

# BAZE DE DATE

Implementarea constrângerilor  
Declanșatoare  
Views

Mihaela Elena Breabăn

© FII 2014-2015

# Objective

---

- ▶ Constrângeri de integritate
- ▶ Declanșatoare
- ▶ Views

# Constrângeri de integritate (statice)

## (1)

---

- ▶ **Restricționează stările posibile ale bazei de date**
  - ▶ Pentru a elimina posibilitatea introducerii eronate de valori la inserare
  - ▶ Pentru a satisface corectitudinea la actualizare
  - ▶ Forțează consistența
  - ▶ Transmit sistemului informații utile stocării, procesării interogărilor
- ▶ **Tipuri**
  - ▶ Non-null
  - ▶ Chei
  - ▶ Integritate referențială
  - ▶ Bazate pe atribut și bazate pe tuplu
  - ▶ Aserțiuni generale

# Constrângeri de integritate (2)

---

- ▶ **Declarare**

- ▶ Odată cu schema
- ▶ După crearea schemei

- ▶ **Realizare**

- ▶ Verificare după fiecare modificare
- ▶ Verificare la final de tranzacție

# Constrângeri de integritate peste 1 variabilă

## Implementare

---

**CREATE TABLE** *tabel* (

*a1* tip **not null**, -- acceptă doar valori nenule

*a2* tip **unique**, --cheie candidat formată dintr-un singur atribut

*a3* tip **primary key**, -- cheie primară formată dintr-un singur atribut, implicit {not null, unique}

*a4* tip **references** *tabel2* (*b1*), --cheie străină formată dintr-un singur atribut

*a5* tip **check** (*condiție*) -- condiția e o expresie booleană formulată peste *a5*: (*a5*<11 and *a5*>4), (*a5* between 5 and 10), (*a5* in (5,6,7,8,9,10))...

)

# Constrângeri de integritate peste $n$ variabile

## Implementare

---

**CREATE TABLE** *tabel* (

*a1* tip,

*a2* tip,

*a3* tip,

*a4* tip,

**primary key** (*a1*,*a2*), --cheie primară formată din 2 (sau mai multe)  
atribute

**unique**(*a2*,*a3*), -- cheie candidat formată din 2 (sau mai multe) atribute

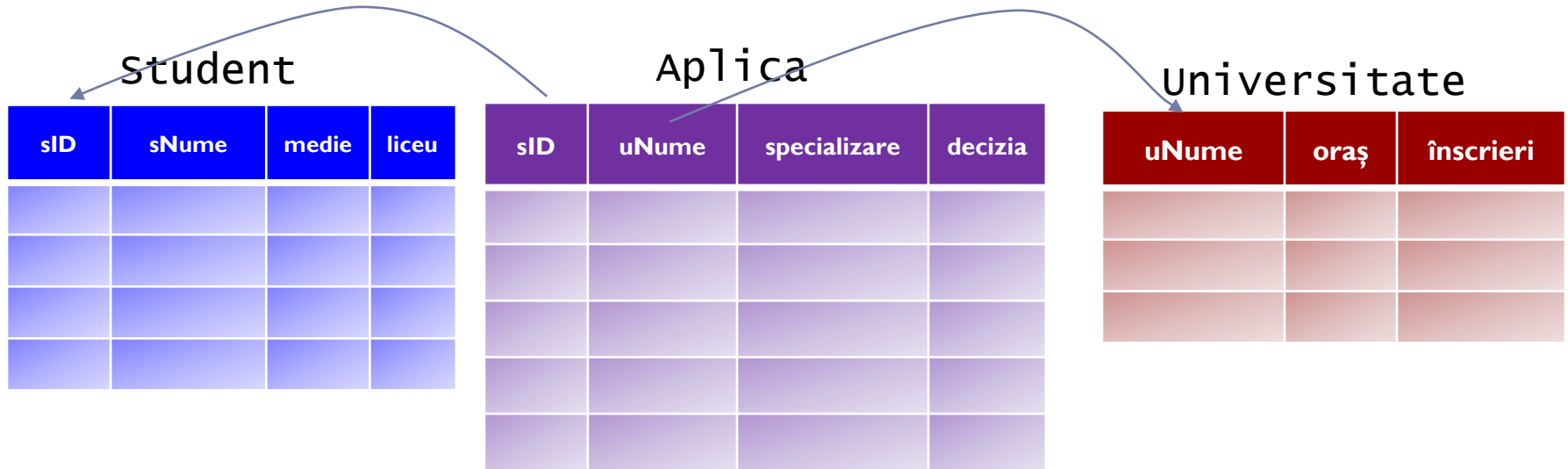
**check** (*condiție*), -- expresie booleană peste variabile declarate  
anterior:  $((a1+a3)/2 \geq 5)$

**foreign key** (*a3*,*a4*) **references** *tabel2*(*b1*,*b2*) -- cheie străină multi-  
atribut

)

# Integritate referențială

## Definiții



- ▶ *Integritate referențială de la R.A la S.B:*
  - ▶ fiecare valoare din coloana A a tabelului R trebuie să apară în coloana B a tabelului S
  - ▶ A se numește cheie străină
  - ▶ B trebuie să fie cheie primară pentru S sau măcar declarat unic
  - ▶ sunt permise chei străine multi-atribut

# Integritate referențială

## Realizare

---

- ▶ Comenzi ce pot genera încălcarea restricțiilor:

- ▶ inserări în R
- ▶ ștergeri în S
- ▶ actualizări pe R.A sau S.B

- ▶ Acțiuni speciale:

- ▶ la ștergere din S:

ON DELETE RESTRICT (implicit) | SET NULL | CASCADE

- ▶ la actualizări pe S.B:

ON UPDATE RESTRICT (implicit) | SET NULL | CASCADE



# Integritate referențială *oul sau găina?*

```
CREATE TABLE chicken (cID INT PRIMARY KEY,  
                        eID INT REFERENCES egg(eID));  
CREATE TABLE egg(eID INT PRIMARY KEY,  
                  cID INT REFERENCES chicken(cID));
```

```
CREATE TABLE chicken(cID INT PRIMARY KEY, eID INT);  
CREATE TABLE egg(eID INT PRIMARY KEY, cID INT);
```

```
ALTER TABLE chicken ADD CONSTRAINT chickenREFegg  
    FOREIGN KEY (eID) REFERENCES egg(eID)  
    DEFERRABLE INITIALLY DEFERRED; -- Oracle  
ALTER TABLE egg ADD CONSTRAINT eggREFchicken  
    FOREIGN KEY (cID) REFERENCES chicken(cID)  
    DEFERRABLE INITIALLY DEFERRED; -- Oracle
```

```
INSERT INTO chicken VALUES(1, 2);  
INSERT INTO egg VALUES(2, 1);  
COMMIT;
```

Cum rezolvați problema inserării dacă  
verificarea constrângerii se efectuează  
imediat după fiecare inserare?  
Dar problema ștergerii tabelelor?

# Aserțiuni

---

create assertion Key

check ((select count(distinct A) from T) =  
      (select count(\*) from T));

create assertion ReferentialIntegrity

check (not exists (select \* from Aplica  
                  where sID not in (select sID from Student)));

create assertion AvgAccept

check (3.0 < (select avg(medie) from Student  
              where sID in  
              (select SID from Aplica where decizie = 'DA')));

# Constrângeri de integritate

## Abateri de la standardul SQL

---

- ▶ Postgres, SQLite, Oracle implementează și validează toate constrângerile anterioare, MySQL permite declararea constrângerilor de tip check dar nu le validează
- ▶ Standardul SQL permite utilizarea de interogări în clauza check însă nici un SGBD nu le suportă
- ▶ Nici un SGBD nu a implementat aserțiunile din standardul SQL, funcționalitatea lor fiind furnizată de declanșatoare

---

...DEMO...  
(fișierul *constrângeri.sql*)

# Declanșatoare (constrangeri dinamice)

---

- ▶ Monitorizează schimbările în baza de date, verifică anumite condiții și inițiază acțiuni
- ▶ Reguli eveniment-condiție-acțiune
  - ▶ Introduc elemente din logica aplicației în SGBD
  - ▶ Forțează constrângeri care nu pot fi exprimate altfel
  - ▶ Sunt expresive
  - ▶ Pot întreprinde acțiuni de reparare
  - ▶ implementarea variază în funcție de SGBD, exemplele de aici urmăresc standardul SQL

# Declanșatoare

## Implementare

---

**Create Trigger** *nume*  
**Before|After|Instead Of** *evenimente*  
**[** *variabile-referențiate* **]**  
**[ For Each Row ]** -- acțiune se execută pt fiecare linie modificată (tip row vs. statement)  
**[ When ( condiție ) ]** -- ca o condiție WHERE din SQL  
*acțiune* -- în standardul SQL e o comandă SQL, în SGBD-uri poate fi bloc procedural

- ▶ *evenimente*:
  - ▶ **INSERT ON** *tabel*
  - ▶ **DELETE ON** *tabel*
  - ▶ **UPDATE [OF a1,a2,...] ON** *tabel*
- ▶ *variabile-referențiate* (după declarare pot fi utilizate în *condiție* și *acțiune*):
  - ▶ **OLD TABLE AS** *var*
  - ▶ **NEW TABLE AS** *var*
  - ▶ **OLD ROW AS** *var* — *pentru ev. DELETE, UPDATE*
  - ▶ **NEW ROW AS** *var* — *pentru ev. INSERT, UPDATE*

} doar pentru triggere de  
tip row

# Declanșatoare

## Exemplu (1)

---

- ▶ integritate referențială de la R.A la S.B cu ștergere în cascadă

**Create Trigger Cascade**  
**After Delete On S**  
**Referencing Old Row As O**  
**For Each Row**  
~~[ fără condiții ]~~  
**Delete From R Where A = O.B**

**Create Trigger Cascade**  
**After Delete On S**  
**Referencing Old Table As OT**  
~~[ For Each Row ]~~  
~~[ fără condiții ]~~  
**Delete From R Where**  
***A in (select B from OT)***

# Declanșatoare

## Capcane

---

- ▶ mai multe declanșatoare activate în același timp: care se execută primul?
- ▶ acțiunea declanșatorului activează alte declanșatoare: înlănțuire sau auto-declanșare ce poate duce la ciclare



# Declanșatoare

## Abateri de la standardul SQL

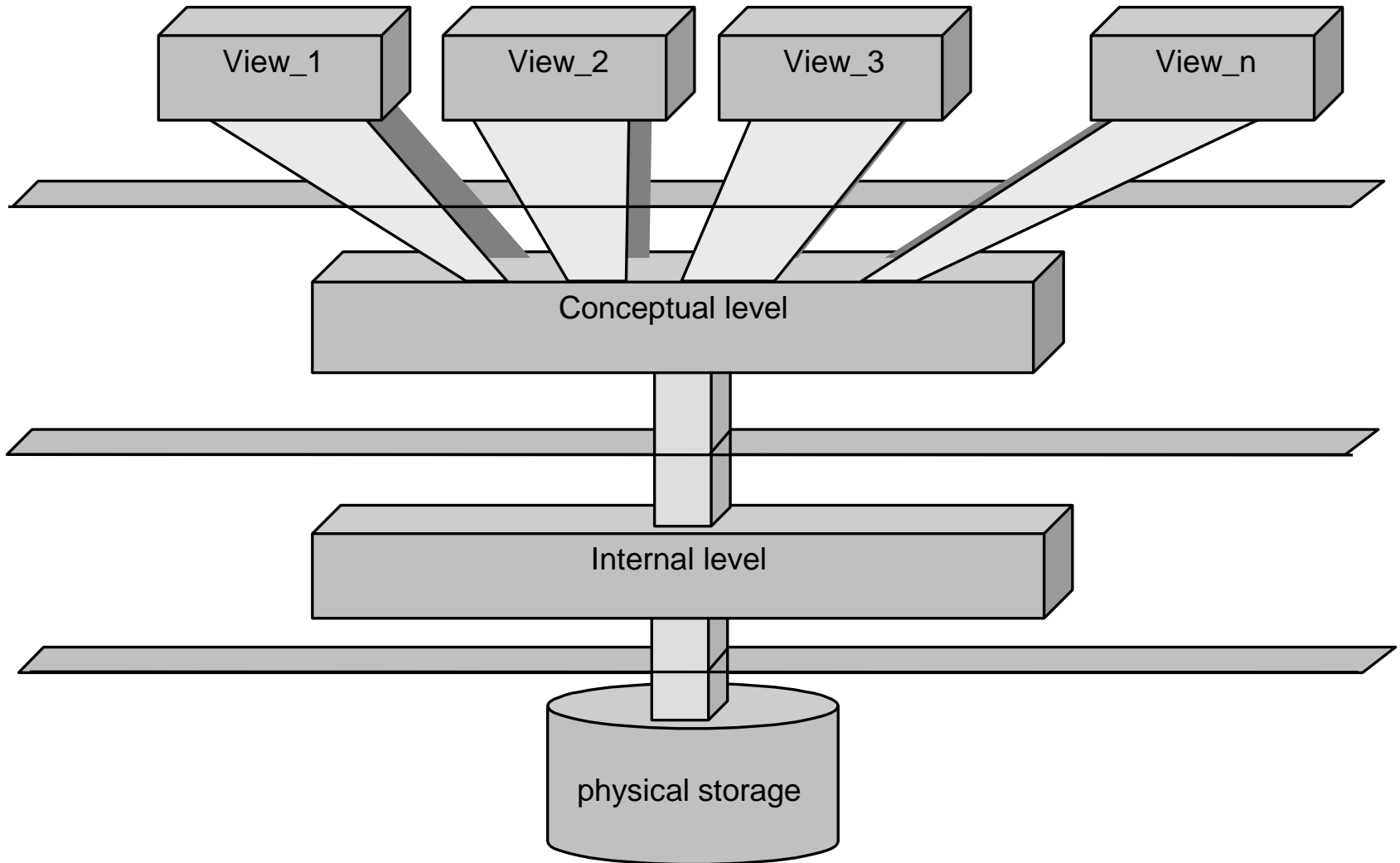
---

- ▶ **Postgres**
  - ▶ cel mai apropiat de standard
  - ▶ implementează row+statement, old/new+row/table
  - ▶ sintaxa suferă abateri de la standard
- ▶ **SQLite**
  - ▶ doar tip row (fără old/new table)
  - ▶ se execută imediat, după modificarea fiecărei linii (abatere comportamentală de la standard)
- ▶ **MySQL**
  - ▶ doar tip row (fără old/new table)
  - ▶ se execută imediat, după modificarea fiecărei linii (abatere comportamentală de la standard)
  - ▶ permite definirea unui singur declanșator / eveniment asociat unui tabel
- ▶ **Oracle**
  - ▶ implementează standardul: row+statement cu modificări ușoare de sintaxă
  - ▶ tipul instead-of e permis numai pt. view-uri
  - ▶ permite inserarea de blocuri procedurale
  - ▶ introduce restricții pentru a evita ciclarea
  - ▶ **aprofundate la laborator**

---

...DEMO...  
(fișierul *declansatoare.sql*)

# View-uri



# Motivație

---

- ▶ ascunderea unor date față de unii utilizatori
  - ▶ ușurarea formulării unor interogări
  - ▶ acces modular la baza de date
- 
- ▶ aplicațiile reale tind să utilizeze foarte multe view-uri

# Definire și utilizare

---

- ▶ Un view este în esență o interogare stocată formulată peste tabele sau alte view-uri
- ▶ Schema view-ului este cea a rezultatului interogării
- ▶ Conceptual, un view este interogat la fel ca orice tabel
- ▶ În realitate, interogarea unui view este rescrisă prin inserarea interogării ce definește view-ul urmată de un proces de optimizare specific fiecărui SGBD
- ▶ Sintaxa

**Create View** *numeView* [*a1,a2,...*] **As** <*frază\_select*>

# Modificarea view-urilor

---

- ▶ View-urile sunt în general utilizate doar în interogări însă pentru utilizatorii externi ele sunt tabele: trebuie să poată suporta comenzi de manipulare/modificare a datelor
- ▶ Soluția: modificări asupra view-ului trebuie să fie rescrise în comenzi de modificare a datelor în tabelele de bază
  - ▶ de obicei este posibil
  - ▶ uneori există mai multe variante
- ▶ Exemplu
  - ▶  $R(A,B), V(A)=R[A]$ , Insert into V values(3)
  - ▶  $R(N), V(A)=avg(N)$ , update V set A=7

# Modificarea view-urilor

## Abordări

---

- ▶ creatorul view-ului rescrie toate comenzile de modificare posibile cu ajutorul declanșatorului de tip **INSTEAD OF**
  - ▶ acoperă toate cazurile
  - ▶ garantează corectitudinea?
- ▶ standardul SQL prevede existența de view-uri inerent actualizabile (updatable views) dacă:
  - ▶ view-ul e creat cu comanda select fără clauza **DISTINCT** pe o singură tabelă T
  - ▶ attributele din T care nu fac parte din definiția view-ului pot fi **NULL** sau iau valoare default
  - ▶ subinterogările nu fac referire la T
  - ▶ nu există clauza **GROUP BY** sau altă formă de agregare

# View-uri materializate

---

**Create Materialized View** *V* [*a1,a2,...*] **As** <*frază\_select*>

- ▶ are loc crearea unui nou tabel *V* cu schema dată de rezultatul interogării
- ▶ tuplele rezultat al interogării sunt inserate în *V*
- ▶ interogările asupra lui *V* se execută ca pe orice alt tabel
- ▶ Avantaje:
  - ▶ specifice view-urilor virtuale + crește viteza interogărilor
- ▶ Dezavantaje:
  - ▶ *V* poate avea dimensiuni foarte mari
  - ▶ orice modificare asupra tabelelor de bază necesită refacerea lui *V*
  - ▶ problema modificării tabelelor de bază la modificarea view-ului rămâne



# Cum alegem ce materializăm

---

- ▶ dimensiunea datelor
- ▶ complexitatea interogării
- ▶ numărul de interogări asupra view-ului
- ▶ numărul de modificări asupra tabelelor de bază ce afectează view-ul și posibilitatea actualizării incrementale a view-ului
- ▶ punem în balanță timpul necesar execuției interogărilor și timpul necesar actualizării view-ului

---

...DEMO...  
(fișierul *views.sql*)

# Bibliografie

---

- ▶ Hector Garcia-Molina, Jeff Ullman, Jennifer Widom: *Database Systems: The Complete Book (2nd edition)*, Prentice Hall; (June 15, 2008)
- ▶ Oracle:
  - ▶ [http://docs.oracle.com/cd/B28359\\_01/server.111/b28310/general005.htm](http://docs.oracle.com/cd/B28359_01/server.111/b28310/general005.htm)
  - ▶ <http://www.oracle-base.com/articles/9i/MutatingTableExceptions.php>
  - ▶ [http://www.dba-oracle.com/t\\_avoiding\\_mutating\\_table\\_error.htm](http://www.dba-oracle.com/t_avoiding_mutating_table_error.htm)