18-11-2022

Esercitazione itinere (dello scorso anno

Si consideri il seguente schema relazionale:

Persona(cf, nome, cognome, indirizzo)
Prodotto(id, tipo, descrizione, nome)
Acquisto prodotto(idprodotto, cfpersona)
TipiProdotti(id,nome)

1. Identificare le chiavi primarie ed esterne dello schema [0 corretta, -1 errata]
2. Rispondere alle seguenti query in algebra relazionale:
   a. Trovare le persone che non hanno effettuato acquisti dei prodotti di tipo "fai da te" [3 punti];
   b. Trovare i tipi dei prodotti che sono stati acquistati da tutti i clienti [4 punti];

Rispondere alle seguenti query in SQL:

   c. Trovare per ogni tipologia il prodotto con il massimo numero di acquisti [4 punti];

   d. Creare una asserzione che non consenta di inserire un acquisto di un prodotto di tipo "informatica" se non è stato acquistato già un prodotto di tipo "Cucina" [4 punti].

2a)Trovare le persone che non hanno effettuato acquisti dei prodotti di tipo "fai da te"

Persone che hanno acquistato prodotti fai da te:
$R_1 = \pi_{cfpersona}(\sigma_{TipiProdotti.nome='faidate'}(ProdottoJOIN_{tipo=TipiProdotto.id}TipiProdotti)JOIN_{Prodotto.id=}$
$\pi_{cf}(Persona)/\delta_{cfpersona \to cf}(R_1)$ //prendo tutte le persone e tolgo quelle che hanno acquistato 'fai da te'. ridenomino perchè gli attributi devono avere lo stesso nome in algebra.

2b) Trovare i tipi dei prodotti che sono stati acquistati da tutti i clienti (con la divisione):

$R_1 = \pi_{cfpersona,tipo}(AcquistoProdottoJOIN_{idprodotto=Prodotto.id}Prodotto)$ Per ogni persona che tipo di prodotto hanno acquistato
devo trovare i clienti-> i nomi degli attributi devono corrispondere. la divisione va a prendere per ogni tipo va a vedere se quel tipo è associato con tutti i codice fiscale delle persone. se sì, lo restituisco, altrimenti no.
$R_2 = \delta_{cfpersona->cf}(\pi_{cf}(Persona))$
Risultato = $R_1 \div R_2$

2c)Trovare per ogni tipologia il prodotto con il massimo numero di acquisti

```
// n acquisti per ogni prodotto

CREATE VIEW NumeroAcquisti AS ---view avrà 3 colonne: id prodotto, tipo
prodotto, numacquisti
        SELECT p.id AS idp, p.tipo, COUNT(*) AS numacq
        FROM Acquistoprodotto ap, Prodotto p
        WHERE ap.idprodotto = p.id
        GROUP BY p.id, p.tipo
---------------------------- possibile soluzione 1
SELECT *
FROM NumeroAcquisti na
WHERE na.numacq = (SELECT MAX(numacq)
                                FROM NumeroAcquisti na1
                                WHERE na1.tipo = na.tipo) -- per quel
specifico tipo, calcola il max del num acquisti. Se il n accquisti è il valore
massimo, allora viene eseguito. è il massimo solo in quel particolare tipo.
---------------------------- possibile soluzione 2
SELECT *
FROM NumeroAcquisti na
WHERE na.numacq >= ALL(SELECT numacq
                                FROM NumeroAcquisti na1
                                WHERE na1.tipo = na.tipo)
------------------- si poteva anche creare un'altra vista
CREATE VIEW NumeroMassimoAcquisti AS (SELECT tipo, MAX(numacq) as massimo
                                                                        FROM
NumeroAcquisti
                                                                        GROUP
BY tipo) -- per ogni tipo indico max num acquisti.
SELECT *
FROM NumeroAcquisti na, NumeroMassimoAcquisti nm
WHERE na.tipo = nm.tipo AND na.numacq = nm.massimo --numero max acquisti e si fa
join

-------------------- altra possibile soluzione (diretta e più complessa)
SELECT p.id AS idp, p.tipo, COUNT(*) AS numacq -- calcolo il conteggio
FROM  Acquistoprodotto ap, Prodotto p
WHERE ap.idprodotto = p.id
GROUP BY p.id, p.tipo
HAVING COUNT(*) >= ALL (SELECT COUNT(*)
                                                FROM Acquistoprodotto ap1,
Prodotto p1
                                                WHERE ap1.idprodotto = p1.id AND
p.1tipo = p.tipo
                                                GROUP BY p1.id)
```

d) Creare una asserzione che non consenta di inserire un acquisto di un prodotto di tipo “informatica” se non è stato acquistato già un prodotto di tipo “Cucina”

Asserzione

```
---CREATE ASSERTION LimitaAcquistiInformatica
---CHECK (SQL_Condition)
--- non esiste una persona che ha acuqistato un prodotto "informatica" e non ha acquistato un prodotto "cucina" (cioè "non ha acquistato un prodotto" allora quella persona non ha un record di tipo "informatica")
CREATE ASSERTION LimitaAcquistiInformatica
CHECK NOT EXIST (
        SELECT *
        FROM Persona p
        WHERE EXIST (SELECT *
                                FROM AcquistiProdotto ap, Prodotto pr,
TipiProdotti tp
                                WHERE ap.idprodotto = pr.id AND pr.tipo =
tp.tipo
                                AND tp.nome = 'Informatica' AND ap.cfpersona =
p.cf)
        AND NOT EXIST (SELECT *
                                FROM AcquistiProdotto ap, Prodotto pr,
TipiProdotti tp
                                WHERE ap.idprodotto = pr.id AND pr.tipo =
tp.tipo
                                AND tp.nome = 'Cucina' AND ap.cfpersona = p.cf)
        ) --controlla che non esiste una persona p che ha acquistato un prodotto ti tipo informatica e non ha acquistato un prodotto di tipo cucina (per una persona specifica)
-------------------------- possibile alternativa

CREATE ASSERTION LimitaAcquistiInformatica
CHECK NOT EXIST (
         SELECT *
         FROM AcquistiProdotto ap, Prodotto pr, TipiProdotti tp
         WHERE ap.idprodotto = pr.id AND pr.tipo = tp.tipo
         AND tp.nome = 'Informatica'
         AND NOT EXIST ( SELECT *
                                        FROM AcquistiProdotto ap, Prodotto pr,
TipiProdotti tp
                                        WHERE ap.idprodotto = pr.id AND pr.tipo
= tp.tipo
                                        AND tp.nome = 'Cucina'
                                        AND ap.cfpersona = p.cf)
```

Si consideri il seguente schema relativo alla gestione dei rifiuti di una città:

Cittadino(CF, Nome, Cognome, Indirizzo, CAP, numeroComponentiFamiliari)

Zona(CAP, numeroAbitanti)

CalendarioSettimanale(giorno_Raccolta, CAP, tipologia)

RifiutiCittadino(CF, giorno, peso, data, esito)

L'attributo giorno_raccolta assume i valori {lun, mar, mer, gio, ven, sab}. L'attributo Esito (Positivo/Negativo) indica se il cittadino ha depositato i rifiuti della tipologia corretta nel giorno previsto.

1. Identificare le chiavi primarie ed esterne dello schema [0 corretta, -1 errata]

2. Rispondere alle seguenti query in Algebra Relazionale:
   a. Per ogni CAP trovare i cittadini che non hanno gettato mai i rifiuti in modo corretto [3 punti];
   b. Trovare i CAP che prevedono il ritiro di tutte le tipologie di rifiuti [3 punti];

3. Rispondere alle seguenti query in SQL:
   a. Trovare per ogni tipologia di rifiuti i CAP che hanno avuto il peso complessivo di rifiuti massimo [3 punti];
   b. Per ogni cittadino stampare un report che indichi il peso medio dei rifiuti per ogni giorno della settimana [5 punti].

2a) Sempre un esito negativo, non posso selezionare dove esito = negativo. Prendo tutti quelli che hanno esito positivo e li tolgo dai RifiutiCittadino

$R_1 = \pi_{CF,CAP}(\sigma_{esito='positivo'}(RifiutiCittadino)JOINCittadino)$ //per ogni cap si ha ogni cittadino che ha conferito in modo positivo

tutti i cittadini che hanno conferito rifiuti - quelli che hanno conferito positivamente
$\pi_{CF,CAP}(RifiutiCittadinoJOINCittadino)/R_1$

2b) CalendarioSettimanale = CS

$R_1 = \pi_{tipologia}(CS)$ -- tutti i tipi di rifuti

$R_2 = \pi_{CAP,tipologia}(CS)$ -- per ogni cap, si indica la tipologia di rifiuto

$R_2 DIVISO R_1$

3a)

```sql
CREATE VIEW PesoPerCAPeTipo AS
        SELECT cs.CAP, cs.tipologia, SUM(cs.peso) AS peso
        FROM CalendarioSettimaale cs, RifiutiCittadino rc, Cittadino c --tutti
rifiuti per cap e giorno raccolta
        WHERE c.CF = rc.CF -- lo stesso cittadino
        AND cs.giorno_Raccolta = rc.giorno --giorno della raccolta
        AND c.CAP=cs.CAP
```

```
        GROUP BY cs.CAP, cs.tipologia

SELECT *
FROM PesoPerCAPeTipo pt
WHERE peso >= ALL (SELECT peso
                   FROM PesoPerCAPeTipo pt1
                   WHERE pt1.tipologia = pt.tipologia)
```

3b)

```
SELECT CF,giorno,AVG(peso)
FROM RifiutiCittadino
GROUP BY CF,giorno
```

Altro esercizio sullo stesso DB

Creare un’asserzione che non consenta di inserire un record in rifiuti cittadino se non è previsto dal corretto calendario settimanale

non esiste un record in rifiuto cittadino tale che l'esito è negativo

```
CREATE ASSERTION NoRifiutiSbagliati
CHECK NOT EXIST (SELECT *
                 FROM RifiutiCittadino
                 WHERE Esito='Negativo')
```