baita, eta ez inplementazio erabat zehatza ematea.

m) *PelikulaProtagonista* taulan azaltzen diren aktore guztiak *Aktore* taulan ere azaltzen direla egiaztatu.

```
ALTER TABLE PelikulaProtagonista
  ADD CONSTRAINT im_m (FOREIGN KEY aktoreIzena REFERENCES AKTOREA(izena))

================
CREATE TRIGGER t_m1
BEFORE INSERT OR UPDATE OF(aktoreIzena) ON PELIKULAPROTAGONISTA
FOR EACH ROW
WHEN (NEW.aktoreIzena NOT IN (SELECT izena FROM AKTOREA))
  INSERT INTO AKTOREA VALUES (:NEW.aktoreIzena, null, null, null);
```

n) Exekutiboak, estudio-zuzendari bezala edo pelikula-ekoizle bezala, edo biak bezala agertu behar direla egiaztatu.

```
CREATE ASSERTION im_n
   CHECK (NOT EXISTS (SELECT *
                      FROM EXEKUTIBOA
                      WHERE kodea NOT IN (SELECT zuzendariKodea FROM ESTUDIOA)
                                  AND kodea NOT IN (SELECT ekoizleKodea FROM PELIKULA)))
```

Abiarazleak:

```
CREATE PROCEDURE  Esleipena (kodeBerria NUMBER)
BEGIN    /* aldagaiak erazagututa daudela suposatzen dugu */
     WRITE("Exekutiboa estudioko zuzendaria edo pelikulako ekoizlea izan behar da");
     WRITE("Pelikula edo Estudioa?"); READ(erantzuna);
     IF (erantzuna == "pelikula") THEN
          WRITE("Titulua: "); READ(titu);
          WRITE("Urtea: "); READ(urt);
          UPDATE PELIKULA SET ekoizleKodea = kodeBerria WHERE titulua = :titu AND urtea = :urt;
     ELSE
          WRITE("Izena: "); READ(izen);
          UPDATE ESTUDIOA SET zuzendariKodea = kodeBerria WHERE izena = :izen;
END;


CREATE TRIGGER t_n1
AFTER INSERT ON EXEKUTIBOA
FOR EACH ROW
WHEN (NEW.kodea NOT IN (SELECT zuzendariKodea FROM ESTUDIOA)
      AND NEW.kodea NOT IN (SELECT ekoizleKodea FROM PELIKULA))
BEGIN
     CALL Esleipena (:NEW.kodea);
END;
```

```
CREATE TRIGGER t_n2
AFTER DELETE OR UPDATE OF (ekoizleKodea) ON PELIKULA
FOR EACH ROW
WHEN (NOT EXISTS (SELECT *                  // Ezabatu/aldatutako pelikularen ekoizleak ez du
                  FROM PELIKULA             // besterik zuzentzen edota ekoizten
                  WHERE titulua <> OLD.titulua AND urtea <> OLD.urtea
                    AND ekoizleKodea = OLD.ekoizleKodea)
     AND NOT EXISTS (SELECT * FROM ESTUDIOA WHERE zuzendariKodea = OLD.ekoizleKodea))
BEGIN
    WRITE("Ezabatu edo aldatutako pelikularen ekoizlea besten bati,");
    WRITE (" edo estudioren bati esleitu beharko diozu);
    CALL Esleipena (:OLD.ekoizleKodea);
END;

CREATE TRIGGER t_n3
AFTER DELETE OR UPDATE OF (zuzendariKodea) ON ESTUDIOA
FOR EACH ROW
WHEN (NOT EXISTS (SELECT *
                  FROM ESTUDIOA
                  WHERE izena <> OLD.izena AND zuzendariKodea = OLD.zuzendariKodea))
     AND NOT EXISTS (SELECT * FROM PELIKULA WHERE ekoizleKodea = OLD.zuzendariKodea)

BEGIN
    WRITE("Ezabatu edo aldatutako estudioaren zuzendaria besten bati, ");
    WRITE("edo pelikularen bati esleitu beharko diozu");
    CALL Esleipena (:OLD.zuzendariKodea);
END;
```

o) Pelikula guztiek beren aktoreen artean gutxienez gizon bat eta emakume bat dituztela egiaztatu.

```
CREATE ASSERTION im_o
   CHECK (NOT EXISTS (SELECT *
                      FROM PELIKULA AS
                             A
                      WHERE NOT EXISTS
                                (SELECT *
                                 FROM AKTOREA INNER JOIN PELIKULAPROTAGONISTA ON izena =
                                 aktoreIzena
                                 WHERE   sexua = "gizona"
                                   AND   pelTitulua = A.titulua AND pelUrtea = A.urtea)
                           OR NOT
                           EXISTS
                                (SELECT *
                                 FROM AKTOREA INNER JOIN PELIKULAPROTAGONISTA ON izena =
                                 aktoreIzena
                                 WHERE   sexua = "emakumea"
                                   AND   pelTitulua = A.titulua AND pelUrtea = A.urtea)))
```

Abiarazleak:

```
CREATE TRIGGER t_o1
AFTER INSERT ON PELIKULA
FOR EACH ROW
WHEN NOT EXISTS (SELECT *
                 FROM AKTOREA INNER JOIN PELIKULAPROTAGONISTA ON izena = aktoreIzena
                 WHERE sexua = "gizona" AND pelTitulua = NEW.titulua AND pelUrtea = NEW.urtea)
BEGIN
    WRITE("Pelikulari gizonezkoa falta zaio: "); READ(izena); // datua ondo dagoela suposatuta
    INSERT INTO PELIKULAPROTAGONISTA VALUES(:NEW.titulua, :NEW.urtea, :izena);
END;
```

```
CREATE TRIGGER t_o2
AFTER INSERT ON PELIKULA
FOR EACH ROW
WHEN NOT EXISTS (SELECT *
                 FROM AKTOREA INNER JOIN PELIKULAPROTAGONISTA ON izena = aktoreIzena
                 WHERE sexua = "emakumea" AND pelTitulua = NEW.titulua AND pelUrtea = NEW.urtea)
BEGIN
    WRITE("Pelikulari emakumea falta zaio: "); READ(izena); // datua ondo dagoela suposatuta
    INSERT INTO PELIKULAPROTAGONISTA VALUES(:NEW.titulua, :NEW.urtea, :izena);
END;

CREATE TRIGGER t_o3
AFTER DELETE ON PELIKULAPROTAGONISTA
FOR EACH ROW
WHEN NOT EXISTS (SELECT *     // pelikulak ez dauka sexu bereko beste protagonistarik
                 FROM (AKTOREA AS A INNER JOIN PELIKULAPROTAGONISTA AS B ON izena = aktoreIzena)
                         INNER JOIN  AKTOREA AS C   ON A.sexua = C.sexua
                 WHERE B.pelTitulua = OLD.pelTitulua AND B.pelUrtea = OLD.pelUrtea
                   AND A.izena <> OLD.aktoreIzena AND C.izena = OLD.aktoreIzena)
DECLARE
    x,y STRING;
    salbuespena  EXCEPTION;
BEGIN
    WRITE("Ezabatutakoaren sexu bereko aktorea sartu beharko duzu: "); READ(izena);
    SELECT sexua INTO :x FROM AKTOREA WHERE izena = :izena;
    SELECT sexua INTO :y FROM AKTOREA WHERE izena = :OLD.aktoreIzena;
    IF x=y THEN  INSERT INTO PELIKULAPROTAGONISTA VALUES(:OLD.pelTitulua, :OLD.pelUrtea, :izena);
    ELSE RAISE salbuespena;
END;

CREATE TRIGGER t_o4
AFTER UPDATE OF (aktoreIzena) ON PELIKULAPROTAGONISTA
FOR EACH ROW
WHEN NOT EXISTS (SELECT *
                 FROM (AKTOREA AS A INNER JOIN PELIKULAPROTAGONISTA AS B ON A.izena= .aktoreIzena)
                 WHERE B.pelTitulua = OLD.pelTitulua AND B.pelUrtea = OLD.pelUrtea
                   AND A.sexua = "gizona")
BEGIN
    WRITE("Orain aldatutako pelikulari gizonezko aktorea falta zaio: "); READ(izena);
    INSERT INTO PELIKULAPROTAGONISTA VALUES(:OLD.pelTitulua, :OLD.pelUrtea, :izena);
END;

CREATE TRIGGER t_o5
AFTER UPDATE OF (aktoreIzena) ON PELIKULAPROTAGONISTA
FOR EACH ROW
WHEN NOT EXISTS (SELECT *
                 FROM (AKTOREA AS A INNER JOIN PELIKULAPROTAGONISTA AS B ON A.izena=B.aktoreIzena)
                 WHERE B.pelTitulua = OLD.pelTitulua AND B.pelUrtea = OLD.pelUrtea
                   AND A.sexua = "emakumezkoa")

BEGIN
    WRITE("Orain aldatutako pelikulari emakumezko aktorea falta zaio: "); READ(izena);
    INSERT INTO PELIKULAPROTAGONISTA VALUES(:OLD.pelTitulua, :OLD.pelUrtea, :izena);
END;
```

Oharra: t_o3 abiarazlearen kasuan salbuespen bat altxarazten dugu, horrelakoak ere egin daitezkeela erakusteko, beste kasuren batean ere erabiltzea posible izango da.

p) Estudio batek edozein urtetan 100 pelikula baino gehiago filmatzen ez dituela egiaztatu.

```
CREATE ASSERTION im_p
CHECK (NOT EXISTS (SELECT *
                   FROM PELIKULA
                   GROUP BY estudioa, urtea
                   HAVING COUNT(*) > 100))
```

Abiarazlea:

```
CREATE TRIGGER t_p1
BEFORE INSERT OR UPDATE OF (estudioa) ON PELIKULA
FOR EACH ROW
WHEN 100 = (SELECT COUNT(*)
            FROM PELIKULA
            WHERE urtea = NEW.urtea AND estudioa = NEW.estudioa)
BEGIN
    WRITE("Estudioak muga gainditu du. Pelikula hau beste estudioren batekoa izan beharko da");
    READ(izena);
    :NEW.estudioa = izena; //insert egiten denerako estudioaren izen berria jarri diogu
END;
```

Oharra: PELIKULA taulan aldaketa/sarrera egin baino lehen, azterketa egiten da, eta baldintza betetzen bada burutzen den ekintza estudioaren izena aldatzea da.

Hau da, NEW balioa aldatzen dugu, horrela INSERT edo UPDATE egiten denean bigarren txandan sartutako balio horrekin egingo da, eta ez jatorrizko eragiketak (abiarazlea altxarazi duenak) zuen balioarekin. Adibidez, hasieran INSERT INTO PELIKULA VALUES (10, ..., '20thCent') egiten da, honek trigger-a altxarazten du eta ikusten da 100 pelikula ja badituela, orduan beste estudio bat eskatuko zaigu (demagun 'Dream' ematen dugula); azkenean datu-basean geratuko den tupla (10, ..., 'Dream') izango da.

NEW balioaren aldaketa hori egin ahal izateko BEFORE motako abiarazlea izan behar du, aldaketa eragiketa egikaritu baino lehen burutu behar delako.

q) Urte berean filmatutako pelikula guztien batezbesteko iraupena 120 baino handiagokoa ez dela egiaztatu.

```
CREATE ASSERTION im_q
   CHECK (NOT EXISTS (SELECT urtea
       FROM PELIKULA
       GROUP BY urtea
       HAVING AVG(iraupena) > 120))
```

Abiarazlea:

```
CREATE TRIGGER t_q
BEFORE INSERT OR UPDATE OF (iraupena) ON PELIKULA
FOR EACH ROW
DECLARE
    batura, kop, batezbeste NUMBER;
    salbuespena  EXCEPTION;
BEGIN
    SELECT SUM(iraupena) INTO :batura FROM PELIKULA WHERE urtea = :NEW.urtea;
    SELECT COUNT(*) INTO :kop FROM PELIKULA WHERE urtea = :NEW.urtea;
    batezbeste = (batura + :NEW.iraupena)/(kop+1);
    kont = 0;

    WHILE (batezbeste > 120) LOOP
        IF (kont=5) THEN RAISE salbuespena ELSE kont = kont+1;
        WRITE("Batezbesteko iraupenak muga gainditu du. Pelikulari beste iraupen bat ezarri");
        READ(kopurua);
        batezbeste = (batura + kopurua)/(kop+1);
    END LOOP;
    :NEW.iraupena = kopurua;    // insert/update egiten denerako pelikularen iraupen berria jarri
END;
```

Oharra: PELIKULA-ko *urtea* atributuaren gainean antzeko beste abiarazle bat defini daiteke ere.

r) Exekutiboen ondarea 500.000 eurotatik behera jaistea ekidin, bi eratara: (i) horrelakorik gertatuko ote den eragiketa egin baino lehen egiaztatu, eta baiezkoan aldaketa ekidin; eta (ii) egiaztapena, aldaketaren ondoren egin eta emaitza okerra dela ikusita, eragiketa bertan behera utzi.

```
CREATE TRIGGER t_r_i
BEFORE INSERT OR UPDATE OF (ondarea) ON EXEKUTIBOA
FOR EACH ROW
WHEN (NEW.ondarea < 500.000)
DECLARE
     kopGaizki BOOLEAN;
     kopurua, kont NUMBER;
     ondareaGainditu
     EXCEPTION;
BEGIN
     kopGaizki = true;
     kont = 0;
     WHILE (kopGaizki  AND  kont<=5) LOOP
          kont = kont+1;
          WRITE("Ondareak 500.000-ko muga gainditu behar du. Beste bat sar ezazu");
          READ(kopurua);
          IF (kopurua > 500.000) THEN  kopGaizki = false  END IF;
     END LOOP;
      IF NOT kopGaizki THEN  :NEW.ondarea = kopurua
                        // insert/update egiten denerako pelikularen iraupen berria jarri
      ELSE  RAISE ondareaGainditu END IF;
END;
```

Oharra: Ondareak baldintza betetzen ez badu, gehienez 5 saio egiten ditugu arazoa konpondu arte. Konpondu ezean salbuespena altxatzen dugu INSERT edo
UPDATE egin ez dadin. Kopuru egokia sartuz gero, orduan NEW-ren ondare-balioa behar bezala aldatu eta eragiketa burutuko da.

==========

```
CREATE TRIGGER t_r_i
AFTER INSERT OR UPDATE OF (ondarea) ON EXEKUTIBOA
FOR EACH ROW
WHEN (NEW.ondarea < 500.000)
DECLARE
     ondareaGainditu EXCEPTION;
BEGIN
     RAISE ondareaGainditu;
END;
```

Oharra: Ondareak aipatu kopurua ez badu gainditzen, abiarazlearen ekintzak salbuespena altxatuko du. Salbuespen hori EXEKUTIBOA taulan INSERT edo UPDATE egin nahi zuen transakziora iritsiko da eta bertan tratatu beharko da. Transakzioaren programatzaileak erabaki beharko du nola tratatu, rollback egin edo komeni zaion beste edozer. 'after' motako abiarazle honetan, ezingo genuke *:**NEW**.ondarea = kopurua* erako adierazpenik idatzi, abiarazlea altxatu aurretik insert/update eragiketa jadanik burututa dagoelako.

4. Bedi guda-batailen eta hauetan parte hartu zuten itsasontzien datu-base sistema. Sisteman ondoren adierazten diren taulak definitzen dira. *ItsasKlasea* taulan itsasontzi-klaseen datuak gordetzen dira (zein motatakoak diren, guda- ontziak, gurutze-ontziak, etab.), zenbat kanoi dituzten, non diseinatutakoak diren, etab.). Itsasontzi bat klase bakarrekoa da eta bataila batean baino gehiagotan parte har dezake. *Konkista* taulan bataila bakoitzean itsasontzi bakoitzari gertatutakoa adierazten da (urperatu, irabazi, kaltetua).

ITSASKLASEA(klasea, herrialdea, mota, kanoiak, kalibrea, tara)
ITSASONTZIA(izena, *klasea*, uretaratzeUrtea)
BATAILA(izena, data)
KONKISTA(*bataila*, *itsasontzia*, emaitza)

Ondoren adierazten diren integritate-murriztapenak idatz itzazu. Kasuren batean, baldintzan bi taula badaude, bi murriztapen definitu behar izatea gerta daiteke.

a) Bi itsasontzi baino gehiago duen klaserik ezin da egon.

```
ALTER TABLE ITSASONTZIA
ADD CONSTRAINT 1A CHECK ((SELECT COUNT (*)
                          FROM ITSASONTZIA
                          GROUP BY Klasea) <2)
```

b) Herrialdeek ezin dituzte, aldi berean guda-itsasontziak eta gurutze-ontziak eduki.

```
CREATE ASSERTION im_b
   CHECK (NOT EXISTS(SELECT *
                     FROM ITSASKLASEA AS A INNER JOIN ITSASKLASEA AS B ON A.herrialdea=B.herrialdea
                     WHERE A.mota='guda-itsasontzia' AND B.mota='gurutze-ontzia'))
```

c) Uretaratze urtea jarrita badu, klaseak ezin du *null* izan.

$\forall x \in$ itsasontzia (x.uretaratzeUrtea jarrita du $\rightarrow$ x.klasea ezin da null izan)
$\Rightarrow \forall x \in$ itsasontzia (x.uretaratzeUrtea is not null $\rightarrow$ x.klasea is not null)
$\Rightarrow \forall x \in$ itsasontzia (x.uretaratzeUrtea is null $\vee$ x.klasea is not null)

```
ALTER TABLE ITSASONTZIA
ADD CONSTRAINT im_d CHECK (uretaratzeUrtea IS NULL OR klasea IS NOT NULL)
```

d) Klase guztietan badago bere izen berbera duen itsasontzi bat.

$\forall x \in$ itsasklasea (x.klasea bere itsasontzi baten izenaren berdina da)
$\Rightarrow \forall x \in$ itsasklasea $\exists y \in$ itsasontzia (y.izena = x.klasea $\wedge$ y.klasea=x.klasea)

```
ALTER TABLE ITSASKLASEA
  ADD CONSTRAINT im_e
    CHECK (klasea IN (SELECT izena FROM ITSASONTZIA WHERE klasea=ITSASKLASEA.klasea)
```

```
CREATE ASSERTION im_e
   CHECK (NOT EXISTS (SELECT *
                      FROM ITSASKLASEA AS A
                      WHERE A.klasea NOT IN (SELECT izena FROM ITSASONTZIA WHERE
                      klasea=A.klasea)))
```

e) 16" baino handiagoko kalibrea duten kanoiak dituen itsasontzi-klaserik ez dago.

```
ALTER TABLE ITSASKLASEA
  ADD CONSTRAINT im_f CHECK (kalibrea < 16)
```

Oharra: Kontuan hartu tuplaren batean *kalibrea* atributuaren balioa NULL izan arren (itsasontziak inolako kanoirik ez duelako), *"kalibrea < 16"* adierazpena *unknown* (edo *null*) balioarekin ebaluatzen dela, eta murriztapena bete egiten dela.

f) Itsasontzi-klase batek 9 kanoi baino gehiago baditu, beren kalibrea ezin da 14" baino handiagoa izan.

$\forall x \in$ itsasklasea (x.kanoiak>9 → x.kalibrea<=14)
$\Rightarrow \forall x \in$ itsasklasea (x.kanoiak<9 $\vee$ x.kalibrea<=14)

```
ALTER TABLE ITSASKLASEA
ADD CONSTRAINT im_g CHECK (kanoiak < 9 OR kalibrea <= 14)
```

g) Uretara bota baino lehen, itsasontzi batek ezin du batailaren batean parte hartu.

$\forall x \in$ itsasontzia (x.uretaratzeUrtea is null → $\neg \exists y \in$ konkista (y.itsasontzia=x.izena))
$\Rightarrow \forall x \in$ itsasontzia ($\neg$ x.uretaratzeUrtea is null $\vee \neg \exists y \in$ konkista (y.itsasontzia=x.izena)) <<<< Taulabarneko adierazpena idazteko
$\Rightarrow \neg \exists x \in$ itsasontzia (x.uretaratzeUrtea is null $\wedge \exists y \in$ konkista (y.itsasontzia=x.izena)) <<<< Taularteko adierazpena idazteko

Taulabarneko murriztapena:

```
ALTER TABLE ITSASONTZIA
  ADD CONSTRAINT im_h CHECK (uretaratzeUrtea IS NOT NULL
                             OR NOT EXISTS (SELECT *
                                            FROM KONKISTA
                                            WHERE itsasontzia=ITSASONTZIA.izena))
```

Taularteko murriztapena:

```
CREATE ASSERTION im_h
   CHECK (NOT EXISTS (SELECT *
                      FROM ITSASONTZIA AS A
                      WHERE uretaratzeUrtea IS NULL AND EXISTS (SELECT *
                                                                FROM KONKISTA
                                                                WHERE itsasontzia=A.izena)))
```

Oharra: KONKISTA taulan txertatze eragiketak egon badaitezke, taulabarneko murriztapena hautatzea ez litzateke egokia izango, murriztapen-mota hau ez delako barneko azpikontsultetan agertzen diren taulen aldaketekin egiaztatzen. Beraz, *uretaratzeUrtea* atributua *null* duen itsasontzi bat eduki dezakegu, eta berari dagokion KONKISTA tupla bat sar daiteke (baldintza hautsiz) eta murriztapena ez da egiaztatuko.

h) *ItsasKlasea* taularen oinarrizko gakoa *klasea* eta *herrialdea* jarri.

```
ALTER TABLE ITSASKLASEA
  ADD CONSTRAINT im_i PRIMARY KEY (klasea, herrialdea)
```

i) *Konkista* taulan azaltzen diren bataila guztiak *Bataila* taulan ere agertu behar direla adierazten duen integritate erreferentziala ezarri.

```
ALTER TABLE KONKISTA
  ADD CONSTRAINT im_j FOREIGN KEY bataila REFERENCES BATAILA(izena)
```

j) Konkista taulan azaltzen diren itsasontzi guztiak *Itsasontzia* taulan ere agertu behar direla adierazten duen integritate erreferentziala ezarri.

```
ALTER TABLE KONKISTA
  ADD CONSTRAINT im_k FOREIGN KEY itsasontzia REFERENCES ITSASONTZIA(izena)
```

k) 14 kanoi baino gehiago dituen itsasontzi klaserik ez egotea eskatzen da.

```
ALTER TABLE ITSASKLASEA
  ADD CONSTRAINT im_l CHECK (kanoiak < 14)
```

l) Uretaratze urteak ezin du null izan.

```
ALTER TABLE ITSASONTZIA
  ADD CONSTRAINT im_m CHECK (uretaratzeUrtea IS NOT NULL)
```

m) *ItsasKlasea* taulan klase berria sartzen denean, klase horren izen bera duen itsasontzi bat ere sartu behar da, *uretaratzeData* ezezaguna (*unknown*) duena.

```
CREATE TRIGGER t_n
AFTER INSERT ON ITSASKLASEA
FOR EACH ROW
BEGIN
    INSERT INTO ITSASONTZIA VALUES (:NEW.klasea, :NEW.klasea, null);
END;
```

n) *ItsasKlasea* taulan 35.000T baino gehiagoko tara duen klase berria sartzen denean, sarrera hori onartu baina taran zehazki 35.000T jarriz.

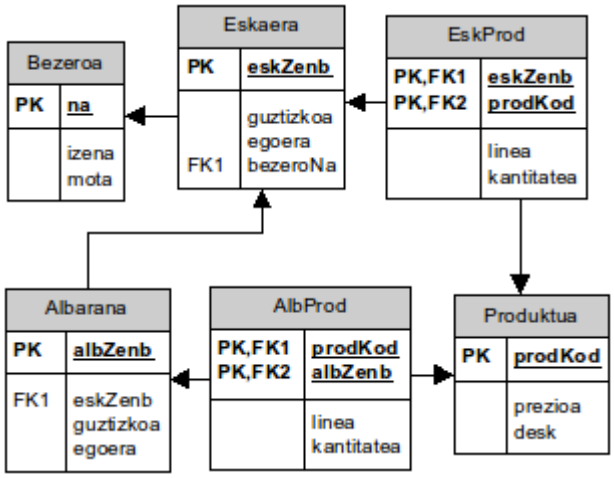
```
CREATE TRIGGER t_o
BEFORE INSERT ON ITSASKLASEA
FOR EACH ROW
WHEN (NEW.tara > 35.000)
BEGIN
     :NEW.tara = 35.000;
END;
```

o) *Konkista* taulan tupla berria sartzen denean, aipatu itsasontzia itsasontzia taulan ez badago, orduan itsasontzia taulan sartu. Aipatutako bataila ez badago *Bataila* taulan, orduan sartu. Kasu guztietan, behar denean *null* balioak jarri.

```
CREATE TRIGGER t_p1
BEFORE INSERT ON KONKISTA
FOR EACH ROW
WHEN (NOT EXISTS (SELECT * FROM ITSASONTZIA WHERE izena = NEW.itsasontzia))
BEGIN
    INSERT INTO ITSASONTZIA VALUES (:NEW.itsasontzia, null, null);
END;

CREATE TRIGGER t_p2
BEFORE INSERT ON KONKISTA
FOR EACH ROW
WHEN (NOT EXISTS (SELECT * FROM BATAILA WHERE izena = NEW.bataila))
BEGIN
    INSERT INTO BATAILA VALUES (:NEW.bataila, null);
END;
```

5. Bedi bezero, produktu eta eskaeren inguruko datu-base sistema, honako eskema erlazionala duena.



BEZEROA(na, izena, mota)
ESKAERA(eskZenb, guztizkoa, egoera, *bezeroNa*)
ESKPROD(*eskZenb, prodKod*, linea, kantitatea)
PRODUKTUA(prodKod, prezioa, desk)
ALBARANA(albZenb, *eskZenb*, guztizkoa, egoera)
ALBPROD(*prodKod, albZenb*, linea, kantitatea)

BEZEROA

na	izena	mota
11	...	E
22	...	O
33	...	E
44	...	O
55	...	E
66	...	O

ESKAERA

eskZenb	bezeroNa	guztizkoa	egoera
1	1	4000	Z
2	2	30000	K
3	3	4000	E

ESKPROD

eskZenb	prodKod	linea	kantitatea
1	1	1	5
1	25	2	1
1	5	3	2
2	25	1	12
3	10	1	2
3	20	2	1

PRODUKTUA

prodKod	Prezioa	desk
1	100	...
2	200	...
3	300	...
4	400	...
5	500	...
6	600	...
7	700	...
8	800	...
9	900	...
10	1000	...
20	2000	...
25	2500	...

ALBARANA

albZenb	eskZenb	guztizkoa	egoera
1	1	3200	O
2	1	700	O
3	1	100	O
4	2	25000	B
5	2	2500	P

ALBPROD

albZenb	prodKod	linea	kantitatea
1	1	1	2
1	5	2	1
1	25	3	1
2	1	1	2
2	5	2	1
3	1	1	1
4	25	1	10
5	25	1	1

Ondoren adierazten diren integritate-murriztapenak idatz itzazu (**taulabarneko murriztapen**, **taularteko murriztapen** edota **abiarazle (trigger)** moduan).

a) Bezeroak ohikoak (O) ala ez-ohikoak (E) dira.

```
ALTER TABLE BEZEROA
  ADD CONSTRAINT BezeroMota CHECK (mota IN ('O', 'E'));
```

b) Albaran baten balizko egoerak dira: Prestatuta (P), eskaeraren zati bat biltegian prest egon arren, oraindik bezeroari bidali ez zaionean; Bidalita (B), eskaeraren zati bat bezeroari bidali zaionean; eta Onartuta (O), bidalitakoa bezeroarengana iritsi eta honek onartu egin duenean.

```
ALTER TABLE ALBARANA
  ADD CONSTRAINT AlbaranaEgoera CHECK (egoera IN ('P', 'B', 'O'));
```

c) Eskaera baten balizko egoerak dira: Eskatuta (E), eskaera egin da baina dagokion albarana oraindik sortu ez denean; Kontuan hartuta (K), eskaerari dagokion albaran bat gutxienez badagoenean (berdin da zein den bere egoera); eta Zerbitzatuta (Z), onartutako albaran guztien artean, eskaera betetzen denean.

Lau murriztapen definitzen ditugu: taulabarnekoa, balio posibleak zehazteko, eta beste hirurak, balio horiek albaranekin duten erlazioa adierazteko.

```
ALTER TABLE ESKAERA
                    ADD CONSTRAINT EskaeraEgoera CHECK (egoera IN ('E', 'K', 'Z'));
```

$\forall x \in$ eskaera (x.egoera='E' $\rightarrow \neg\exists y \in$ albarana (y.eskZenb=x.eskZenb)) $\Rightarrow$ $\neg\exists x \in$ eskaera (x.egoera='E' $\wedge \exists y \in$ albarana (y.eskZenb=x.eskZenb)

```
CREATE ASSERTION  Eskaera_E
     CHECK (NOT EXISTS (SELECT *
                        FROM ESKAERA AS A
                        WHERE egoera = 'E' AND EXISTS (SELECT *
                                                       FROM ALBARANA
                                                       WHERE eskZenb = A.eskZenb)))
```

$\forall x \in$ eskaera (x.egoera='K' $\rightarrow \exists y \in$ albarana (y.eskZenb=x.eskZenb)) $\Rightarrow$ $\neg\exists x \in$ eskaera (x.egoera='K' $\wedge \neg\exists y \in$ albarana (y.eskZenb=x.eskZenb)

```
CREATE ASSERTION  Eskaera_K
     CHECK (NOT EXISTS (SELECT *
                        FROM ESKAERA AS A
                        WHERE egoera = 'K' AND NOT EXISTS (SELECT *
                                                           FROM ALBARANA
                                                           WHERE eskZenb = A.eskZenb)))
```

$\forall x \in$ eskaera (x.egoera='Z' $\rightarrow$ x.guztizkoa=sum(albaran.guztizkoa....)) $\Rightarrow$ $\neg\exists x \in$ eskaera (x.egoera='Z' $\wedge \neg$ (x.guztizkoa=sum(albaran.guztizkoa....))

```
CREATE ASSERTION  Eskaera_Z
     CHECK (NOT EXISTS (SELECT *
                        FROM ESKAERA AS A
                        WHERE egoera = 'Z'
                              AND guztizkoa <> (SELECT SUM(guztizkoa)
                                               FROM ALBARANA
                                               WHERE eskZenb = A.eskZenb AND egoera='O')))
```

d) Ez-ohiko bezeroek ezin dituzte 5000€ edo 10 produktu baino gehiagoko eskaerak egin (kontuz, egoera hau onargarria da: ez-ohiko bezero batek produkturen baten 10 unitate baino gehiago eskatzea, 5000€ gainditu gabe betiere).

$\forall x \in$ bezeroa (x.mota='E' $\rightarrow$ ezin da honakoa gertatu: x-ren eskaeren guztizkoa > 5000 $\vee$ x-ren eskaeren produktu-kopurua> 10)
$\Rightarrow \forall x \in$ bezeroa (x.mota='E' $\rightarrow$ x-ren eskaeren guztizkoa < 5000 $\wedge$ x-ren eskaeren produktu-kopurua < 10)
$\Rightarrow \neg\exists x \in$ bezeroa (x.mota='E' $\wedge$ (x-ren eskaeren guztizkoa > 5000 $\vee$ x-ren eskaeren produktu-kopurua < 10))

```
CREATE ASSERTION  EzOhikoBezeroenEskaerak
     CHECK (NOT EXISTS (SELECT *
                        FROM ESKAERA AS A INNER JOIN BEZEROA ON bezeroNa = na
                        WHERE mota = 'E'
                          AND (guztizkoa > 5000  OR  10 < (SELECT COUNT(*)
                                                           FROM ESKPROD
                                                           WHERE eskZenb = A.eskZenb))))
```

e) Eskaera bati dagokion guztizko balioa, eskaerako produktuen prezioa eta eskatutako unitate-kopuruaren arteko biderkaketa da. Eskaera prozesatzen den bitartean produktuen prezioa alda daiteke; hala ere, eskaera egin zeneko datako prezioak mantendu egingo zaizkio. Aldiz, bezeroak eskaerako produkturen batekin aldaketa (eskatutako unitate-kopurua gehitu edo murriztu) egiten badu, eskaeraren guztizko berria kalkulatzean uneko prezioak aplikatuko dira.

```
ALTER TABLE ESKAERA
  ADD CONSTRAINT guztizkoaKontrola
        CHECK (guztizkoa = (SELECT SUM(ESKPROD.kantitatea*PRODUKTUA.prezioa)
                            FROM ESKPROD NATURAL JOIN PRODUKTUA
                            WHERE eskZenb = ESKAERA.eskZenb))
```

Oharra: Produktuaren prezioa alda daiteke, baina aurreko eskaerak zeuden bezala geratu behar dira. Beraz, murriztapena ESKAERAren taulabarneko bezala definitu behar dugu, PRODUKTUA aldatzean egiazta ez dadin.

Taularteko murriztapen bezala definituko bagenu, jarraian erakusten den moduan, murriztapena edozein kasutan egiaztatuko litzateke, PRODUKTUA nahiz ESKPROD aldatu. Kasu honetan ez zaigu horrelakorik interesatzen, produktuaren prezioa aldatzen denean ez dugu produktu horren aurreko eskaerak aztertzea nahi. Beraz taularteko murriztapena ez da aukera egokia.

```
CREATE ASSERTION guztizkoaKontrola
     CHECK (NOT EXISTS (SELECT *
                        FROM Eskaera AS A
                       WHERE guztizkoa <> (SELECT SUM (EskProd.kantitatea * Produktua.prezioa)
                                           FROM EskProd NATURAL JOIN Produktua
                                           WHERE eskZenb = A.eskZenb)))
```

Bestalde, eskatutako unitate-kopurua gehitu edo murriztu delako, eskaera aldatuz gero, guztizko berria kalkulatu behar da. Horretarako abiarazle (trigger) bat definituko dugu.

```
CREATE TRIGGER EskaeraAldaketaKontrola
AFTER INSERT OR UPDATE(kantitatea) ON ESKPROD
FOR EACH ROW
DECLARE
    tot, nprezioa NUMBER;
BEGIN
    #sql tot = {SELECT SUM(P.prezioa * EP.kantitatea)
                FROM PRODUKTUA AS P NATURAL JOIN ESKPROD AS EP
                WHERE EP.eskZenb = :NEW.eskZenb AND EP.prodKod <> :NEW.prodKod};
    #sql :nprezioa = {SELECT prezioa FROM PRODUKTUA WHERE prodKod = :NEW.prodKod};
    tot = tot + (nprezioa * :NEW.kantitatea);
    #sql {UPDATE ESKAERA SET guztizkoa = :tot WHERE eskZenb = :NEW.eskZenb};
END
```

Oharra: AFTER motakoa izan arren, lehenengo SELECT-en ezin dugu *EP.prodKod <> :NEW.prodKod* baldintza kendu, horrela produktu guztien (aldatutako hori barne) kalkulua egin dezan, ESKPROD taula aldaketa-fasean dagoela esanez errore bat ateratzen baita.

Abiarazlea BEFORE motakoa balitz, gorputzean, ESKAERA taularen UPDATE egin baino lehen, *guztizkoaKontrola* murriztapena atzeratu egin beharko genuke, ESKPRODeko tupla sartu/aldatu ondoren egiazta dadin.

f) Albaran batean sartutako produktuak eskaera bakarrari dagozkio.

```
ALTER TABLE ALBARANA  ALTER COLUMN eskZenb NOT NULL;

CREATE ASSERTION albaranaEskaeraBakarrarena
    CHECK (NOT EXISTS (SELECT *
                       FROM ALBARANA AS A
                       WHERE EXISTS (SELECT *
                                     FROM ALBPROD
                                     WHERE albKod = A.albKod
                                       AND kodProd NOT IN (SELECT kodProd
                                                           FROM ESKPROD
                                                           WHERE eskZenb = A.eskZenb))))
```

Oharra: Lehenengo murriztapenarekin, albarana eskaeraren batekin lotzen dela kontrolatzen dugu. Bigarrenarekin, albaranean azaltzen diren produktuak albaranaren eskaeran ere azaltzen direla kontrolatzen da.

g) Zerbitzatutakoa ezin da eskatutakoa baino gehiago izan. Beste era batera esanda, produktu bakoitzarekin, eskatutako kantitatea albaranetan erregistratu edo zerbitzatutakoa baino handiagoa edo berdina izan behar da.

```
CREATE ASSERTION AlbaranenGuztizkoa
      CHECK (NOT EXISTS (SELECT *
                         FROM ESKAERA AS A NATURAL JOIN ESKPROD AS B
                         WHERE B.kantitatea< (SELECT SUM(C.kantitatea)
                                              FROM ALBPROD AS C NATURAL JOIN ALBARANA AS D
                                              WHERE D.eskZenb = B.eskZenb AND  C.prodKod=B.prodKod)))
```

h) Jadanik zerbitzatuta dagoen eskaera ezin da aldatu.

Oharra: Eskaera zerbitzatuta badago aldaketa atzera bota behar dugu. Beraz, abiarazlean egiaztapena egin eta gaizki dagoela ikusi eta gero, salbuespen bat altxarazten dugu (***raise*** *salbuespena* aginduaren bitartez). Salbuespena DECLARE atalean erazagutzen dugu. Horrela, gaizki dagoen update edo delete agindua exekutatzen saiatzean, errorea itzuliko da eta azkenean agindua ezin izango da burutu.

Dena borobiltzeko, delete edo update eragiketak burutzen diren programan salbuespen hori tratatu egin beharko litzateke. Honela, gutxi gorabehera:

```
#sql {set  transaction isolation level read committed} ;
...
try {    #sql {update Eskaera ....}   }
catch(Exception e) {   if (e.getErrorCode() == ZerbitzatutakoEskaera) {  System.out.println("Zerbait gaizki. Berriz saiatu");   #sql {rollback};   }
                                /* eta hasierara joan */
                    }
                    ...
#sql {commit} ;
```

```
CREATE TRIGGER EzAldatuZerbitzatutakoEskaera_1
BEFORE DELETE OR UPDATE ON ESKAERA
FOR EACH ROW
WHEN (OLD.egoera = 'Z')
DECLARE
    ZerbitzatutakoEskaera EXCEPTION;
BEGIN
    RAISE ZerbitzatutakoEskaera;
END;

CREATE TRIGGER EzAldatuZerbitzatutakoEskaera_2
BEFORE DELETE OR UPDATE ON ESKPROD
FOR EACH ROW
WHEN (EXISTS (SELECT *  FROM ESKAERA AS A NATURAL JOIN ESKPROD AS B
              WHERE A.egoera = 'Z' AND B.eskZenb = OLD.eskZenb))
DECLARE  ZerbitzatutakoEskaera EXCEPTION;
BEGIN
    RAISE ZerbitzatutakoEskaera;
END
```

i) Eskaera batean ezin dira jadanik bidaltzen hasita dauden produktuen datuak aldatu; 'bidaltzen hasita'-k esan nahi du bidalita edo onartuta dagoen albaranen batean agertzen dela. Hala bada, salbuespen bat altxarazi.

```
CREATE TRIGGER EzAldatuBidalitaDaudenProduktuenEskaerak
BEFORE DELETE OR UPDATE ON ESKPROD
FOR EACH ROW
WHEN (EXISTS (SELECT *
              FROM ALBARANA AS A NATURAL JOIN ALBPROD AS B
              WHERE (A.egoera = 'B' OR A.egoera = 'O')
                AND A.eskZenb = OLD.eskZenb AND B.prodKod = OLD.prodKod)
DECLARE
    AlbaranaBidalitaOnartua EXCEPTION;
BEGIN
    RAISE AlbaranaBidalitaOnartua;
END
```

Demagun aurreko murriztapen guztiak DEFERRABLE INITIALLY IMMEDIATE motakoak direla. Datubasearen gainean ondorengoko eragiketak egikarituko dituzten prozedurak idatz itzazu. Aurrean definitutako integritate- murriztapenetakoren bat transakzioko uneren batean ez betetzea gerta daitekeela kontuan hartu behar da. Horregatik, integritate-murriztapenen egiaztapena atzeratzeko aukera erabiliko dugu, baina egiaztapen hori transakzioaren barnean ahal den gutxien atzeratzen saiatu beharko dugu.

j) Ez-ohikoa den eta 11 NA duen bezeroaren eskaera sartu. Bertan honako kodeak dituzten produktuak eskatu dira: 2 (5 unitate), 4 (5 unitate) eta 1 (10 unitate).

```
int kod, prezioBat, prezioBi, prezioHiru, total=0;
#sql {select max(eskZenb) into :kod from ESKAERA};
kod = kod+1;
#sql {select prezioa into :prezioBat from PRODUKTUA where prodKod = 2};
#sql {select prezioa into :prezioBi from PRODUKTUA where prodKod = 4};
#sql {select prezioa into :prezioHiru from PRODUKTUA where prodKod = 1};
total = (prezioBat*5) + (prezioBi*5) + (prezioHiru*10);

#sql {set transaction isolation level read committed};    // Transakzioaren hasiera
#sql {set constraints guztizkoaKontrola, EzOhikoBezeroenEskaerak deferred};
#sql {insert into ESKAERA values (:kod, :total, 'E', 11)};
#sql {insert into ESKPROD values (:kod, 2, null, 5)};
#sql {insert into ESKPROD values (:kod, 4, null, 5)};
#sql {insert into ESKPROD values (:kod, 1, null, 10)};
#sql {set constraints guztizkoaKontrola, EzOhikoBezeroenEskaerak immediate};
#sql {commit};
```

Oharra: Lehenik ESKAERAko tupla sartu behar da, ESKPROD-eko integritate erreferentzialaren murriztapenak arazoak eman ez ditzan. Bestalde, *guztizkoa* atributuaren balioa, *guztizkoaKontrola* murriztapenak kontrolatzen du, eta, beraz, lehendabizi ESKAERAn sartu ahal izateko, murriztapen horren egiaztapena atzeratu behar dugu. Gainera, *EzOhikoBezeroenEskaerak* murriztapenak bezeroek eskatutako unitateak kontrolatzen dituenez, hau ere ESKPRODeko tupla guztiak sartu ondoren egiaztatu beharko da.
Transakzioa bukatu baino lehen, murriztapenak IMMEDIATE modura pasatzen ditugu; horrela, baten bat beteko ez balitz, tratatu ahal izango genuke *try/catch* egitura batekin edo. IMMEDIATE modura esplizitukoi pasatzen ez baldin badugu, *commit* egin baino lehen, sistemak automatikoki egiaztatuko du, eta murriztapenen bat betetzen ez bada, automatikoki *rollback* egingo du.
Murriztapenen egiaztapena ezin da ESKPROD-eko tuplak sartu baino lehen egin.

k) Ez-ohikoa den 77 bezeroaren eskaera sartu. Bertan 11 kodea duen produktuaren 20 unitate eskatu dira (produktuaren prezioa 110€ dira).

```
int kod;
#sql {select max(eskZenb) into :kod from Eskaera};
kod = kod+1;

#sql {set transaction isolation level read committed};    // Transakzioaren hasiera
#sql {set constraint guztizkoaKontrola, EzOhikoBezeroenEskaerak deferred}; //(d) eta (e) atalekoak
#sql {insert into Eskaera values (:kod, 77, 20*110, 'E')};
#sql {insert into EskProd values (:kod, 11, null, 20)};
#sql {set constraints guztizkoaKontrola, EzOhikoBezeroenEskaerak immediate};
#sql {commit};
```

Oharra: Lehenik ESKAERAko tupla sartu behar da, ESKPROD-eko integritate erreferentzialaren murriztapenak arazoak eman ez ditzan. *Guztizkoa* atributuaren balioa *guztizkoaKontrola* murriztapenak kontrolatzen du, eta, beraz, lehendabizi ESKAERAn sartu ahal izateko, murriztapen horren egiaztapena atzeratu behar dugu. Gainera, *EzOhikoBezeroenEskaerak* murriztapenak bezeroek eskatutako unitateak kontrolatzen dituenez, hau ere ESKPRODeko tupla guztiak sartu ondoren egiaztatu beharko da. Transakzioa bukatu baino lehen murriztapenak IMMEDIATE modura pasatzen ditugu, horrela baten bat beteko ez balitz, tratatu ahal izango genuke try/catch egitura batekin edo. IMMEDIATE modura guk ez baldin badugu pasatzen, *commit* egin baino lehen sistemak automatikoki egiaztatuko du eta, murriztapenen bat ez bada betetzen automatikoki *rollback* egingo du.
Bestaldetik kontuan hartu murriztapenen egiaztapena ezin dela dagozkien ESKPROD-eko tuplak sartu baino lehen egin.

Deferred eta *immediate* aukerekin joka daitekeela ikusteko, eta erroreetatik berreskuratzea posible dela ikusteko, honako kode alternatiboa ematen dugu.

```
        int kod, kop, tot;
        while (true) {
               try{
                 kod = kodeaKalkulatu();
                 read(kop); read(tot);
                 #sql {set transaction isolation level read committed};  // Transakzioaren hasiera
                 #sql {set constraint guztizkoaKontrola deferred};
                           //guztizkoaren kontrola bakarrik, errazago egitearren
                 #sql {insert into Eskaera values (:kod, 77, :tot, 'E')};
                 #sql {insert into EskProd values (:kod, 11, null, 20)};
                 #sql {set constraints guztizkoaKontrola immediate};
                 #sql {commit};
                 break;
            }catch(Exception e)
                    {
                 if (e.getErrorCode()==2020) {
                      println('Guztizkoa gaizki. Sartu beste bat');  read(tot);
                                          // sartutako guztizko berria ondo dagoela suposatuz
                      #sql {delete from Eskaera where eskZenb=:kod};
                      #sql {insert into Eskaera values (:kod, 77, :tot, 'E')};
                      #sql {commit};
                 }
                 else { #sql {rollback}; }
            }
        }
```

Lehendabizi murriztapena atzeratu eta gero (*commit* egin baino lehen) berehalako egoerara pasa ordez, beste aukera bat sistemak hitzarmen garaian egiteko uztea da. Horrela, eta egiaztapena beteko ez balitz, transakzioan *rollback* automatikoa egingo litzateke eta ezingo genuke errorea 'konpondu'.

l) 77 bezeroari 11 kodea duen produktuaren 12 unitate bidali zaizkiola erregistratu eta dagokion albarana sortu.

```
        // oraindik erabat zerbitzatu ez diren eskaeren zerrenda lortu
#sql iter = { select A.eskZenb from ESKAERA as A natural join ESKPROD as B
              where A.bezeroNa = 77 and B.prodKod=11
                and B.kantitatea > (select sum(D.kantitatea)
                                    from ALBARANA as C natural join ALBPROD as D
                                    where C.eskZenb=A.eskZenb and D.prodKod=B.prodKod) };

while (iter.next()) {
    try {
        #sql {set transaction isolation level read committed};    // Transakzioaren hasiera
        #sql {select prezioa into :prez from PRODUKTUA where prodKod = 11};
        #sql {select max(albZenb) into :kod from ALBARANA};
        kod=kod+1;
        #sql {set constraints AlbaranenGuztizkoa deferred};   // (g) atalekoa
        #sql {insert into ALBARANA values (:kod, :iter.eskZenb(), :prez*12, 'B'};
        #sql {insert into ALBPROD values (:kod, 11, null, 12)};
        #sql {set constraints AlbaranenGuztizkoa deferred};
        #sql {commit};
        break;
    } catch (SQLException e) {
        if (e.getErrorCode() == 2020) {
                System.out.println("Albaranean sartutako kopurua (12) eskaerakoa gainditzen du");
                #sql {rollback};
                break;
        }
    }
}
```

m) 3 zenbakia duen eskaera ezabatu.

```
try {
     #sql {set transaction isolation level read committed};    // Transakzioaren hasiera
     #sql {delete from ESKAERA where eskZenb = 3};
     #sql {commit};
} catch (SQLException e) {
     if (e.getErrorCode() == 2020) {
          System.out.print("Eskaera ezin da ezabatu, berarekin erlazionatutako albaran
          batzuk"); System.out.println(" bidalita daudelako");
          #sql {rollback};
     }
}
```