

# Modifica all'istanza di una relazione / 03-05

INSERT inserimento

UPDATE aggiornamento

DELETE cancellazione

• INSERT INTO R  $\{C_1, C_2, \dots, C_n\}$  VALUES  $(v_1, v_2, \dots, v_n)$  ( $\begin{smallmatrix} \text{tupolo} \\ \text{valore} \end{smallmatrix}$ )

↓

se voglio specificare le colonne

dove mettere i valori

(e soprattutto se voglio stabilire un ordine di inserimento)

→ l'ordine attributi - valori deve coincidere!



se non specifico l'ordine è quello

della CREATE TABLE

(> specifico le altre  
colonne)

se non specifico la colonna / valore viene messo il valore di default = null

se non avesse default. Si genera errore se c'è vincolo NOT NULL

• INSERT INTO R  $\{C_1, C_2, \dots, C_n\}$  Q

↓  
query per inserire più tuple

es. inserire i noleggi del cliente 3365 e a tutti i video contenenti film

di Johnnie Salvadoras che non aveva ancora noleggiato

Q: SELECT colloc, 6635 → costante, necessaria per far riferimento al cliente  
FROM Video  
(dominio compatibile con l'inserimento in Noleggi)

WHERE regista = 'GS' AND colloc NOT IN

(SELECT colloc FROM Noleggi WHERE codcli = 6635)

↳ video noleggiati da GS

necessario selezionare perciò  $Q$  restituiscce è eff.

INSERT INTO Noleggi (colloc, codcli)  $Q$

con l'ipotesi che DataNol di DEFAULT prende Current Date

Se in  $Q$  una tuple non soddisfa i vincoli di integrità, il comando non viene eseguito (e non solo perodiment, non si capirebbe ottimamente cosa è stato inserito)

- DELETE FROM R [ $<\text{colios}>$ ] [WHERE F]; si applica a una singola relazione
  - (aggiornamento DDL) pur prima delle cancellazioni vengono cancellate tutte le tuple se non specifico (l'intera istanza non R rimane)
  - DROP cancella anche le istanze
  - DDL

es. Cancellare i clienti che non hanno effettuato noleggi nell'ultimo anno

DELETE FROM Clienti

WHERE codcli NOT IN

(SELECT codcli FROM Noleggi WHERE DataNol > Current Date )

• UPDATE R [ $<\text{colios}>$ ]

SET  $C_1 = \{c_1 | \text{null}\}, \dots, C_n = \{c_n | \text{null}\}$

[WHERE F] → assegna e, come nuovo valore di  $C_i$  oppure null

↳ se non presente, vengono aggiornate tutte

es 2o doppioire lo valutazioni dei film

UPDATE Film  $\rightarrow$  valore prec.

SET valutaz = valutaz \* 2

es il cliente GG35 restituisce tutti i video che ha in noleggio e segnala che il noleggio e' iniziato ieri

UPDATE Noleggio

SET dataRest = Current Date,  
dataNol = (Current Date - 1 day)

WHERE codCli = GG35 AND DataRest IS NULL  
 $\rightarrow$  val prec.

es aumentare del 10% la valutaz. dei film horror usciti dopo il 1/3/93 che sono stati noleggiati da almeno due clienti

UPDATE Film

SET valutaz = valutaz + 1,1

WHERE genere = 'horror' AND anno > 1993

AND (titolo, regista) IN

(SELECT titolo, regista

FROM Noleggio NATURAL JOIN Video

GROUP BY titolo, regista

HAVING COUNT(DISTINCT codCli) >= 2)

Assegnare ad ogni film che è stato noleggiato almeno una volta nell'ultimo mese, una valutazione pari al 110% della valutazione media dei film dello stesso anno e genere

UPDATE Film X

```
SET valutaz = ( SELECT 1.1 * AVG(valutaz)
                  FROM Film
                 WHERE anno = X.anno AND genere = X.genere)
WHERE EXISTS (
    SELECT * (titolo, regista)
      FROM Noleggio NATURAL JOIN Video
     WHERE dataNol > CURRENT_DATE - 1 MONTH
       AND titolo = X.titolo AND regista = X.regista
  WHERE (titolo, regista) IN (...)
```

} → *funo è inserito a  
R in un certo momento  
(se si aggiorna nel  
frattempo le sostituzio-  
nioni restituiscono  
risultati differenti)*

me quindi ogni volta AVG(valutaz) è diversa!

Infatti le modifiche sono eseguite in SQL in modo SET-ORIENTED ovvero WHERE e SET vengono fatti una volta sola sull'istanza postata originalmente, <sup>Mentre</sup> delle modifiche (anche in caso di seth-inter-correlate).

Quindi AVG(valutaz) è lo stesso durante il comando UPDATE

## Vincoli di integrità

SQL permette di specificare dei vincoli che i dati devono verificare

- primary key e unique
- not null
- foreign key

Vi sono due tipi:

- statici: verificabili su un singolo stato della base di dati
- dinamici: si verificano considerando due istanze diverse  
(es. si considerano valori memorizzati nel DB rispetto a quelli nuovi → dato inserito > dato presente)  
→ si usano i **trigger** ( $\rightarrow$  solo progetto, no esami)

## Vincoli statici

Su singola relazione		Su relazioni multiple	
Su singola tupla	Su tuple multiple	un video non può essere noleggiato prima che il film che lo contiene sia uscito	
Su singolo attributo	Su attributi multipli	vincoli di aggregazione	vincoli di dipendenza funzionale
NOT NULL Valutazione tra 0 e 5	Data_restituzione > Data_noleggio	non ci sono più di 3 noleggi attivi per lo stesso cliente	il nome di un cliente è determinato dal suo codice

specificabili con

**CHECK**

o singolo column

o relazione

specificabili con

- **ASSEZIONI SQL** (no PostgreSQL)

- **TRIGGER**

dentro lo CREATE TABLE film {

valore decimal (3,2)

CHECK (valore BETWEEN 0,00 AND 5,00),

...

- puoi includere sette-interrogazioni (non le vediamo)

- condizioni puoi esser incluse in un WHERE

I vincoli vengono verificati doppio per tupla  
Un vincolo si ritiene non soddisfatto se la condizione è falsa,  
se TRUE o UNKNOWN <sup>DATA DA CONDIZIONI CON NULLI</sup> non si ritiene "non soddisfatto", quindi non  
verifica il vincolo

I vincoli vengono verificati ad ogni inserimento / modifica della tabella  
se tutti coincidono gli attributi del class

una CHECK (dato NOT IN Q)

alcuni DBMS permettono la verifica  
perdendo il vincolo può essere violato in  
caso di modifica

} Notaggi Client → se modifica client,  
il check dell'oltra  
tabella non viene verificato  
(PostgreSQL)

• CONSTRAINT < nomeC > CHECK < condizione >  
posso dare dei nomi ai vincoli, così è più facile gestirli;

es. CONSTRAINT Pkey PRIMARY KEY  
" TOC CHECK (tipo IN ('d','v'))  
" Run NOT NULL  
" FK FOREIGN KEY (...) REFERENCES ...

posso aggiungerli, senza ricostituire le tabella.

ALTER TABLE Video DROP CONSTRAINT TOC;  
" " ADD CONSTRAINT TOC CHECK  
(tipo IN ('d','v','x'))