

Constrângeri

Chei Străine

Constrângeri Locale și Globale
“Triggers”

Constrângeri și Trigere

- O *constrângere* este o relație de legătură între elemente de date, pe care SGBD-ul este obligat să le forțeze.
 - Exemplu: constrângeri de tip cheie.
- "*Triggers*" sunt executate atunci când apare o condiție specifică, de exemplu, adăugarea (INSERT) unei tuple.
 - Sunt mai ușor de implementat decât constrângerile complexe.

Tipuri de Constrângeri

- Chei.
- Cheie-străină, sau integritate-referențială.
- Constrângeri la nivel Valoare.
 - Constrâng valorile unui atribut particular.
- Constrângeri la nivel Tuplă.
 - Relație de legătură între componente.
- Constrângeri Declarative ("Assertions"): orice expresie logică SQL.

Recapitulare: Chei de tip Atribut-Singular

- Se folosește PRIMARY KEY sau UNIQUE după tip în declarația atributului.
- Exemplu:

```
CREATE TABLE Beers (  
    name        CHAR(20)  UNIQUE,  
    manf        CHAR(20)  
);
```

Recapitulare: Cheie Multi-atribut

- În relația vânzări (Sells) barul (bar) și berea (beer) împreună formează cheia:

```
CREATE TABLE Sells (  
    bar          CHAR(20) ,  
    beer         VARCHAR(20) ,  
    price        REAL ,  
    PRIMARY KEY (bar, beer)  
);
```

Chei străine

- Valorile ce apar în attributele unei relații trebuie să apară împreună în anumite attribute din altă relație.
- Exemplu: în `Sells(bar, beer, price)`, se așteaptă ca valoarea pentru bere (beer) să apară de asemenea în `Beers.name`.

Exprimarea Cheilor Străine

- Se folosește cuvântul cheie REFERENCES, în felul următor:
 1. După un atribut (pentru chei formate dintr-un singur atribut).
 2. Un element al schemei:
FOREIGN KEY (<listă de attribute>)
REFERENCES <relație> (<attribute>)
- Atributele referențiate trebuie să fie declarate PRIMARY KEY sau UNIQUE.

Exemplu: În cadrul definiției Atributului

```
CREATE TABLE Beers (  
    name      CHAR(20) PRIMARY KEY,  
    manf      CHAR(20) );
```

```
CREATE TABLE Sells (  
    bar       CHAR(20),  
    beer      CHAR(20) REFERENCES Beers(name),  
    price     REAL );
```


Exemplu: Element al Schemei

```
CREATE TABLE Beers (  
    name      CHAR(20) PRIMARY KEY,  
    manf      CHAR(20) );  
  
CREATE TABLE Sells (  
    bar       CHAR(20),  
    beer      CHAR(20),  
    price     REAL,  
    FOREIGN KEY(beer) REFERENCES  
        Beers(name) );
```

Fortarea Constrângerilor Cheie Străină

- Dacă există o constrângere cheie străină de la relația R la relația S , două violări sunt posibile:
 1. La o adăugare (insert) sau modificare (update) a unei tuple în R se introduc valori ce nu se regăsesc în S .
 2. După o ștergere (delete) sau modificare (update) a unei tuple în S rezultă anumite tuple inconsecvente în R (nu au pereche în S).

Exemplu:

- Presupunem $R = \text{Sells}$, $S = \text{Beers}$.
- O adăugare (insert) sau modificare (update) în **Sells** ce introduce o bere (beer) inexistentă trebuie respinsă.
- O ștergere (delete) sau modificare (update) în **Beers** ce elimină o valoare "beer" ce se regăsește în anumite tuple din **Sells** poate fi gestionată în una din trei modalități:

Forțarea Constrângerilor Cheie Străină

1. *Implicit*: Modificarea este respinsă.
2. *"Cascade"*: Se aplică aceleași modificări în Sells.
 - *"Delete beer"*: se elimină tuple în Sells.
 - *"Update beer"*: se modifică valori în Sells.
3. *"Set NULL"*: se completează valoarea NULL pentru "beer".

Exemplu: "Cascade"

- Se șterge tupla "Bud" din Beers:
 - Urmarea este că se șterg toate tuplele din Sells ce au "beer = 'Bud'".
- Se modifică tupla "Bud" completând 'Budweiser' în locul lui 'Bud' în Beers:
 - Urmarea este că se modifică toate tuplele din Sells cu "beer = 'Bud'" la "beer = 'Budweiser'".

Exemplu: "Set NULL"

- Se șterge tupla "Bud" din Beers:
 - Se modifică toate tuplele din Sells cu "beer = 'Bud'" la "beer = NULL".
- Se modifică tupla "Bud" completând 'Budweiser' în locul lui 'Bud' în Beers :
 - Se modifică toate tuplele din Sells cu "beer = 'Bud'" la "beer = NULL".

Alegerea unei Politici de aplicare modificări

- La declararea cheii străine, se poate preciza politica SET NULL sau CASCADE independent pentru "delete" față de "update".
- Se completează declarația cheii străine cu:
ON [UPDATE, DELETE][SET NULL CASCADE]
- Pot fi folosite două astfel de clauze.
- În caz că această clauză lipsește se aplică politica "rejected".

Exemplu: Stabilire Politică

```
CREATE TABLE Sells (  
    bar      CHAR(20),  
    beer     CHAR(20),  
    price    REAL,  
    FOREIGN KEY (beer)  
        REFERENCES Beers (name)  
        ON DELETE SET NULL  
        ON UPDATE CASCADE  
);
```


Constrângeri la nivel Valoare (Atribut)

- Se adaugă CHECK(<condiție>) la declarația pentru un atribut.
- Condiția poate folosi numele atributului, dar orice altă relație sau atribut ce apar trebuie folosite într-o interogare imbricată (subquery).

Exemplu: Check la nivel atribut

```
CREATE TABLE Sells (  
    bar      CHAR(20),  
    beer     CHAR(20)    CHECK ( beer IN  
                                (SELECT name FROM Beers) ),  
    price    REAL CHECK ( price <= 5.00 )  
);
```

Când se aplică "Check" la nivel atribut?

- Constrângerile la nivel valoare de atribut (check) sunt aplicate în momentul când se efectuează o adăugare (insert) sau modificare (update).
- **Exemplu:** `CHECK (price <= 5.00)` verifică orice preț nou și respinge modificarea (acelei tuple) dacă prețul este mai mare de 5 \$.
- **Exemplu:** `CHECK (beer IN (SELECT name FROM Beers))` nu se verifică atunci când o bere (beer) este ștearsă din Beers (așa cum se verifică la constrângerea de tip cheie străină).

Constrângeri la nivel Tuplă

- CHECK (<condiție>) poate fi adăugată ca element al schemei de relație.
- Condiția poate face referire la orice atribut al relației.
 - Alte attribute sau relații necesită o interogare imbricată (subquery).
- Se verifică doar la INSERT sau UPDATE.

Exemplu: "Check" la nivel tuplă

- Doar "Joe's Bar" poate vinde bere mai scumpă de 5 (\$ sau altă monedă):

```
CREATE TABLE Sells (  
    bar          CHAR(20),  
    beer         CHAR(20),  
    price        REAL,  
    CHECK (bar = 'Joe's Bar' OR  
           price <= 5.00)  
);
```

Constrângeri Declarative ("Assertions")

- Sunt elemente ale schemei bazei de date, cum sunt relațiile sau vederile.

- Se definesc în felul următor:

```
CREATE ASSERTION <nume>  
CHECK (<condiție>);
```

- Condiția se poate referi la orice relație sau atribut din schema bazei de date.

Exemplu: Constrângere "Assertion"


- În `Sells(bar, beer, price)`, nici un bar nu poate vinde la un preț mediu mai mare de 5 \$.

```
CREATE ASSERTION NoRipoffBars CHECK (  
  NOT EXISTS (  
    SELECT bar FROM Sells
```

```
    GROUP BY bar  
    HAVING 5.00 < AVG(price)
```

```
  ));
```

Baruri cu
Preț mediu
sub 5 \$



Exemplu: Constrângere "Assertion"

- În `Drinkers(name, addr, phone)` și în `Bars(name, addr, license)`, nu pot exista mai multe baruri decât persoane.

```
CREATE ASSERTION FewBar CHECK (  
    (SELECT COUNT(*) FROM Bars) <=  
    (SELECT COUNT(*) FROM Drinkers)  
);
```


Când se aplică Constrângerea "Assertion"?

- În principiu, fiecare constrângere declarativă (assertion) se verifică după orice modificare a oricărei relații din baza de date.
- Un sistem inteligent ar observa doar acele modificări ce pot cauza violarea unei constrângeri declarative.
 - **Exemplu:** O modificare (update) în Beers nu afectează "FewBar". De asemenea nici o adăugare în Drinkers.

“Triggers”: Motivație

- Constrângerile declarative (assertion) reprezintă o unealtă puternică, dar SGBD-ul nu știe cu exactitate când trebuie verificate.
- Constrângerile la nivel Atribut și Tuplă sunt verificate la momente de timp bine determinate, dar nu sunt unelte la fel de puternice.
- Trigerele permit utilizatorului să decidă când să se verifice orice fel de condiție.

Reguli "Event-Condition-Action"

- O altă denumire pentru "trigger" este "*ECA rule*", sau "*event-condition-action* rule".
- *Event* : de obicei este un tip de modificare bază de date, de exemplu, adăugare în Sells.
- *Condition* : Orice expresie logică SQL.
- *Action* : orice secvență de instrucțiuni SQL.

Exemplu: Trigger

- În loc să se folosească o constrângere cheie străină și să se respingă adăugările în `Sells(bar, beer, price)` pentru mărci de bere necunoscute, un “trigger” poate adăuga acea marcă de bere în Beers, cu valoarea NULL pentru producător (manf).

Exemplu: Definiția Trigerului

```
CREATE TRIGGER BeerTrig
  AFTER INSERT ON Sells
  REFERENCING NEW ROW AS NewTuple
  FOR EACH ROW
  WHEN (NewTuple.beer NOT IN
        (SELECT name FROM Beers))
  INSERT INTO Beers(name)
  VALUES(NewTuple.beer);
```

Eveniment

Condiție

Acțiune

Opțiuni: Evenimentul

- AFTER poate fi BEFORE.
- Dacă relația este o vedere (view), atunci poate fi **INSTEAD OF**.
 - Actualizările prin intermediul vederilor pot fi efectuate prin definiția de trigere ce translatează actualizările în operații asupra tabelelor de bază.
- INSERT poate fi DELETE sau UPDATE.
- UPDATE poate fi UPDATE . . . pentru un atribut particular.

Opțiuni: FOR EACH ROW

- Trigerele sunt “row-level” sau “statement-level”.
- FOR EACH ROW indică “row-level”; dacă lipsește indică “statement-level”.
- *Row level triggers* : se execută o singură dată pentru fiecare tuplă modificată.
- *Statement-level triggers* : se execută o singură dată pentru o instrucțiune SQL, indiferent de numărul tuplelor modificate.

Opțiuni: REFERENCING

- Instrucțiunile INSERT implică o tuplă nouă (pentru "row-level") sau o tabelă nouă (pentru "statement-level").
 - "tabela nouă" este mulțimea tuplelor inserate.
- DELETE implică o tuplă sau o tabelă "old".
- UPDATE implică ambele.
- Se face referire la aceste lucruri prin
[NEW OLD][TUPLE TABLE] AS <nume>

Caz particular MS SQL Server

- Trigerele sunt “statement-level”.
- Pentru referirea la valorile vechi există tabela (temporară) “deleted”.
- Pentru referirea la valorile noi există tabela (temporară) “inserted”.

Opțiuni: Condiția

- Orice condiție ce are valoare logică.
- Este evaluată pentru starea bazei de date înainte sau după evenimentul ce a declanșat triggerul, depinde de ce cuvânt cheie a fost folosit în definiția triggerului (BEFORE sau AFTER).
 - Evaluarea se face totdeauna înainte ca modificările să aibă efect.
- Tupla/Tabela "new"/"old" este accesată prin intermediul numelor folosite în clauza REFERENCING.

Opțiuni: Acțiunea

- Pot exista mai multe instrucțiuni SQL ca și acțiune.
 - Se folosesc BEGIN . . . END atunci când există mai multe instrucțiuni SQL.
- Interogările nu au sens ca și acțiune, prin urmare există limitare doar la actualizări.

Caz particular MS SQL Server

- Acțiunea unui trigger poate conține orice instrucțiune validă Transact SQL:

```
IF OBJECT_ID ('Purchasing.LowCredit','TR') IS NOT NULL
    DROP TRIGGER Purchasing.LowCredit
GO
CREATE TRIGGER LowCredit ON Purchasing.PurchaseOrderHeader
AFTER INSERT
AS
DECLARE @creditrating tinyint, @vendorid int
SELECT @creditrating = v.CreditRating, @vendorid = p.VendorID
FROM Purchasing.PurchaseOrderHeader p INNER JOIN inserted i ON
    p.PurchaseOrderID = i.PurchaseOrderID JOIN Purchasing.Vendor v on
    v.VendorID = i.VendorID
IF @creditrating = 5
BEGIN
    RAISERROR ('This vendor''s credit rating is too low to accept new
        purchase orders.', 16, 1)
ROLLBACK TRANSACTION
END
```

Exemplu

- Se folosesc relațiile **Sells(bar, beer, price)** și **RipoffBars(bar)** (are un singur atribut), pentru a întreține lista barurilor ce cresc prețul oricărei mărci de bere cu mai mult de 1 \$.

Trigerul

```
CREATE TRIGGER PriceTrig
```

```
AFTER UPDATE OF price ON Sells
```

Evenimentul –
doar modificări
ale prețului

```
REFERENCING
```

```
OLD ROW AS ooo
```

```
NEW ROW AS nnn
```

Update ne permite
să discutăm despre
tuple "old" și "new"

Condiția:
creșterea
prețului
mai mult
de 1 \$

```
FOR EACH ROW
```

Se ia în considerare
fiecare preț modificat

```
WHEN(nnn.price > ooo.price + 1.00)
```

```
INSERT INTO RipoffBars  
VALUES(nnn.bar);
```

La creșterea prețului
suficient de mult, se
adaugă barul în
RipoffBars

Vederi

“View”

- O vedere (*view*) este o relație definită în termeni tabele stocate (numite *tabele de bază*) și alte vederi.
- Există două categorii de vederi:
 1. *Virtuale* = datele afișate nu sunt stocate în baza de date; reprezintă doar o interogare pentru construirea relației.
 2. *Materializate* = sunt stocate.

Declararea Vederilor

- Se declară cu instrucțiunea:

```
CREATE [MATERIALIZED] VIEW  
    <nume> AS <interogare>;
```

- Implicit o vedere este virtuală.

Exemplu

- **CanDrink(drinker, beer)** este o vedere ce “conține” perechi drinker-beer astfel încât persoana frecventează cel puțin un bar ce oferă berea la vânzare:

```
CREATE VIEW CanDrink AS
  SELECT drinker, beer
  FROM Frequents, Sells
  WHERE Frequents.bar = Sells.bar;
```

Exemplu: Accesul la o Vedere

- Interogarea unei vederi se face ca și în cazul tabeli de bază.
 - Actualizarea unei vederi este limitată la o singură tabelă de bază.
- Exemplu de interogare:

```
SELECT beer FROM CanDrink  
WHERE drinker = 'Sally';
```


Trigere definite pentru Vederi

- În general, este imposibil de actualizat o vedere virtuală, pentru că ea nu există fizic.
- Există însă triggerul **INSTEAD OF**, ce permite interpretarea actualizărilor produse vederii astfel încât actualizările să aibă sens.
- **Exemplu:** Vederea Synergy are triplete (**drinker, beer, bar**) cu semnificația barul servește berea, persoana frecventează barul și persoanei îi place berea.


Exemplu: Vederea Synergy

```
CREATE VIEW Synergy AS
SELECT Likes.drinker, Likes.beer, Sells.bar
FROM Likes, Sells, Frequents
WHERE Likes.drinker = Frequents.drinker
AND Likes.beer = Sells.beer
AND Sells.bar = Frequents.bar;
```

Câte o copie a
fiecărui atribut



Natural join între Likes,
Sells și Frequents



Interpretarea Adăugării într-o Vedere

- Nu se poate adăuga în Synergy --- este o vedere virtuală definită pe trei tabele de bază.
- Se poate în schimb folosi un trigger INSTEAD OF pentru a transforma adăugarea tripletului (drinker, beer, bar) în vederea Synergy, în adăugări în toate cele trei tabele de bază, câte o adăugare pentru fiecare: Likes, Sells și Frequents.
 - Sells.price va trebui completat cu NULL.

Trigerul

```
CREATE TRIGGER ViewTrig
  INSTEAD OF INSERT ON Synergy
  REFERENCING NEW ROW AS n
  FOR EACH ROW
  BEGIN
    INSERT INTO LIKES VALUES(n.drinker, n.beer);
    INSERT INTO SELLS(bar, beer) VALUES(n.bar, n.beer);
    INSERT INTO FREQUENTS VALUES(n.drinker, n.bar);
  END;
```

Vederi Materializate

- **Problemă:** de fiecare dată când una din tabelele de bază este actualizată, vederea materializată este posibil să trebuiască să fie actualizată.
 - Nu ne putem permite să recalculăm vederea la fiecare actualizare.
- **Soluție:** Periodic se reconstruiește vederea materializată, în caz contrar vederea este "out of date".

Exemplu: Listă e-mail

- Lista de e-mail cu studenții unei grupe se presupune o vedere materializată.
- Dacă este introdus un nou student în baza de date, el nu apare în vedere până la împrospătarea vederii.
 - De aceea studentul respectiv nu va primi e-mailuri primite de colegii săi până la împrospătarea vederii.

Exemplu: Data Warehouse

- ❑ Wal-Mart (lanț de magazine) stochează fiecare vânzare la fiecare magazin într-o bază de date.
- ❑ În cursul nopții, vânzările din ziua respectivă sunt folosite pentru a actualiza *data warehouse* = vederi materializate cu vânzările.
- ❑ Datele din data warehouse sunt analizate de analiști ce prezic tendințe și mută bunurile la magazinele unde sunt vânzările cele mai bune.