

Si consideri il seguente schema relazionale:

Libro(titolo, autore, data_prima_pubblicazione)
 Autore(id, nome, cognome)
 Genere(id, nome, descrizione)
 GenereLibro(idg, titolo)
 Utente(cf, nome, cognome)
 LibroLetto(cf, titolo, data, commenti)

2) Rispondere alle seguenti query in algebra relazionale:

a. Trovare gli utenti che hanno letto tutti i libri di "Stephen King" [4 punti]

$$R1 = \prod_{\text{titolo}} \left(\sigma_{\substack{\text{name} = "Stephen" \\ \text{cognome} = "King"} } (Libro) \bowtie_{\text{Libro.autore} = \text{Autore.id}} \text{Autore} \right)$$

$$R2 = \prod_{\text{cf}, \text{titolo}} (\text{LibroLetto}) \div R1$$

b. Trovare gli autori che non hanno mai scritto "thriller" o "horror" [4 punti]

$$R1 = \left(\text{Libro} \bowtie_{\text{Generi.Libro}} \text{Generi.Libro} \bowtie_{\text{Generi.Libro.idg} = \text{Generi.id}} \text{Generi} \right)$$

$$R2 = \sigma_{\text{Generi.name} = "thriller" \vee \text{Generi.name} = "horror"} (R1)$$

$$R3 = \sum_{\text{autore} \rightarrow \text{id}} (\prod_{\text{autore}} (R2))$$

$$RL = \prod_{\text{id}} (\text{Autore}) - R3$$

3) Rispondere alle seguenti query in SQL

Libro(titolo, autore, data_prima_pubblicazione)
 Autore(id, nome, cognome)
 Genere(id, nome, descrizione)
 GenereLibro(idg, titolo)
 Utente(cf, nome, cognome)
 LibroLetto(cf, titolo, data, commenti)

utente
F:]

a. Per ogni utente contare quanti libri di ogni genere ha letto [4 punti]

SELECT cf, idg, COUNT(DISTINCT titolo) AS conteggio
 FROM LibroLetto JOIN Libro ON Libro.titolo = LibroLetto.titolo
 JOIN Generi.Libro ON Generi.Libro.titolo = Libro.titolo
 GROUP BY cf, idg

b. Trovare i libri che sono stati letti da tutti gli utenti [4 punti]

SELECT * FROM Libro l
 WHERE NOT EXISTS (SELECT * FROM Utente u
 WHERE NOT EXISTS (SELECT * FROM LibroLetto ll
 WHERE ll.cf = u.cf AND
 ll.titolo = l.titolo))

4) Si supponga di avere la relazione GenerePreferito(cf,genere) che contiene i generi di libri preferiti da ogni utente. Implementare un trigger che ogni qual volta viene inserito un record nella relazione LibroLetto si verifica per il particolare utente se sono stati letti almeno 10 libri del genere appena inserito, in questo caso viene inserito, se non esiste già, il record nella relazione GenerePreferito. [4 punti]

GeneroPreferito. [4 punti]

CREATE TRIGGER T1 AFTER INSERT ON LibroLetto FOR EACH ROW
 WHEN 10 <= (SELECT COUNT(DISTINCT titolo)
 FROM LibroLetto
 WHERE cf = NEW.cf AND
 titolo IN (SELECT g1.titolo
 FROM GeneroLibro g1, GeneroLibro g2
 WHERE g1.idg = g2.idg AND
 g2.titolo = NEW.titolo))

DECLARE x INT DEFAULT 0;
 DECLARE cf INT;
 BEGIN
 SELECT COUNT(*)
 WHERE cf = NEW.cf AND genero IN (SELECT idg
 FROM GeneroLibro
 WHERE titolo = NEW.titolo) INTO x
 FROM GeneroPreferito
 WHERE cf = NEW.cf AND genero = (SELECT idg
 FROM GeneroLibro
 WHERE titolo = NEW.titolo);
 IF (x = 0) THEN
 INSERT INTO GeneroPreferito(cf, genero)
 VALUES (NEW.cf, 1);
 ELSE
 SELECT idg INTO idg FROM GeneroLibro
 WHERE titolo = NEW.titolo;
 END IF;
 END

MySQL:
BEGIN
SELECT NEW.ef INTO Genere Preferito
FROM t0g WHERE titol = NEW.titol
INSERT IGNORE NEW.ef INTO Genere Libro
END

- 1) Si consideri il seguente schema relazionale relativo alla gestione di un centro d'assistenza:

CLIENTE (cf, nome, cognome)

TECNICO (codice, nome, cognome, costo_orario)

ELETTRODOMESTICO (id, marca, tipo, modello)

ACQUISTO (cliente, elettrodomestico, data_acquisto, durata_garanzia)

CHIAMATEASSISTENZA (cf, elettrodomestico, dataora_chiamata, guasto, chiusa, ore_lavorate, costo_ricambi, costo_totale)

ASSEGNAZIONE (cf, elettrodomestico, dataora_chiamata, tecnico)

RICAMBIO (cf, elettrodomestico, dataora_chiamata, ricambio, costo)

Si consideri che la durata della garanzia è espressa in mesi, il costo dei ricambi è zero se l'elettrodomestico è in garanzia, e che una chiamata può essere assegnata ad un solo tecnico.

1. Identificare le chiavi primarie ed esterne dello schema [0 punti se corretta, -1 se sbagliata o senza risposta].

2. Scrivere in algebra relazionale le seguenti query:

1. Trovare tutti i clienti che hanno avuto almeno una chiamata con costo superiore a 500€, indicando nome, cognome, data della prima chiamata e data di acquisto dell'elettrodomestico [2 punti].

$$R1 = \pi_{Costo_totale > 500} (CHIAMATEASSISTENZA)$$

$$R2 = R1$$

$$R3 = \pi_{\begin{array}{l} R1 \\ R1.eff = R2.cf \end{array}} \left(R1 \Delta_{R1.eff = R2.cf} n R1.dataora_chiamata > R2.dataora_chiamata \right)$$

$$R4 = \left(\pi_{\begin{array}{l} cf, dataora_chiamata \\ R1 \end{array}} (R1) - R3 \right) \Delta_{\begin{array}{l} Cliente \\ Cliente.eff = Acquisto.chiavi \\ R1.eff = Acquisto.chiavi \end{array}} \left(\begin{array}{l} Acquisto \\ R1.elettrodomestico = Acquisto.elettrodomestico \end{array} \right)$$

$$\pi_{\begin{array}{l} nome, cognome, data_acquisto, dataora_chiamata \\ \dots \end{array}} (R4)$$

1. Trovare tutti i clienti che hanno fatto chiamate di assistenza per tutte le marche di elettrodomestici [3 punti].

$$R1 = \pi_{\begin{array}{l} cf \\ marca \end{array}} (CHIAMATEASSISTENZA \Delta_{elettrodomestico = id} ELETTRODOMESTICO)$$

$$R2 = \pi_{\begin{array}{l} marca \end{array}} (ELETTRODOMESTICO)$$

$$R1 \div R2$$

3. Scrivere in SQL le seguenti query:

1. Per ogni pezzo di ricambio indicare quante volte è stato usato per riparazioni fuori garanzia [2 punti].

```
SELECT ricambio, COUNT(*) AS conteggio
FROM RICAMBIO
WHERE costo > 0
GROUP BY ricambio
```

1. Trovare la tipologia di elettrodomestico che ha richiesto il maggior numero di ore di riparazione [3 punti]

```
CREATE VIEW OreAssistenza AS
SELECT elettrodomestico, SUM(ore_lavorate) AS ore
FROM CHIAMATEASSISTENZA GROUP BY elettrodomestico;

SELECT DISTINCT tipo
FROM OreAssistenza JOIN ELETTRODOMESTICO ON elettrodomestico = id
GROUP BY tipo
HAVING SUM(ore) = ALL (SELECT SUM(ore)
FROM OreAssistenza JOIN ELETTRODOMESTICO
ON elettrodomestico = id
GROUP BY tipo)
```

```
CREATE VIEW OreAssistenza AS
SELECT tipo, SUM(ore_lavorate) AS ore
FROM CHIAMATEASSISTENZA JOIN ELETTRODOMESTICO
ON elettrodomestico = id
```

```

SELECT tipo , sum(ore) AS ore_elettrdomestico
FROM CHIASTE ASSISTENZA JOIN ELETTRODOMESTICO ON elettrdomestico = id
GROUP BY tipo;

```

```

SELECT DISTINCT tipo

```

```

FROM OraAssistenza
WHERE ore = (SELECT MAX(ore) FROM OraAssistenza);

```

4. Implementare un trigger che, quando una chiamata di assistenza è chiusa, calcoli il costo dei ricambi e il costo totale.
 Ricordarsi che il costo dei ricambi è zero se l'elettrodomestico è in garanzia. Il costo totale è dato dalla somma del costo dei ricambi e del costo del tecnico.
 Il costo del tecnico è ottenuto moltiplicando le ore lavorate per il costo orario [4 punti].

```

CREATE TRIGGER T1 AFTER UPDATE OF chiusa ON CHIAMATE ASSISTENZA
FOR EACH ROW
WHEN new.chiusa = 1
DECLARE garanzia INT DEFAULT 0;
DECLARE costo_ricambi FLOAT;
DECLARE costo_tecnico FLOAT;
BEGIN
    SELECT (costo_orario * new.ore_lavorate) INTO costo_tecnico
    FROM TECNICO JOIN ASSEGNAZIONE ON tecnico_id = tecnico_id
    WHERE cf = new.cf AND elettrodomestico = new.elettrodomestico AND
        data_ora_chiamata = new.data_ora_chiamata;
    SELECT (new.data_ora_chiamata - (data_acquisto + durata_garanzia)) INTO garanzia
    SELECT (new.acquisto * cf) INTO costo_ricambi
    WHERE cliente = new.cf AND
        elettrodomestico = new.elettrodomestico;
    IF (garanzia = 0) THEN
        SET costo_ricambi = 0;
    ELSE
        SELECT sum(costo) INTO costo_ricambi
        FROM RICAMBI
        WHERE cf = new.cf AND elettrodomestico = new.elettrodomestico AND
            data_ora_chiamata = new.data_ora_chiamata;
    END IF;
    UPDATE CHIAMATE ASSISTENZA
    SET costo_totale = costo_ricambi + costo_tecnico
    WHERE cf = new.cf AND elettrodomestico = new.elettrodomestico AND
        data_ora_chiamata = new.data_ora_chiamata;
END

```

Si consideri il seguente db per la gestione di un campeggio:

```
Piazzola(id, dimensione, prezzo_giornaliero_adulti, prezzo_giornaliero_bambini,
idoneac)
Servizi(id, nome, descrizione, prezzo_giornaliero)
ServiziPiazzola(idp, ids)
Prenotazione(id, data_arrivo, data_partenza, cfcliente, idpiazz, numero_adulti,
numero_bambini, numero_notti, tipo)
ServiziPrenotati(idprenotazione, idp, ids)
Cliente(cf, nome, cognome, tel)
```

L'attributo idoneac della relazione Piazzola è un campo booleano che indica se la piazzola e' idonea ad ospitare camper.
L'attributo tipo della relazione Prenotazione assume i valori {Tenda, Camper}.

b) Rispondere alle seguenti query in SQL

Trovare le piazzole che hanno ospitato tutti i clienti [4 punti]

```
SELECT * FROM Piazzola p
WHERE NOT EXISTS (SELECT *
                   FROM Cliente c
                   WHERE NOT EXISTS (
                       SELECT * FROM Prenotazione
                       WHERE cfcliente = c.cf AND
                             idpiazz = p.id))
```

Per ogni prenotazione determinare il costo finale e il numero di servizi inclusi, e il numero di persone ospitate [5 punti]

```
SELECT p.id, (numero_adulti + numero_bambini) AS persone,
       COUNT(*) AS num_servizi
  FROM Prenotazione p, ServiziPrenotati sp, Servizi s, Piazzola pz
 WHERE sp.idprenotazione = p.id AND s.id = sp.ids AND
       pz.id = p.idpiazz
 GROUP BY p.id
 (( (pz.prezzo_giornaliero_adulti * p.numero_adulti) + (pz.prezzo_giornaliero_bambini *
 p.numero_bambini)) * numero_notti) + SUM(sp.prezzo_giornaliero *
 numero_notti) AS costo_finale
```

a) Rispondere alle seguenti query in algebra relazionale:

Trovare le piazzole, visualizzando l'id, che hanno tutti i servizi [4 punti]

$$\text{ServiziPiazzola} \vdash \delta_{id \rightarrow id} (\Pi_{id}(\text{Servizi}))$$

Trovare le prenotazioni, effettuate nel mese di agosto 2020, di piazzole che non hanno richiesto servizi aggiuntivi [4 punti]

$$R1 = \sigma_{data_arrivo \geq '1/8/2020'} \cap \sigma_{data_arrivo \leq '31/8/2020'} (\text{Prenotazione})$$

$$R2 = \pi_{id\text{prenotazione}}(\text{ServiziPrenotati})$$

$$\Pi_{id}(R1) - \delta_{id\text{prenotazione} \rightarrow id}(R2)$$

Si consideri il seguente schema relazionale relativo alla gestione dei benefit e dei gadget assegnati agli impiegati di una azienda.

IMPIEGATO (matricola, nome, cognome, dipartimento)
DIPARTIMENTO (id, nome, sede)
BENEFIT (id, nome, valore)
GADGET (id, nome, tipo, valore)
BENEFITASSEGNATO (matricola, id, data) BA
GADGETASSEGNATO (matricola, id, data) BA

- a. Identificare le chiavi primarie ed esterne dello schema;

- b. Rispondere alle seguenti query in algebra relazionale:

1. Trovare gli impiegati che nel 2018 non hanno avuto benefit ma hanno avuto gadget;

$$R1 = \Pi_{\text{matricole}} (\text{IMPIEGATO}) - \Pi_{\text{matricole}} (\alpha_{\text{data} \geq '11/11/2018' \wedge \text{data} \leq '31/12/2018'} (\text{BA}))$$

$$R2 = \Pi_{\text{matricole}} (\alpha_{\text{data} \geq '11/11/2018' \wedge \text{data} \leq '31/12/2018'} (\text{GA}))$$

$$R1 \cap R2 = R1 - (R1 - R2)$$

2. Trovare i dipartimenti che hanno avuto tutti i benefit e tutti i gadget;

$$R2 = \frac{R1}{\pi_{i,k}(\text{BENEFIT})} \quad R4 = \frac{R3}{\pi_{i,k}(\text{BADGE})}$$

3. Trovare i gadget che sono stati dati a tutti i dipendenti;

$$R_1 = \pi_{il}(\text{GADGET})$$

$$R_2 = \pi_{\text{material}, il}(\text{GA})$$

$$R_2 \doteq R_1$$

- c. Rispondere alle seguenti query in SQL:

2. Trovare i dipendenti che hanno avuto tutti i tipi di gadget;

SELECT *
FROM IMPIEGATO i
WHERE NOT EXISTS (SELECT *
FROM GADGET g
WHERE NOT EXISTS (SELECT *
FROM GADGET ASSEGNAZIONE GA
NATURAL JOIN GADGET g1
WHERE g1.tipo = g.tipo AND
g.a.matricola = i.matricola))

1. Trovare i dipartimenti che hanno avuto gadget per un valore totale maggiore di quello medio di tutti i dipartimenti con sede 'SICILIA';

```
CREATE VIEW ValoriTotaliGadget AS  
SELECT d.id, d.descr, SUM(g.valore) AS totale  
FROM DIPARTIMENTO d, GADGET g
```

```
SELECT d.id, d.sede, SUM(g.valore) AS Totale
FROM DIPARTIMENTO d
JOIN IMPIEGATO i ON i.dipartimento = d.id
JOIN GADGET ASSEGNAZIONE ga ON ga.matricola = i.matricola
JOIN GADGET g ON g.id = ga.id
GROUP BY d.id, d.sede
```

```
SELECT f.id, f.totale
FROM ValoriTotaliGadget f
WHERE f.totale > (SELECT AVG(totale) FROM ValoriTotaliGadget
WHERE sede = "SICILIA")
```

3. Trovare i dipendenti che hanno avuto benefit e dipendenti che non hanno avuto benefit
(scrivere solo una query piana);

```
SELECT i.id, COALESCE(b.id, "Non Assegno") AS benefit
FROM IMPIEGATO i
LEFT JOIN BENEFIT ASSEGNAZIONE b
ON b.matricola = i.matricola
```

RICETTA (codice, nome, istruzioni, tipo)

TIPOLOGIA (id, nome)

ALIMENTO (id, nome, costo, gruppo_alimentare)

UTENSILE (id, nome, quantita_disponibile)

INGREDIENTI (ricetta, alimento, quantita)

USAUTENSILE (ricetta, utensile, numero)

a. Identificare le chiavi primarie ed esterne dello schema [1 punto];

Rispondere alle seguenti query in algebra relazionale:

Per ogni ricetta trovare l'ingrediente usato in maggior quantità [3 punti];

$$R1 = \text{R2} = \text{INGREDIENTI}$$

$$R3 = \pi_{\text{ricetta}, \text{alimento}} \left(\sigma_{R1.\text{quantita} > R2.\text{quantita}} \wedge R1.\text{id} = R2.\text{id} \right) (R1 \times R2)$$

$$R4 = R1 - R3$$

Trovare le tipologie di ricette che richiedono tutti gli utensili e tutti i cibi [2 punti].

$$R1 = \pi_{\text{tipo}, \text{elemento}} (\text{INGREDIENTI} \bowtie_{\text{ricetta} = \text{id}} \text{RICETTA})$$

$$R2 = \pi_{\text{tipo}, \text{utensile}} (\text{USAUTENSILE} \bowtie_{\text{ricetta} = \text{id}} \text{RICETTA})$$

$$R3 = R1 \div \delta_{\text{id} \rightarrow \text{alimento}} (\pi_{\text{id}} (\text{ALIMENTO}))$$

$$R4 = R2 \div \delta_{\text{id} \rightarrow \text{utensile}} (\pi_{\text{id}} (\text{UTENSILE}))$$

$$R3 \cap R4$$

Trovare gli utensili che sono stati usati in tutte le ricette [2 punti].

$$\pi_{\text{utensile}} (\text{USAUTENSILE}) \div \delta_{\text{costo} \rightarrow \text{ricetta}} (\pi_{\text{id}} (\text{RICETTA}))$$

Rispondere alle seguenti query in SQL:

Trovare le tipologie di ricette che hanno un costo di cibo maggiore di quello medio di tutte le ricette di tipo 'NOUVELLE CUISINE' [2 punti];

```
CREATE VIEW CostoPerTipo AS
SELECT t.nome, SUM(a.costo * i.quantita) AS costo_cibo, COUNT(DISTINCT r.id) AS numero
FROM RICETTA r, ALIMENTO a, INGREDIENTI i, TIPOLOGIA t
WHERE a.id = r.id AND a.gruppo_alimentare = i.id AND
      a.tipo = t.id
GROUP BY t.nome

SELECT *
FROM CostoPerTipo
WHERE costo_cibo > (SELECT costo_cibo / numero
                      FROM CostoPerTipo WHERE nome = "NOUVELLE CUISINE")
```

Trovare le ricette che hanno usato tutti gli utensili [2 punti];

```
SELECT *
FROM ricette r
WHERE NOT EXISTS (SELECT * FROM UTENSILE u
                   WHERE NOT EXISTS (SELECT * FROM USAUTENSILE
                                     WHERE ricetta = r.id AND
                                         utensile = u.id))
```

Trovare le ricette che hanno usato utensili e quelle che non hanno usato utensili (scrivere solo una query piana) [2 punti];

```
SELECT RICETTA.id, USAUTENSILE.utensile
FROM RICETTA
LEFT JOIN USAUTENSILE
ON ricetta.id = usa utensile
```

Esercitazione 3

mercoledì 4 novembre 2020 09:59

Persona (id, username, numero_messaggi)

Gruppo (id, nome)

Appartenenza (idg, idp, datainizio)

MessaggioGruppo (idm, idg, idp, data, oggetto, contenuto)

PresaVisione (idm, idp)

SQL

- Trovare il gruppo con piu' membri che hanno scritto almeno 5 messaggi [3 punti]

CREATE VIEW UtentiAttivi AS

```
SELECT idg, idp
FROM MessaggioGruppo
GROUP BY idg, idp
HAVING COUNT(*) >= 5;
```

CREATE VIEW Statistiche AS

```
SELECT idg, COUNT(*) AS numero
FROM UtentiAttivi
GROUP BY idg;
```

```
SELECT *
FROM Statistiche
WHERE numero = (SELECT MAX(numero) FROM Statistiche)
```

- Trovare le persone che hanno preso visione di tutti i messaggi per ogni singolo gruppo [3 punti]

```
SELECT *
FROM Persona P
WHERE NOT EXISTS(
    SELECT *
    FROM Gruppo g
    WHERE NOT EXISTS( SELECT *
                        FROM MessaggioGruppo m
                        JOIN PresaVisione p1 ON m.idm = p1.idm
                        WHERE m.idg = g.id AND
                        p1.idp = P.idp))
```

Algebra

- Trovare le persone che appartengono a tutti gruppi [3 punti]

$$\Pi_{idp, idg} (\text{APPARTENENZA}) \div \delta_{id \rightarrow idg} (\Pi_{id} (\text{GRUPPO}))$$

- Trovare le persone che non hanno preso visione di nessun messaggio del gruppo [2 punti]

$$Rf = \Pi_{\substack{\text{PresaVisione}.idp \\ \text{MessaggioGruppo}.idg}} (\text{MessaggioGruppo} \setminus \text{MessaggioGruppo}.idm = \text{PresaVisione}.idm \text{ PresaVisione})$$

MessaggioGruppo. idg

$\boxed{\text{TI}_{idP, idG}(\text{Appartenenza}) - RI}$

Esercitazione 4

mercoledì 4 novembre 2020 10:01

Si consideri il seguente schema relazionale relativo ad un sistema di elezioni online:

```
DIRETTIVO(idpersona, ruolo, dataInizioIncarico, dataFineIncarico)
ELETTORATOPASSIVO(idcandidato, ruolo, votiRicevuti, idelezione)
ELEZIONE(idelezione, DataOraAperturaUrne, DataOraChiusuraUrne, elezioneChiusa)
```

Implementare un trigger che dopo l'aggiornamento del campo "elezioneChiusa" inserisce all'interno della tabella DIRETTIVO i neoeletti [4 punti].

CREATE TRIGGER Monitoraggio_Elezioni
AFTER UPDATE OF elezioneChiusa ON ELEZIONE
FOR EACH ROW
WHEN new.elezioneChiusa = 1
INSERT INTO DIRETTIVO VALUES
(SELECT idcandidato, ruolo, NEW.DataOraChiusuraUrne,
NEW.DataOraChiusuraUrne + 3
FROM ELETTORATO PASSIVO
WHERE (idcandidato, ruolo) IN (
SELECT idcandidato, ruolo
FROM ELETTORATO PASSIVO e1
WHERE idelezione = NEW.idelezione AND
voti ricevuti >= ALL (SELECT voti ricevuti
FROM ELETTORATO PASSIVO e2
WHERE e1.ruolo = e2.ruolo AND
e2.idelezione = e1.idelezione))
)

Formalizzazione
 merdi 15 gennaio 2021 10:43

Tel (Pref, Num, Località, Abbonato, Via)
 $F = \{ \text{Pref, Num} \rightarrow \text{Loc, Abbonato, Via}; \text{Loc} \rightarrow \text{pref} \}$

Chiavari = Prof, Num | Num, loc

See → Prof

La relazione è in 3NF

$P_N \rightarrow L A \checkmark$

Decomposition in BCNF
 $\text{Loc} \rightarrow \text{Pref}$ ↗

$$\rightarrow R1 = (\text{Loc}, \text{Pref})$$

R2 = (Num, Loe, Abb, Via)

$\text{LN} \rightarrow \text{LA}^V$

© 2010 Pearson Education, Inc.

$R(A, B, C, D, E, F, G, H, I)$

 $F = \{A \rightarrow B \in EI, B \rightarrow A, D \rightarrow F, E \rightarrow D, F \rightarrow C, G \rightarrow F, I \rightarrow H, H \rightarrow I\}$

Chiavi: AF AF BF BG

 $A^+ = \{A, B, C, \underline{E}, F, H, D\}$
 $B^+ = \{B, A, C, \bar{E}, \bar{I}, \# , D\}$

non è ne BCNF né 3NF

$D \rightarrow E \rightarrow R1 = (D, E)$
 $R2 = (A, B, C, D, \bar{F}, G, H, I) \quad \pi_F(R2) = \{A \rightarrow B \in DI, B \rightarrow A, F \rightarrow G, G \rightarrow F, I \rightarrow H, H \rightarrow I\}$

$F \rightarrow G \rightarrow R3 = (F, G)$
 $R4 = (A, B, C, D, F, H, I) \quad \pi_F(R4) = \{A \rightarrow B \in DF, B \rightarrow A, \underline{I} \rightarrow H, \underline{H} \rightarrow I\}$

$H \rightarrow I \rightarrow R5 = (H, I)$
 $R6 = (A, B, C, D, F, H) \quad \pi_F(R6) = \{A \rightarrow B \in DH, \underline{B} \rightarrow A\}$

$H \rightarrow B \in DH \rightarrow R7 = (A, B, C, D, H) \quad \pi_F(R7) = \{A \rightarrow B \in DH, B \rightarrow A\}$
 $\rightarrow R8 = (A, F) \quad \pi_F(R8) = \{\}$

SELECT DISTINCT B.colore
 FROM M P B
 WHERE M.idm = P.idm AND P.idb = B.idb AND
 M.nome_m = "Marco"

[FROM B JOIN P ON P.idb = B.idb
 JOIN M ON M.idm = P.idm
 WHERE M.nome_m = "Marco"]

Si consideri lo schema Marinaio-Barca-Prenotazione
 M(idm, nomem, rating, eta)
 B(idb, nomeb, colore)
 P(idm, idb, data)

Stampare gli id dei marinai che hanno un rating di almeno 8 o che hanno prenotato la barca 103;

SELECT idm
 FROM M
 WHERE rating >= 8
 UNION
 SELECT idm
 FROM P
 WHERE idb = 103

Si consideri lo schema Marinaio-Barca-Prenotazione
M(idm, nomem, rating, eta)
B(idb, nomeb, colore)
P(idm, idb, data)

Trovare il nome dei marinai che **non hanno** prenotato barche rosse;

```
SELECT nome m
FROM M
WHERE NOT EXISTS (SELECT * FROM P, B
                    WHERE P.idb = B.idb AND
                      B.colore = "rosso" AND
                      P.idm = M.idm)
WHERE M.idm < ALL (SELECT idm FROM P
                     NATURAL JOIN B
                     WHERE B.colore = "rosso")
```

Si consideri lo schema Marinaio-Barca-Prenotazione
M(idm, nomem, rating, eta)
B(idb, nomeb, colore)
P(idm, idb, data)

Trovare il nome dei marinai che hanno prenotato **almeno** due barche;

```
SELECT M.nomem
FROM P, M
WHERE P.idm = M.idm
GROUP BY M.nomem
HAVING COUNT(*) >= 2
HAVING COUNT(DISTINCT P.idb) >= 2
```

Si consideri lo schema Marinaio-Barca-Prenotazione

M(idm, nomem, rating, eta)

B(idb, nomeb, colore)

P(idm, idb, data)

Trovare i nomi dei marinai che hanno prenotato tutte le barche:

```

SELECT nomem
FROM M
WHERE NOT EXISTS (
    SELECT * FROM B
    WHERE NOT EXISTS (
        SELECT *
        FROM P
        WHERE P.idm = M.idm AND
        P.idb = B.idb
    )
)

```

EVENTO(id, titolo, data, categoria, costo_partecipazione, idcatering)

ORGANIZZATORE(id, idevento)

PERSONA(id, nome, cognome, provincia_residenza)

PARTECIPANTE(idevento, idpersona)

CATERING(idcompagnia, nome, descrizione)

CATEGORIAEVENTO(id, descrizione)

Trovare gli eventi che hanno avuto il maggiore incasso.

```

SELECT e.id, costo_partecipazione * COUNT(*) AS costo_totale
FROM EVENTO e, PARTECIPANTE p
WHERE p.id_evento = e.id
GROUP BY p.id
HAVING costo_totale >= ALL (SELECT costo_partecipante * COUNT(*)
                                FROM EVENTO e, PARTECIPANTE p
                                WHERE p.id_evento = e.id
                                GROUP BY p.id)

HAVING costo_totale = (SELECT MAX(t.costo)
                        FROM (SELECT costo_partecipante * COUNT(*) AS costo
                              FROM EVENTO e, PARTECIPANTE p
                              WHERE p.id_evento = e.id
                              GROUP BY p.id) AS t)

```

EVENTO(id, titolo, data, categoria, costo_partecipazione, idcatering)

ORGANIZZATORE(id, idevento)

PERSONA(id, nome, cognome, provincia_residenza)

PARTECIPANTE(idevento, idpersona)

CATERING(idcompagnia, nome, descrizione)

CATEGORIAEVENTO(id, descrizione)

Trovare gli eventi che hanno avuto il minor numero di partecipanti nel 2013.

GROUP BY p.id) AS T)

EVENTO(id, titolo, data, categoria, costo_partecipazione, idcatering)
ORGANIZZATORE(id,idevento)
PERSONA(id, nome, cognome, provincia_residenza)
PARTECIPANTE(idevento,idpersona)
CATERING(idcompagnia, nome, descrizione)
CATEGORIAEVENTO(id,descrizione)

Trovare gli eventi che hanno avuto il minor numero di partecipanti nel 2013.

EVENTO(id, titolo, data, categoria, costo_partecipazione, idcatering)
ORGANIZZATORE(id,idevento)
PERSONA(id, nome, cognome, provincia_residenza)
PARTECIPANTE(idevento,idpersona)
CATERING(idcompagnia, nome, descrizione)
CATEGORIAEVENTO(id,descrizione)

Trovare le persone che hanno ~~partecipato~~ a tutti gli eventi organizzati dagli organizzatori di Catania.

SELECT *
FROM PERSONA P1

WHERE NOT EXISTS (SELECT * FROM EVENTO e
JOIN ORGANIZZATORE o ON o.idevento = e.id
JOIN PERSONA p2 ON p2.id = o.id
WHERE p2.provincia_residenza = "Catania" AND
NOT EXISTS (SELECT * FROM PARTECIPANTE pa
WHERE pa.idpersona = p1.id
AND pa.idevento = e.id))

CONTOCORRENTE(id_conto,saldo,data_apertura) *CC*
 PRODOTTOFINANZIARIO(id_conto, id_prodotto,data_stipula,numero_rate,id_contraente) *PF*
 PERSONA(id_persona, nome,cognome,data_nascita) *P*
 TITOLARECC(id_persona, ID_conto) *TCC*
 TRANSAZIONE(id_contoIN,id_contoOUT,data,causale,dare_avere,importo) *T*
 TRANSAZIONEPRODOTTOFINANZIARIO(id_conto,data,causale ,importo,id_prodotto) *TPF*

Trovare la persona che ha il saldo totale più elevato

```

SELECT TCC.id_persona, CC.saldo |
FROM CC
JOIN TCC ON TCC.ID_conto = CC.id_conto
WHERE CC.saldo = (SELECT MAX (saldo) FROM CC)
      CC.saldo >= ALL (SELECT saldo FROM CC)
    
```

CONTOCORRENTE(id_conto,saldo,data_apertura)
 PRODOTTOFINANZIARIO(id_conto, id_prodotto,data_stipula,numero_rate,id_contraente)
 PERSONA(id_persona, nome,cognome,data_nascita)
 TITOLARECC(id_persona, ID_conto)
 TRANSAZIONE(id_contoIN,id_contoOUT,data,causale,dare_avere,importo)
 TRANSAZIONEPRODOTTOFINANZIARIO(id_conto,data,causale ,importo,id_prodotto)

Trovare il prodotto finanziario che ha più rate

```

SELECT id_prodotto
FROM PF
WHERE numero_rate = (SELECT MAX(numero_rate)
                      FROM PF)
    
```

$$\geq \text{ALL} (\text{SELECT numero_rate} \\
\text{FROM PP})$$

Escursione(id, titolo, descrizione, durata, difficoltà, costo) F
DataEscursione(id, data, idescursione, id guida)
Partecipante(idpartecipante, idescursione)
Persona(id, nome, cognome)

Trovare le escursioni (indicando titolo, descrizione e difficoltà) che hanno un costo massimo

*SELECT titolo, descrizione, "difficoltà"
FROM ESCURSIONE
WHERE costo >= ALL (SELECT costo
FROM ESCURSIONE)*

Escursione(id, titolo, descrizione, durata, difficoltà, costo)
DataEscursione(id, data, idescursione, id guida)
Partecipante(idpartecipante, idescursione)
Persona(id, nome, cognome)

Trovare i partecipanti (dando nome e cognome in output) che hanno partecipato a tutte le escursioni

*SELECT nome, cognome
FROM PERSONA P
WHERE NOT EXISTS (SELECT *
FROM ESCURSIONE E
WHERE NOT EXISTS (SELECT *
FROM PARTECIPANTE
WHERE idpartecipante = p.id AND
idescursione = e.id))*

Escursione(id, titolo, descrizione, durata, difficoltà, costo)
DataEscursione(id, data, idescursione, id guida)
Partecipante(idpartecipante, idescursione)
Persona(id, nome, cognome)

Trovare le guide che non hanno mai partecipato ad escursioni di
difficoltà massima

Escursione(id, titolo, descrizione, durata, difficoltà, costo)
DataEscursione(id, data, idescursione, id guida)
Partecipante(idpartecipante, idescursione)
Persona(id, nome, cognome)

Trovare le coppie di persone che hanno partecipato sempre alle
stesse escursioni

Escursione(id, titolo, descrizione, durata, difficoltà, costo)

DataEscursione(id, data, idescursione, id guida)

Partecipante(idpartecipante, idescursione)

Persona(id, nome, cognome)

Dire ogni accompagnatore quante escursioni ha guidato

Studente(matricola,nome,cognome)Materia(id,titolo,descrizione)Esercizi(id,testo,soluzione,materia,numerosoluzioni)Risolto(idesercizio,idstudente,data)

Si supponga di avere le seguenti operazioni:

q1 - Inserisci soluzione nella relazione Risolto 100 volte al giorno

q2 - Dai il numero di soluzioni proposte per un esercizio ↪

Valutare se conviene mantenere l'attributo "numerosoluzioni"

$$\begin{array}{ll}
 \text{TOT con risol.} & \text{Senza Risol.} \\
 500 + x & 200 + 11x \\
 500 + xL & 200 + 11x \\
 500 - 200 \leq 10x \\
 300 \leq 10x \\
 x \geq 30 \quad \boxed{\text{volte al giorno}}
 \end{array}$$

Con risoluzioni da

q1	1S	Risolto
q2	1L	Esercizi
	1S	Esercizi
q2	1L	Esercizi

$$5L \times 100/g = 500L/g$$

$$1L \times x$$

$$2L \times 100/g = 200L/g$$

$$11L \times 2x$$

Senza Risoluzioni

q1	1S	Risolto
q2	1L	Esercizi
	10L	Risolto

$$R = \{ \text{Nome}, \text{Livello}, \text{Stipendio} \} \quad F = \{ N \rightarrow L, S \mid L \rightarrow S \}$$

- Dove lo schema è in 3NF o BCNF

$$F = \{ \underline{N \rightarrow L}, N \rightarrow S \mid L \rightarrow S \} \quad N^+ = \{ N, L, S \}$$

non è BCNF o 3NF

- Decomponere in BCNF

$$R_1 = \{ \text{Nome}, \text{Livello} \} \quad R_2 = \{ \text{Nome}, \text{Stipendio} \}$$

Progettazione 2

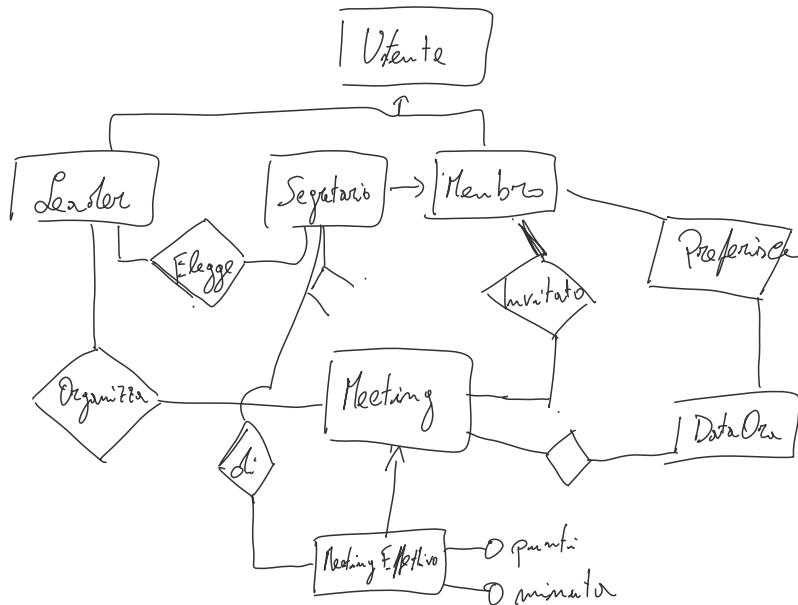
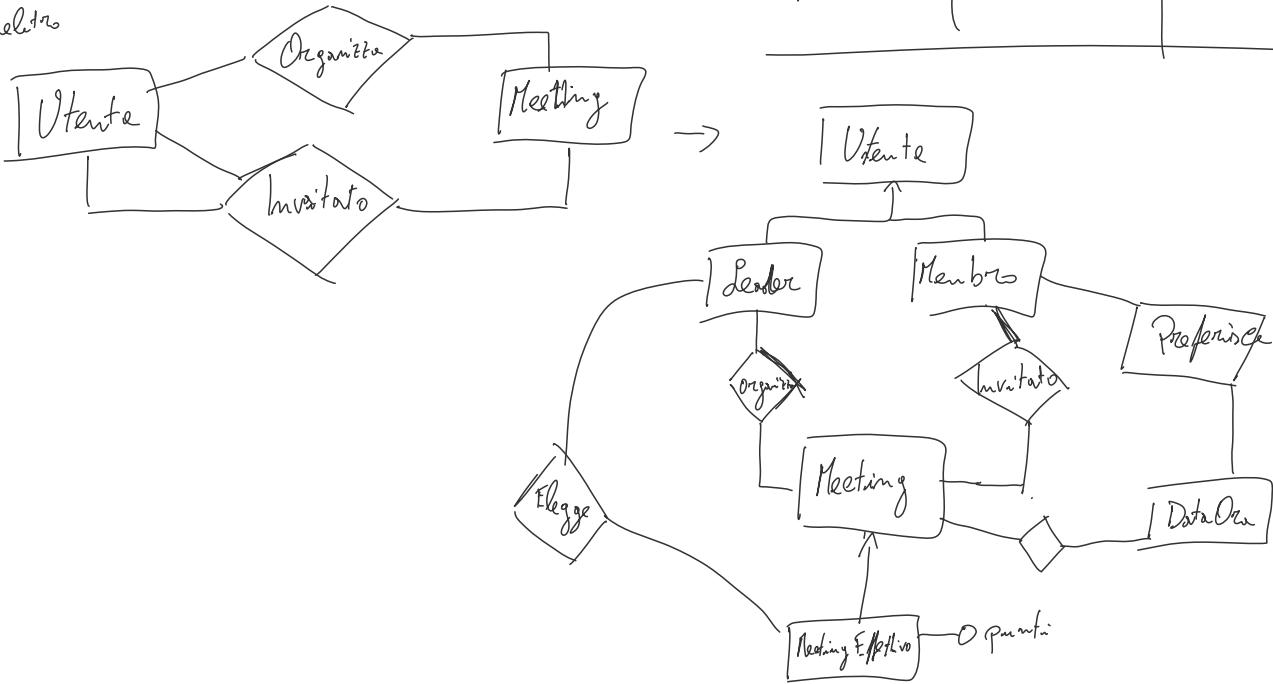
mercoledì 13 gennaio 2021 11:22

Si vuole progettare un database per la gestione dei meeting. Il sistema consente di organizzare dei meeting remoti tramite un meccanismo ad inviti. Un utente (leader) inizia un meeting proponendo una serie di date e di ore. Allo stesso tempo invita un insieme di utenti (membri) a partecipare al meeting. Questi entro 7 giorni dalla creazione del meeting devono dare le preferenze le quali verranno notificate al leader. Una volta ricevute tutte le preferenze o superati i sette giorni il leader identifica la data-ora più condivisa e rende effettivo il meeting notificandolo ai membri.

Dopo il meeting il leader elegge uno dei membri per stilare una minuta contenente tutti i punti trattati nel meeting. L'utente eletto una volta compilata la minuta effettua una notifica a tutti gli utenti.

1. Effettuare un'analisi dei requisiti e sviluppare la progettazione concettuale usando la strategia top-down [4 punti];

Scheda 1



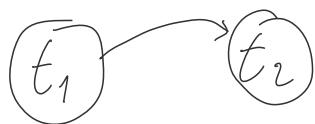
Indicare quali di questi schedule sono VSR oppure CSR

Inoltre, Se gli schedule si presentassero a uno scheduler che usa il locking a due fasi, quali transazioni verrebbero messe in attesa? Si noti che, una volta posta in attesa una transazione, le sue successive azioni non vanno più considerate.

1. r1(x), w1(x), r2(z), r1(y), w1(y), r2(x), w2(x), w2(z)

g

La schedule è CSR

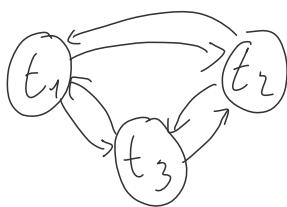


X	$r_1 w_1 r_2 w_2$
y	$r_1 w_1$
z	$r_2 w_2$

ZPL
Non ho dead-lock, ma ho transazioni in attesa

2. r1(x), w1(x), w3(x), r2(y), r3(y), w3(y), w1(y), r2(x)

Non è CSR



X	$r_1 w_1 w_3 r_2$
y	$r_2 r_3 w_3 w_1$

$$\text{I) } r_1(x) w_1(x) w_3(x) r_3(y) w_3(y) r_2(y) r_2(x)$$

$$\text{II) } w_3(x) r_3(y) w_3(y) r_1(x) w_1(x) w_1(y) r_2(y) r_2(x) \quad \text{dove ha finali } w_1(x) w_1(y)$$

$$\text{III) } r_2(y) r_2(x) w_3(x) r_3(y) w_3(y) r_1(x) w_1(x) w_1(y) \quad \text{dove ha finali } w_1(x) w_1(y)$$

Non è VSR

	rlock	wlock
X	t_1	
y	t_2	

t_3 è in attesa di t_1
 t_1 è in attesa di t_2
 t_2 è in attesa di t_3

3. r1(x), r2(x), w2(y), r2(x), r4(z), w1(x), w3(y), w3(z), w1(y), w5(x), w1(z), w5(y), r5(z)

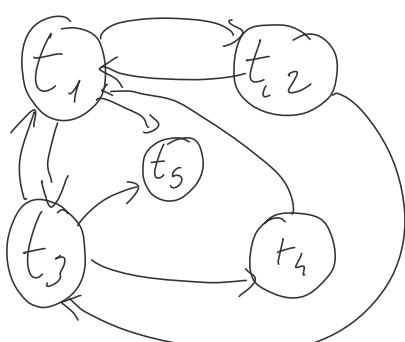
non è CSR

A A A n



$r_5(z)$ legge $w_1(z)$
scrive finali $w_5(x) w_1(x) w_5(y)$

non è VSR



	rlock	wlock
X	t_1, t_2	
y		t_3

c'è un dead lock
 t_1 attende t_2
 t_2 attende t_1
... + + +

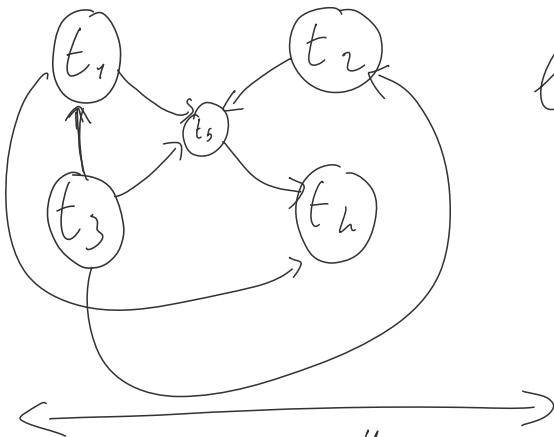
\bar{z}
 y

t_3

t_2 attende t_1
 t_5 attende t_1 e t_2

4. $r1(x), r3(y), w1(y), w4(x), w1(t), w5(x), r2(z), r3(z), w2(z), w5(z), r4(t), r5(t)$

X	z_1, w_4, w_5
y	z_3, w_1
z	z_2, z_4, w_2, w_5
t	w_1, z_4, z_5



lo schedule è
CSR
equivalente anche VSR

le transazioni t_1, t_4, t_5 sono mero un altro

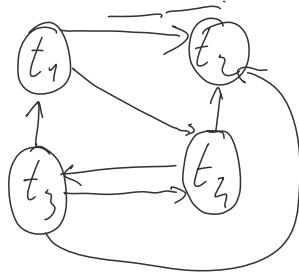
al termine non ci sono dead-lock perché tutte le transazioni terminano

5. $r1(x), r2(x), w2(x), r3(x), r4(z), w1(x), \bar{r}3(y), r3(x), w1(y), w5(x), w1(z), r5(y), r5(z)$

X

6. $r1(x), r1(t), r3(z), r4(z), w2(z), r4(x), r3(x), w4(x), w4(y), w3(y), w1(y), w2(t) \sim$

X	z_1, z_4, z_3, w_4
y	w_4, w_3, w_1
z	z_3, z_4, w_2
t	z_1, w_2



non è CSR

↓ ↓ ↓ ↓

$t_3 \quad t_4 \quad t_1 \quad t_2$

$t_4 \quad t_3 \quad t_1$

VSR

non è VSR
perché qualunque transazione
abbia la stessa scrittura finale
 t_4 deve precedere t_1) introdurre
la legge da " $r_1(x)$ legge da $w_1(x)$ "

scrittura finale: $w_2(t), \underline{w_1(y)}, \underline{w_4(x)}, w_2(z)$

$t_1 \quad \underline{z_1(x)} \underline{z_1(t)} \quad w_1(y)$
 $t_2 \quad w_2(z) \quad w_2(t)$
 $t_3 \quad z_3(z) \quad z_3(x) \quad w_3(y)$
 $t_4 \quad z_4(z) \quad z_4(x) \quad \underline{w_4(x)}$

legge da: non ci sono legge da

$r1(x), r1(t), r3(z), r4(z), w2(z), r4(x), r3(x), w4(x), w4(y), w3(y), w1(y), w2(t)$

$\cancel{t_1 \quad r_lock \quad x, t \quad w_lock \quad y}$
 $\cancel{t_3 \quad r_lock \quad z, x \quad w_lock \quad y}$
 $\cancel{t_4 \quad r_lock \quad z, x}$

t_2 è in attesa su x di t_3 e t_4
 t_4 è in attesa su x di t_1 e t_3
 t_3 termina
 t_1 termina
 t_L riprende e termina

lo schedule non
ha dead-lock

t_4 ~~è lock di t_1~~

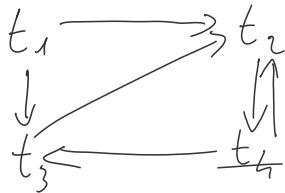
t_1 termina

t_3 riprende e termina

t_2 riprende e termina

7. $r2(x), r4(x), w4(x), r1(y), r4(z), w4(z), w3(y), w3(z), w1(t), w2(z), w2(t)$

x	$r_2 \ r_3 \ w_4$
y	$r_1 \ w_3$
z	$r_4 \ w_4 \ w_3 \ w_2$
t	$w_1 \ w_2$



non è CSR

\rightarrow
x-lock $t_2 t_4$
y-lock

x y z \rightarrow

t_4 è in attesa su x della t_2
 t_3 è in attesa su y della t_1

t_3 si rivolge a termina

Esercitazione 2

lunedì 11 gennaio 2021 11:13

Descrivere la ripresa a caldo, indicando la costituzione progressiva degli insiemi di UNDO e REDO e le azioni di recovery, a fronte del seguente log:

DUMP, B(T1), B(T2), B(T3), I(T1, O1, A1), D(T2, O2, B2), B(T4), U(T4, O3, B3, A3),
 U(T1, O4, B4, A4), C(T2), CK(T1, T3, T4), B(T5), B(T6), U(T5, O5, B5, A5), A(T3), CK(T1, T4, T5, T6),
 B(T7), A(T4), U(T7, O6, B6, A6), U(T6, O3, B7, A7), B(T8), A(T7), guasto

$$\begin{aligned} \text{UNDO} &= \{T_1, T_4, T_5, T_6\} & \text{REDO} &= \{\} \\ \text{UNDO} &= \{T_1, T_4, T_5, T_6, T_2\} & \text{REDO} &= \{\} \\ \text{UNDO} &= \{T_1, T_2, T_5, T_6, T_2, T_8\} & \text{REDO} &= \{\} \end{aligned}$$



aggiorniamo O₃ con B₂
 " O₆ con B₆
 " O₅ con B₅
 " O₄ con B₄
 " O₃ con B₃

cancelare

P₁

Si supponga che nella situazione precedente si verifichi un guasto di dispositivo che coinvolge gli oggetti O₁, O₂, O₃; descrivere la ripresa a freddo.

Esercitazione 3

venerdì 15 gennaio 2021 10:00

- . Indicare se i seguenti schedule possono produrre anomalie; i simboli ci e ai indicano l'esito (commit o abort) della transazione.
 1. r1(x), w1(x), r2(x), w2(y), a1, c2
 2. r1(x), w1(x), r2(y), w2(y), a1, c2
 3. r1(x), r2(x), r2(y), w2(y), r1(z), a1, c2
 4. r1(x), r2(x), w2(x), w1(x), c1, c2
 5. r1(x), r2(x), w2(x), r1(y) , c1, c2
 6. r1(x), w1(x), r2(x), w2(x) , c1, c2

Esercitazione 1

venerdì 15 gennaio 2021 10:02

- Dato il seguente frammento di DTD:

```
<!ELEMENT libretti (libretto*) >
<!ELEMENT libretto (studente,esami?) >
<!ELEMENT studente (nome,cognome) >
<!ATTLIST studente cdl CDATA #REQUIRED
    matricola CDATA #REQUIRED >
<!ELEMENT esami (esame+) >
<!ELEMENT esame (materia,voto,data) >
```

dove per tutti gli elementi non specificati si assuma una definizione di (#PCDATA), si fornisca:

- un file xml, che esemplifichi l'uso di tale DTD, contenente i dati relativi ai primi due esami sostenuti a Informatica dal candidato e ad uno studente Teo Verdi che non ha dato esami

```
<?xml version="1.0" ?>
<libretti>
    <libretto>
        <studente cdl="667" matricola="002723">
            <nome>Salvatore</nome>
            <cognome>Alaimo</cognome>
        </studente>
        <esami>
            <esame>
                <materia>Analisi 1</materia>
                <voto>18</voto>
                <data>01/01/2010</data>
            </esame>
            <esame>
                <materia>Formazione Discreta 1</materia>
                <voto>18</voto>
                <data>31/12/2010</data>
            </esame>
        </esami>
    </libretto>
    <libretto>
        <studente cdl="M01" matricola="1234567">
            <nome>Teo</nome>
            <cognome>Verdi</cognome>
        </studente>
    </libretto>
</libretti>
```

- una interrogazione XQuery che da un file libretti.xml conforme a tale DTD estragga cognome e nome di tutti gli studenti che hanno preso almeno un trenta agli esami

```
<risultati>
{
    for $lib in doc("libretti.xml")//libretto
    where $lib//voto>29
    return <studente>
        Alternativa 1:
        <nome>{$lib/studente/nome/text()}</nome>
        <cognome>{$lib/studente/cognome/text()}</cognome>
        Alternativa 2:
        { $lib/studente/nome, $lib/studente/cognome }
    </studente>
}
</risultati>
```

- una interrogazione XQuery che, per ogni materia presente in libretti.xml, fornisca un elenco delle matricole degli studenti che hanno sostenuto il relativo esame.

```
<risultati>
{
    for $m in distinct-values(doc("libretti.xml")//esame/materia)
    return <risultato>
        <materia>{$m/text()}</materia>
        {
            for $lib in doc("libretti.xml")//libretto[esami/esame/materia=$m]
            return <matricola>{$lib/studente/@matricola}</matricola>
        }
    </risultato>
}
</risultati>
```

Esercitazione 2

venerdì 15 gennaio 2021 10:02

Dato il seguente DTD:

```
<?xml version="1.0"?>
<!DOCTYPE campionati[
<!ELEMENT campionati(campionato)*>
<!ELEMENT campionato(parita)*>
<!ELEMENT partita(squadraCasa,SquadraOspite,risultato)*>
<!ELEMENT squadraCasa (#PCDATA)>
<!ELEMENT squadraOspite (#PCDATA)>
<!ELEMENT risultato(gol)*>
<!ELEMENT gol(giocatore)*>
<!ELEMENT giocatore(#PCDATA)>
<!ATTLIST campionato anno CDATA #REQUIRED>
<!ATTLIST partita data CDATA #REQUIRED>
<!ATTLIST risultato golSquadraCasa CDATA #REQUIRED>
<!ATTLIST risultato golSquadraOspite CDATA #REQUIRED>
<!ATTLIST giocatore nomeSquadra IDREF #REQUIRED>
]>
```

1. Dare un file XML che esemplifichi l'uso del DTD

```
<campionati>
<campionato anno="2020">
  <partita data="29/02/2020">
    <squadraCasa>Squadra 1</squadraCasa>
    <squadraOspite>Squadra 2</squadraOspite>
    <risultato golSquadraCasa="1" golSquadraOspite="1">
      <gol><giocatore nomeSquadra="Squadra 1">Mario Rossi</giocatore></gol>
      <gol><giocatore nomeSquadra="Squadra 2">Saro Falsaperla</giocatore></gol>
    </risultato>
  </partita>
</campionato>
</campionati>
```

2. Trovare il giocatore nel campionato del 2014 ha segnato più gol.

```
{
let $conteggi := <risultato>
{
  let $campionato := doc("partite.xml")//campionato[@anno="2014"]
  for $g in distinct-values ($campionato//giocatore)
  let $gol := count ($campionato//giocatore[text()=$g/text()])
  return <giocatore>
    <nome>{$g/text()}</nome>
    <conteggio>{$gol}</conteggio>
  </giocatore>
}
</risultato>

for $g in $conteggi//nome
where $g/conteggio >= max($conteggi//conteggio)
return <giocatore>{$g/text()}</giocatore>
}
```

3. Dare il risultato di tutte le partite giocate dalla Juventus contro la Roma

4. Dare l'elenco delle partite che hanno avuto come risultato 0 0.

XML version "1.0"

```
<!DOCTYPE campionati [
  <!ELEMENT campionati>
  <!ELEMENT campionato(parita)*>
  <!ELEMENT partita (squadraCasa, squadraOspite, risultato)*>
  <!ELEMENT squadraCasa (#PCDATA)>
  <!ELEMENT squadraOspite (#PCDATA)>
  <!ELEMENT risultato (gol)*>
  <!ELEMENT gol (giocatore)*>
  <!ELEMENT giocatore (#PCDATA)>
  <!ATTLIST campionato anno CDATA #REQUIRED>
  <!ATTLIST partita data CDATA #REQUIRED>
  <!ATTLIST giocatore nomeSquadra CDATA #REQUIRED>
]>
```

Si consideri il seguente schema relazionale relativo alla gestione dei benefit e dei gadget assegnati agli impiegati di una azienda.

IMPIEGATO (matricola, nome, cognome, dipartimento)
DIPARTIMENTO (id, nome, sede)
BENEFIT (id, nome, valore)
GADGET (id, nome, tipo, valore)
BENEFITASSEGNNATO (matricola, id, data)
GADGETASSEGNNATO (matricola, id, data)

BA
GA

a. Identificare le chiavi primarie ed esterne dello schema;

b. Rispondere alle seguenti query in algebra relazionale:

1. Trovare gli impiegati che nel 2018 non hanno avuto benefit ma hanno avuto gadget;

$$\begin{aligned} R1 &= \pi_{\text{matricola}}(\text{IMPIEGATO}) - \pi_{\text{matricola}}(\sigma_{\text{data} \geq '1/1/2018' \wedge \text{data} \leq '31/12/2018'}(\text{BA})) \\ R2 &= \pi_{\text{matricola}}(\sigma_{\text{data} \geq '1/1/2018' \wedge \text{data} \leq '31/12/2018'}(\text{GA})) \\ R1 \cap R2 &= R1 - (R1 - R2) \end{aligned}$$

2. Trovare i dipartimenti che hanno avuto tutti i benefit e tutti i gadget;

$$\begin{aligned} R1 &= \pi_{\text{dipartimento}, \text{id}}(\text{IMPIEGATO} \bowtie \text{BA}) & R3 &= \pi_{\text{dipartimento}, \text{id}}(\text{IMPIEGATO} \bowtie \text{GA}) \\ R2 &= R1 \div \pi_{\text{id}}(\text{BENEFIT}) & R4 &= R3 \div \pi_{\text{id}}(\text{GADGET}) \\ R2 \cap R4 & & R2 \bowtie R4 & \end{aligned}$$

3. Trovare i gadget che sono stati dati a tutti i dipendenti;

$$\begin{aligned} R1 &= \pi_{\text{id}}(\text{GADGET}) \\ R2 &= \pi_{\text{matricola}, \text{id}}(\text{GA}) \\ R2 \div R1 & \end{aligned}$$

c. Rispondere alle seguenti query in SQL:

2. Trovare i dipendenti che hanno avuto tutti i tipi di gadget;

—

```

SELECT *
FROM IMPIEGATO i
WHERE NOT EXISTS (
    SELECT *
    FROM GADGET g
    WHERE NOT EXISTS (
        SELECT *
        FROM GADGET ASSEGNNATO ga
        NATURAL JOIN GADGET gj
        WHERE gj.tipo = gj1.tipo AND
        ga.matricola = i.matricola
    )
)

```

1. Trovare i dipartimenti che hanno avuto gadget per un valore totale maggiore di quello medio di tutti i dipartimenti con sede 'SICILIA';

CREATE VIEW **Valori Totali Gadget AS**
 $\text{SELECT d.id, d.sede, SUM(g.valore) AS totale}$
 $\text{FROM DIPARTIMENTO d, GADGET g}$

```
SELECT d.id, d.sede, SUM(g.valore) AS Totale
FROM DIPARTIMENTO d
JOIN IMPIEGATO i ON i.dipartimento = d.id
JOIN GADGET ASSEGNAZIONE ga ON ga.matricola = i.matricola
JOIN GADGET g ON g.id = ga.id
GROUP BY d.id, d.sede
```

```
SELECT f.id, f.totale
FROM ValoriTotaliGadget f
WHERE f.totale > (SELECT AVG(totale) FROM ValoriTotaliGadget
WHERE sede = "SICILIA")
```

3. Trovare i dipendenti che hanno avuto benefit e dipendenti che non hanno avuto benefit
(scrivere solo una query piana);

```
SELECT i.id, COALESCE(b.id, "Non Assegno") AS benefit
FROM IMPIEGATO i
LEFT JOIN BENEFIT ASSEGNAZIONE b
ON b.matricola = i.matricola
```

RICETTA (codice, nome, istruzioni, tipo)

TIPOLOGIA (id, nome)

ALIMENTO (id, nome, costo, gruppo_alimentare)

UTENSILE (id, nome, quantita_disponibile)

INGREDIENTI (ricetta, alimento, quantita)

USAUTENSILE (ricetta, utensile, numero)

a. Identificare le chiavi primarie ed esterne dello schema [1 punto];

Rispondere alle seguenti query in algebra relazionale:

Per ogni ricetta trovare l'ingrediente usato in maggior quantità [3 punti];

$$R1 = \pi_{\text{ricetta}} (\text{INGREDIENTI})$$

$$R3 = \pi_{\text{ricetta}, \text{alimento}, \text{quantita}} (\text{INGREDIENTI}) \quad \text{R1.quantita} > \text{R2.quantita} \quad \text{R1.} \text{id} = \text{R2.} \text{id}$$

$$R4 = R1 - R3$$

Trovare le tipologie di ricette che richiedono tutti gli utensili e tutti i cibi [2 punti].

$$R1 = \pi_{\text{tipo}, \text{elemento}} (\text{INGREDIENTI})$$

$$R2 = \pi_{\text{tipo}, \text{utensile}} (\text{USAUTENSILE})$$

$$R3 = R1 \div \delta_{\text{id} \rightarrow \text{alimento}} (\pi_{\text{id}} (\text{ALIMENTO}))$$

$$R4 = R2 \div \delta_{\text{id} \rightarrow \text{utensile}} (\pi_{\text{id}} (\text{UTENSILE}))$$

$$R3 \cap R4$$

Trovare gli utensili che sono stati usati in tutte le ricette [2 punti].

$$\pi_{\text{utensile}} (\text{USAUTENSILE}) \div \delta_{\text{costo} \rightarrow \text{ricetta}} (\pi_{\text{costo}} (\text{RICETTA}))$$

Rispondere alle seguenti query in SQL:

Trovare le tipologie di ricette che hanno un costo di cibo maggiore di quello medio di tutte le ricette di tipo 'NOUVELLE CUISINE' [2 punti];

```
CREATE VIEW CostoPerTipo AS
SELECT t.nome, SUM(a.costo * i.quantita) AS costo_cibo, COUNT(DISTINCT r.codice) AS numero
FROM RICETTA r, ALIMENTO a, INGREDIENTI i, TIPOLOGIA t
WHERE a.id = r.id AND a.gruppo_alimentare = i.id AND
      a.tipo = t.id
GROUP BY t.nome

SELECT *
FROM CostoPerTipo
WHERE costo_cibo > (SELECT costo_cibo / numero
                      FROM CostoPerTipo WHERE nome = "NOUVELLE CUISINE")
```

Trovare le ricette che hanno usato tutti gli utensili [2 punti];

```
SELECT *
FROM ricette r
WHERE NOT EXISTS (SELECT * FROM UTENSILE u
                   WHERE NOT EXISTS (SELECT * FROM USAUTENSILE
                                     WHERE ricetta = r.codice AND
                                         utensile = u.id))
```

Trovare le ricette che hanno usato utensili e quelle che non hanno usato utensili (scrivere solo una query piana) [2 punti];

```
SELECT RICETTA.codice, USAUTENSILE.utensile
FROM RICETTA
LEFT JOIN USAUTENSILE
ON codice = ricetta
```

Esercitazione 4

mercoledì 4 novembre 2020 10:01

Si consideri il seguente schema relazionale relativo ad un sistema di elezioni online:

```
DIRETTIVO(idpersona, ruolo, dataInizioIncarico, dataFineIncarico)
ELETTORATOPASSIVO(idcandidato, ruolo, votiRicevuti, idelezione)
ELEZIONE(idelezione, DataOraAperturaUrne, DataOraChiusuraUrne, elezioneChiusa)
```

Implementare un trigger che dopo l'aggiornamento del campo "elezioneChiusa" inserisce all'interno della tabella DIRETTIVO i neoeletti [4 punti].

CREATE TRIGGER Monitoraggio_Elezioni
AFTER UPDATE OF elezioneChiusa ON ELEZIONE
FOR EACH ROW
WHEN new.elezioneChiusa = 1
INSERT INTO DIRETTIVO VALUES
(SELECT idcandidato, ruolo, NEW.DataOraChiusuraUrne,
NEW.DataOraChiusuraUrne + 3
FROM ELETTORATO PASSIVO
WHERE (idcandidato, ruolo) IN (
SELECT idcandidato, ruolo
FROM ELETTORATO PASSIVO e1
WHERE idelezione = NEW.idelezione AND
voti ricevuti >= ALL (SELECT voti ricevuti
FROM ELETTORATO PASSIVO e2
WHERE e1.ruolo = e2.ruolo AND
e2.idelezione = e1.idelezione))
)

Si consideri il seguente schema relazionale relativo alla gestione dei benefit e dei gadget assegnati agli impiegati di una azienda.

IMPIEGATO (matricola, nome, cognome, dipartimento)
DIPARTIMENTO (id, nome, sede)
BENEFIT (id, nome, valore)
GADGET (id, nome, tipo, valore)
BENEFITASSEGNNATO (matricola, id, data)
GADGETASSEGNNATO (matricola, id, data)

BA
GA

a. Identificare le chiavi primarie ed esterne dello schema;

b. Rispondere alle seguenti query in algebra relazionale:

1. Trovare gli impiegati che nel 2018 non hanno avuto benefit ma hanno avuto gadget;

$$\begin{aligned} R1 &= \pi_{\text{matricola}}(\text{IMPIEGATO}) - \pi_{\text{matricola}}(\sigma_{\text{data} \geq '1/1/2018' \wedge \text{data} \leq '31/12/2018'}(\text{BA})) \\ R2 &= \pi_{\text{matricola}}(\sigma_{\text{data} \geq '1/1/2018' \wedge \text{data} \leq '31/12/2018'}(\text{GA})) \\ R1 \cap R2 &= R1 - (R1 - R2) \end{aligned}$$

2. Trovare i dipartimenti che hanno avuto tutti i benefit e tutti i gadget;

$$\begin{aligned} R1 &= \pi_{\text{dipartimento}, \text{id}}(\text{IMPIEGATO} \bowtie \text{BA}) & R3 &= \pi_{\text{dipartimento}, \text{id}}(\text{IMPIEGATO} \bowtie \text{GA}) \\ R2 &= R1 \div \pi_{\text{id}}(\text{BENEFIT}) & R4 &= R3 \div \pi_{\text{id}}(\text{GADGET}) \\ R2 \cap R4 & & R2 \bowtie R4 & \end{aligned}$$

3. Trovare i gadget che sono stati dati a tutti i dipendenti;

$$\begin{aligned} R1 &= \pi_{\text{id}}(\text{GADGET}) \\ R2 &= \pi_{\text{matricola}, \text{id}}(\text{GA}) \\ R2 \div R1 & \end{aligned}$$

c. Rispondere alle seguenti query in SQL:

2. Trovare i dipendenti che hanno avuto tutti i tipi di gadget;

—

```

SELECT *
FROM IMPIEGATO i
WHERE NOT EXISTS (
    SELECT *
    FROM GADGET g
    WHERE NOT EXISTS (
        SELECT *
        FROM GADGET ASSEGNNATO ga
        NATURAL JOIN GADGET gj
        WHERE gj.tipo = gj1.tipo AND
        ga.matricola = i.matricola
    )
)

```

1. Trovare i dipartimenti che hanno avuto gadget per un valore totale maggiore di quello medio di tutti i dipartimenti con sede 'SICILIA';

CREATE VIEW **Valori Totali Gadget AS**
 $\text{SELECT d.id, d.sede, SUM(g.valore) AS totale}$
 $\text{FROM DIPARTIMENTO d, GADGET g}$

```

SELECT d.id, d.sede, SUM(g.valore) AS totale
FROM DIPARTIMENTO d
JOIN IMPIEGATO i ON i.dipartimento = d.id
JOIN GADGET ASSEGNAZIONE ga ON ga.matricola = i.matricola
JOIN GADGET g ON g.id = ga.id
GROUP BY d.id, d.sede

SELECT f.id, f.totale
FROM ValoriTotaliGadget f
WHERE f.totale > (SELECT AVG(totale) FROM ValoriTotaliGadget
WHERE sede = "SICILIA")

```

3. Trovare i dipendenti che hanno avuto benefit e dipendenti che non hanno avuto benefit
(scrivere solo una query piana);

```

SELECT i.id, COALESCE(b.id, "Non Assegno") AS benefit
FROM IMPIEGATO i
LEFT JOIN BENEFIT ASSEGNAZIONE b
ON b.matricola = i.matricola

```

Persona (id, username, numero_messaggi)Gruppo (id, nome)Appartenenza (idg, idp, datainizio)MessaggioGruppo (idm, idg, idp, data, oggetto, contenuto)Presavisione (idm, idp)**SQL**

- Trovare il gruppo con piu' membri che hanno scritto almeno 5 messaggi [3 punti]

CREATE VIEW UtentiAttivi AS

```
SELECT idg, idp
FROM MessaggioGruppo
GROUP BY idg, idp
HAVING COUNT(*) >= 5;
```

CREATE VIEW Statistiche AS

```
SELECT idg, COUNT(*) AS numero
FROM UtentiAttivi
GROUP BY idg;
```

```
SELECT *
FROM Statistiche
WHERE numero = (SELECT MAX(numero) FROM Statistiche)
```

- Trovare le persone che hanno preso visione di tutti i messaggi per ogni singolo gruppo [3 punti]

```
SELECT *
FROM Persona P
WHERE NOT EXISTS(
    SELECT *
    FROM Gruppo g
    WHERE NOT EXISTS( SELECT *
                        FROM MessaggioGruppo m
                        JOIN PresaVisione p1 ON m.idm = p1.idm
                        WHERE m.idg = g.id AND
                        p1.idp = P.idp))
```

Algebra

- Trovare le persone che appartengono a tutti gruppi [3 punti]

$$\Pi_{idp, idg} (\text{APPARTENENZA}) \div \delta_{id \rightarrow idg} (\Pi_{id} (\text{GRUPPO}))$$

- Trovare le persone che non hanno preso visione di nessun messaggio del gruppo [2 punti]

$$Rf = \Pi_{\substack{\text{Presavisione}.idp \\ \text{MessaggioGruppo}.idg}} (\text{MessaggioGruppo} \setminus \Pi_{\substack{\text{Presavisione}.idm \\ \text{MessaggioGruppo}.idm}} (\text{Presavisione}.idm))$$

MessaggioGruppo. idg

$\boxed{\text{TI}_{idP, idG}(\text{Appartenenza}) - RI}$

Si consideri lo schema Marinaio-Barca-Prenotazione

M(idm, nomen, rating, eta)

B(idb, nomeb, colore)

P(idm, idb, data)

Trovare i **colori** della barca prenotata dal marinaio Marco;

```
SELECT DISTINCT B.colore
FROM M P B
WHERE M.idm = P.idm AND P.idb = B.idb AND
M.nomenom = "Marco"
```

```
FROM B JOIN P ON P.idb = B.idb
JOIN M ON M.idm = P.idm
WHERE M.nomenom = "Marco"
```

Si consideri lo schema Marinaio-Barca-Prenotazione

M(idm, nomen, rating, eta)

B(idb, nomeb, colore)

P(idm, idb, data)

Stampare gli id dei marinai che hanno un rating di almeno 8 o che hanno prenotato la barca 103;

```
SELECT idm
FROM M
WHERE rating >= 8
UNION
SELECT idm
FROM P
WHERE idb = 103
```

Si consideri lo schema Marinaio-Barca-Prenotazione
M(idm, nomem, rating, eta)
B(idb, nomeb, colore)
P(idm, idb, data)

Trovare il nome dei marinai che **non hanno** prenotato barche rosse;

```
SELECT nome m
FROM M
WHERE NOT EXISTS (SELECT * FROM P, B
                    WHERE P.idb = B.idb AND
                      B.colore = "rosso" AND
                      P.idm = M.idm)
```

WHERE M.idm < ALL (SELECT idm FROM P
 NATURAL JOIN B
 WHERE B.colore = "rosso")

Si consideri lo schema Marinaio-Barca-Prenotazione
M(idm, nomem, rating, eta)
B(idb, nomeb, colore)
P(idm, idb, data)

Trovare il nome dei marinai che hanno prenotato **almeno** due barche;

```
SELECT M.nomem
FROM P, M
WHERE P.idm = M.idm
GROUP BY M.nomem
HAVING COUNT(*) >= 2
```

HAVING COUNT(DISTINCT P.idb) >= 2

Si consideri lo schema Marinaio-Barca-Prenotazione

M(idm, nomem, rating, eta)

B(idb, nomeb, colore)

P(idm, idb, data)

Trovare i nomi dei marinai che hanno prenotato tutte le barche:

```

SELECT nomem
FROM M
WHERE NOT EXISTS (
    SELECT * FROM B
    WHERE NOT EXISTS (
        SELECT *
        FROM P
        WHERE P.idm = M.idm AND
        P.idb = B.idb
    )
)

```

EVENTO(id, titolo, data, categoria, costo_partecipazione, idcatering)

ORGANIZZATORE(id,idevento)

PERSONA(id, nome, cognome, provincia_residenza)

PARTECIPANTE(idevento,idpersona)

CATERING(idcompagnia, nome, descrizione)

CATEGORIAEVENTO(id,descrizione)

Trovare gli eventi che hanno avuto il maggiore incasso.

```

SELECT e.id, costo_partecipazione * COUNT(*) AS costo_totale
FROM EVENTO e, PARTECIPANTE p
WHERE p.id_evento = e.id
GROUP BY p.id
HAVING costo_totale >= ALL (SELECT costo_partecipante * COUNT(*)
                                FROM EVENTO e, PARTECIPANTE p
                                WHERE p.id_evento = e.id
                                GROUP BY p.id)

HAVING costo_totale = (SELECT MAX(t.costo)
                        FROM (SELECT costo_partecipante * COUNT(*) AS costo
                              FROM EVENTO e, PARTECIPANTE p
                              WHERE p.id_evento = e.id
                              GROUP BY p.id) AS t)

```

EVENTO(id, titolo, data, categoria, costo_partecipazione, idcatering)

ORGANIZZATORE(id,idevento)

PERSONA(id, nome, cognome, provincia_residenza)

PARTECIPANTE(idevento,idpersona)

CATERING(idcompagnia, nome, descrizione)

CATEGORIAEVENTO(id,descrizione)

Trovare gli eventi che hanno avuto il minor numero di partecipanti nel 2013.

GROUP BY p.id) AS T)

EVENTO(id, titolo, data, categoria, costo_partecipazione, idcatering)
ORGANIZZATORE(id,idevento)
PERSONA(id, nome, cognome, provincia_residenza)
PARTECIPANTE(idevento,idpersona)
CATERING(idcompagnia, nome, descrizione)
CATEGORIAEVENTO(id,descrizione)

Trovare gli eventi che hanno avuto il minor numero di partecipanti nel 2013.

EVENTO(id, titolo, data, categoria, costo_partecipazione, idcatering)
ORGANIZZATORE(id,idevento)
PERSONA(id, nome, cognome, provincia_residenza)
PARTECIPANTE(idevento,idpersona)
CATERING(idcompagnia, nome, descrizione)
CATEGORIAEVENTO(id,descrizione)

Trovare le persone che hanno ~~partecipato~~ tutti gli eventi organizzati dagli organizzatori di Catania.

SELECT *
FROM PERSONA P1

WHERE NOT EXISTS (SELECT * FROM EVENTO e
JOIN ORGANIZZATORE o ON o.idevento = e.id
JOIN PERSONA p2 ON p2.id = o.id
WHERE p2.provincia_residenza = "Catania" AND
NOT EXISTS (SELECT * FROM PARTECIPANTE pa
WHERE pa.idpersona = p1.id
AND pa.idevento = e.id))

CONTOCORRENTE(id_conto,saldo,data_apertura) *CC*
 PRODOTTOFINANZIARIO(id_conto, id_prodotto,data_stipula,numero_rate,id_contraente) *PF*
 PERSONA(id_persona, nome,cognome,data_nascita) *P*
 TITOLARECC(id_persona, ID_conto) *TCC*
 TRANSAZIONE(id_contoIN,id_contoOUT,data,causale,dare_avere,importo) *T*
 TRANSAZIONEPRODOTTOFINANZIARIO(id_conto,data,causale ,importo,id_prodotto) *TPF*

Trovare la persona che ha il saldo totale più elevato

```

SELECT TCC.id_persona, CC.saldo |
FROM CC
JOIN TCC ON TCC.ID_conto = CC.id_conto
WHERE CC.saldo = (SELECT MAX (saldo) FROM CC)
      CC.saldo >= ALL (SELECT saldo FROM CC)
    
```

CONTOCORRENTE(id_conto,saldo,data_apertura)
 PRODOTTOFINANZIARIO(id_conto, id_prodotto,data_stipula,numero_rate,id_contraente)
 PERSONA(id_persona, nome,cognome,data_nascita)
 TITOLARECC(id_persona, ID_conto)
 TRANSAZIONE(id_contoIN,id_contoOUT,data,causale,dare_avere,importo)
 TRANSAZIONEPRODOTTOFINANZIARIO(id_conto,data,causale ,importo,id_prodotto)

Trovare il prodotto finanziario che ha più rate

```

SELECT id_prodotto
FROM PF
WHERE numero_rate = (SELECT MAX(numero_rate)
                      FROM PF)
      >= ALL (SELECT numero_rate
              FROM PP)
    
```

Escursione(id, titolo, descrizione, durata, difficoltà, costo) F
DataEscursione(id, data, idescursione, id guida)
Partecipante(idpartecipante, idescursione)
Persona(id, nome, cognome)

Trovare le escursioni (indicando titolo, descrizione e difficoltà) che hanno un costo massimo

*SELECT titolo, descrizione, "difficoltà"
FROM ESCURSIONE
WHERE costo >= ALL (SELECT costo
FROM ESCURSIONE)*

Escursione(id, titolo, descrizione, durata, difficoltà, costo)
DataEscursione(id, data, idescursione, id guida)
Partecipante(idpartecipante, idescursione)
Persona(id, nome, cognome)

Trovare i partecipanti (dando nome e cognome in output) che hanno partecipato a tutte le escursioni

*SELECT nome, cognome
FROM PERSONA P
WHERE NOT EXISTS (SELECT *
FROM ESCURSIONE E
WHERE NOT EXISTS (SELECT *
FROM PARTECIPANTE
WHERE idpartecipante = p.id AND
idescursione = e.id))*

`Escursione(id, titolo, descrizione, durata, difficoltà, costo)`
`DataEscursione(id, data, idescursione, id guida)`
`Partecipante(idpartecipante, idescursione)`
`Persona(id, nome, cognome)`

Trovare le guide che non hanno mai partecipato ad escursioni di
difficoltà massima

`Escursione(id, titolo, descrizione, durata, difficoltà, costo)`
`DataEscursione(id, data, idescursione, id guida)`
`Partecipante(idpartecipante, idescursione)`
`Persona(id, nome, cognome)`

Trovare le coppie di persone che hanno partecipato sempre alle
stesse escursioni

Escursione(id, titolo, descrizione, durata, difficoltà, costo)

DataEscursione(id, data, idescursione, id guida)

Partecipante(idpartecipante, idescursione)

Persona(id, nome, cognome)

Dire ogni accompagnatore quante escursioni ha guidato

Si consideri il seguente schema relazionale:

Libro(titolo, autore, data_prima_pubblicazione)
 Autore(id, nome, cognome)
 Genere(id, nome, descrizione)
 GenereLibro(idg, titolo)
 Utente(cf, nome, cognome)
 LibroLetto(cf, titolo, data, commenti)

2) Rispondere alle seguenti query in algebra relazionale:

a. Trovare gli utenti che hanno letto tutti i libri di "Stephen King" [4 punti]

$$R1 = \prod_{\text{titolo}} \left(\sigma_{\substack{\text{name} = \text{"Stephen"} \\ \text{cognome} = \text{"King"}}} (\text{Libro} \setminus \text{Libro} \rightarrow \text{Autore} . \text{id} = \text{Autore} . \text{id}) \right)$$

$$R2 = \prod_{\text{cf}, \text{titolo}} (\text{LibroLetto}) \div R1$$

b. Trovare gli autori che non hanno mai scritto "thriller" o "horror" [4 punti]

$$R1 = (\text{Libro} \setminus \text{Genere} (\text{Libro} \setminus \text{GenereLibro}.idg = \text{Genere}.id \text{ Genere}))$$

$$R2 = \sigma_{\text{genere.name} = \text{"thriller"} \vee \text{genere.name} = \text{"horror"} } (R1)$$

$$R3 = \sum_{\text{autore} \rightarrow \text{id}} (\prod_{\text{autore}} (R2))$$

$$RL = \prod_{\text{id}} (\text{Autore}) - R3$$

3) Rispondere alle seguenti query in SQL

Libro(titolo, autore, data_prima_pubblicazione)
 Autore(id, nome, cognome)
 Genere(id, nome, descrizione)
 GenereLibro(idg, titolo)
 Utente(cf, nome, cognome)
 LibroLetto(cf, titolo, data, commenti)

a. Per ogni utente contare quanti libri di ogni genere ha letto [4 punti]

AS conteggio

$$\begin{aligned} & \text{SELECT cf, idg, COUNT(DISTINCT titolo)} \\ & \text{FROM LibroLetto JOIN Libro ON Libro.titolo = LibroLetto.titolo} \\ & \text{JOIN GenereLibro ON GenereLibro.titolo = Libro.titolo} \\ & \text{GROUP BY cf, idg} \end{aligned}$$

b. Trovare i libri che sono stati letti da tutti gli utenti [4 punti]

$$\begin{aligned} & \text{SELECT * FROM Libro l} \\ & \text{WHERE NOT EXISTS (SELECT * FROM Utente u} \\ & \text{WHERE NOT EXISTS (SELECT * FROM LibroLetto ll} \\ & \text{WHERE ll.cf = u.cf AND} \\ & \text{ll.titolo = l.titolo)) \end{aligned}$$

4) Si supponga di avere la relazione GenerePreferito(cf, genere) che contiene i generi di libri preferiti da ogni utente. Implementare un trigger che ogni qual volta viene inserito un record nella relazione LibroLetto si verifica per il particolare utente se sono stati letti almeno 10 libri del genere appena inserito, in questo caso viene inserito, se non esiste già, il record nella relazione GenerePreferito. [4 punti]

CREATE TRIGGER T1 AFTER INSERT ON LibroLetto FOR EACH ROW
WHEN 10 <= (SELECT COUNT(DISTINCT titolo)
FROM LibroLetto
WHERE cf = NEW.cf AND
titolo IN (SELECT g1.titolo
FROM GeneraLibro g1, GeneraLibro g2
WHERE g1.idg = g2.idg AND
g2.titolo = NEW.titolo))

DECLARE x, y INT
BEGIN
SELECT COUNT(*), genere INTO x, y FROM GenerePreferito
WHERE cf = NEW.cf AND genere IN (SELECT idg
FROM GeneraLibro
WHERE titolo = NEW.titolo)
GROUP BY genere;
IF (x = 0) THEN
INSERT INTO GenerePreferito(cf, genere)
VALUES (NEW.cf, y);
END IF
END

- 1) Si consideri il seguente schema relazionale relativo alla gestione di un centro d'assistenza:

CLIENTE (cf, nome, cognome)

TECNICO (codice, nome, cognome, costo_orario)

ELETTRODOMESTICO (id, marca, tipo, modello)

ACQUISTO (cliente, elettrodomestico, data_acquisto, durata_garanzia)

CHIAMATEASSISTENZA (cf, elettrodomestico, dataora_chiamata, guasto, chiusa, ore_lavorate, costo_ricambi, costo_totale)

ASSEGNAZIONE (cf, elettrodomestico, dataora_chiamata, tecnico)

RICAMBIO (cf, elettrodomestico, dataora_chiamata, ricambio, costo)

Si consideri che la durata della garanzia è espressa in mesi, il costo dei ricambi è zero se l'elettrodomestico è in garanzia, e che una chiamata può essere assegnata ad un solo tecnico.

1. Identificare le chiavi primarie ed esterne dello schema [0 punti se corretta, -1 se sbagliata o senza risposta].

2. Scrivere in algebra relazionale le seguenti query:

1. Trovare tutti i clienti che hanno avuto almeno una chiamata con costo superiore a 500€, indicando nome, cognome, data della prima chiamata e data di acquisto dell'elettrodomestico [2 punti].

$$R1 = \pi_{Costo_totale > 500} (\text{CHIAMATE ASSISTENZA})$$

$$R2 = R1$$

$$R3 = \pi_{\substack{R1.cf \\ R1.dataora_chiamata}} (R1 \bowtie_{R1.ef=R2.cf} R2)$$

$$R4 = (R1 \bowtie_{\substack{R1.cf \\ R1.dataora_chiamata}} R3) \bowtie_{\substack{\text{Cliente} \\ R1.ef=Aquisto.chiama \\ R1.elettrodomestico=Aquisto.elettrodomestico}}$$

$$\pi_{\substack{\text{nome, cognome, data_acquisto, dataora_chiamata} \\ (R4)}}$$

1. Trovare tutti i clienti che hanno fatto chiamate di assistenza per tutte le marche di elettrodomestici [3 punti].

$$R1 = \pi_{\substack{ef \\ \text{marca}}} (\text{CHIAMATE ASSISTENZA} \bowtie_{\text{elettrodomestico} = id} \text{ELETTRODOMESTICO})$$

$$R2 = \pi_{\text{marca}} (\text{ELETTRODOMESTICO})$$

$$R1 \div R2$$

3. Scrivere in SQL le seguenti query:

1. Per ogni pezzo di ricambio indicare quante volte è stato usato per riparazioni fuori garanzia [2 punti].

```
SELECT ricambio, COUNT(*) AS conteggio
FROM RICAMBIO
WHERE costo > 0
GROUP BY ricambio
```

1. Trovare la tipologia di elettrodomestico che ha richiesto il maggior numero di ore di riparazione [3 punti]

```
CREATE VIEW OreAssistenza AS
SELECT elettrodomestico, SUM(ore_lavorate) AS ore
FROM CHIAMATE ASSISTENZA GROUP BY elettrodomestico;

SELECT DISTINCT tipo
FROM OreAssistenza JOIN ELETTRODOMESTICO ON elettrodomestico = id
GROUP BY tipo
HAVING SUM(ore) = ALL (SELECT SUM(ore)
FROM OreAssistenza JOIN ELETTRODOMESTICO
ON elettrodomestico = id
GROUP BY tipo)
```

```
CREATE VIEW OreAssistenza AS
SELECT tipo, SUM(ore_lavorate) AS ore
FROM CHIAMATE ASSISTENZA JOIN ELETTRODOMESTICO
ON elettrodomestico = id
```

```

SELECT tipo , sum(ore) AS ore_elettricita
FROM CHIASTE ASSISTENZA JOIN ELETTRODOMESTICO ON elettrodomestico = id
GROUP BY tipo;

```

```

SELECT DISTINCT tipo

```

```

FROM CHIASTE ASSISTENZA
WHERE ore = (SELECT MAX(ore) FROM CHIASTE ASSISTENZA);

```

4. Implementare un trigger che, quando una chiamata di assistenza è chiusa, calcoli il costo dei ricambi e il costo totale.
 Ricordarsi che il costo dei ricambi è zero se l'elettrodomestico è in garanzia. Il costo totale è dato dalla somma del costo dei ricambi e del costo del tecnico.
 Il costo del tecnico è ottenuto moltiplicando le ore lavorate per il costo orario [4 punti].

```

CREATE TRIGGER T1 AFTER UPDATE OF chiusa ON CHIAMATE ASSISTENZA
FOR EACH ROW
WHEN new.chiusa = 1
DECLARE garanzia INT DEFAULT 0;
DECLARE costo_ricambi FLOAT;
DECLARE costo_tecnico FLOAT;
BEGIN
    SELECT (costo_orario * new.ore_lavorate) INTO costo_tecnico
    FROM TECNICO JOIN ASSEGNAZIONE ON tecnico_id = tecnico_id
    WHERE cf = new.cf AND elettrodomestico = new.elettrodomestico AND
        data_ora_chiamata = new.data_ora_chiamata;
    SELECT (new.data_ora_chiamata - (data_acquisto + durata_garanzia)) INTO garanzia
    SELECT (new.acquisto * cf) INTO costo_ricambi
    WHERE cliente = new.cf AND
        elettrodomestico = new.elettrodomestico;
    IF (garanzia = 0) THEN
        SET costo_ricambi = 0;
    ELSE
        SELECT sum(costo) INTO costo_ricambi
        FROM RICAMBI
        WHERE cf = new.cf AND elettrodomestico = new.elettrodomestico AND
            data_ora_chiamata = new.data_ora_chiamata;
    END IF;
    UPDATE CHIAMATE ASSISTENZA
    SET costo_totale = costo_ricambi + costo_tecnico
    WHERE cf = new.cf AND elettrodomestico = new.elettrodomestico AND
        data_ora_chiamata = new.data_ora_chiamata;
END

```

Si consideri il seguente db per la gestione di un campeggio:

```
Piazzola(id, dimensione, prezzo_giornaliero_adulti, prezzo_giornaliero_bambini,
idoneac)
Servizi(id, nome, descrizione, prezzo_giornaliero)
ServiziPiazzola(idp, ids)
Prenotazione(id, data_arrivo, data_partenza, cfcliente, idpiazz, numero_adulti,
numero_bambini, numero_notti, tipo)
ServiziPrenotati(idprenotazione, idp, ids)
Cliente(cf, nome, cognome, tel)
```

L'attributo idoneac della relazione Piazzola è un campo booleano che indica se la piazzola e' idonea ad ospitare camper.
L'attributo tipo della relazione Prenotazione assume i valori {Tenda, Camper}.

b) Rispondere alle seguenti query in SQL

Trovare le piazzole che hanno ospitato tutti i clienti [4 punti]

```
SELECT * FROM Piazzola p
WHERE NOT EXISTS (SELECT *
                   FROM Cliente c
                   WHERE NOT EXISTS (
                       SELECT * FROM Prenotazione
                       WHERE cfcliente = c.cf AND
                             idpiazz = p.id))
```

Per ogni prenotazione determinare il costo finale e il numero di servizi inclusi, e il numero di persone ospitate [5 punti]

```
SELECT p.id, (numero_adulti + numero_bambini) AS persone,
       COUNT(*) AS num_servizi
  FROM Prenotazione p, ServiziPrenotati sp, Servizi s, Piazzola pz
 WHERE sp.idprenotazione = p.id AND s.id = sp.ids AND
       pz.id = p.idpiazz
 GROUP BY p.id
 (( (pz.prezzo_giornaliero_adulti * p.numero_adulti) + (pz.prezzo_giornaliero_bambini *
 p.numero_bambini)) * numero_notti) + SUM(sp.prezzo_giornaliero *
 numero_notti) AS costo_finale
```

a) Rispondere alle seguenti query in algebra relazionale:

Trovare le piazzole, visualizzando l'id, che hanno tutti i servizi [4 punti]

$$\text{ServiziPiazzola} \vdash \delta_{id \rightarrow id} (\Pi_{id}(\text{Servizi}))$$

Trovare le prenotazioni, effettuate nel mese di agosto 2020, di piazzole che non hanno richiesto servizi aggiuntivi [4 punti]

$$R1 = \sigma_{data_arrivo \geq '1/8/2020'} \cap \sigma_{data_arrivo \leq '31/8/2020'} (\text{Prenotazione})$$

$$R2 = \pi_{id\text{prenotazione}}(\text{ServiziPrenotati})$$

$$\Pi_{id}(R1) - \delta_{id\text{prenotazione} \rightarrow id}(R2)$$