Basi di Dati e Sistemi Informativi I, 02 Marzo 2012

Facoltà di Scienze M.F.N., Corso di Laurea in Informatica

$LINEA(\underline{COD-L}, PRIMA-P, ULTIMA-P, FREQUENZA)$
$FERMATA(\underline{COD-F}, NOME, ZONA)$
$ZONA(\underline{COD-Z}, NOME, POPOLAZIONE)$
$COMPOSIZIONE(\underline{COD-L, COD-F}, ORD)$
$CORSE(\underline{Cod-Corsa, COD-L, COD-F, Data}, OraP, OraA)$

Lo schema $LINEA$ contiene la descrizione di linee di trasporto urbano (bus): codice linea, orario della prima e ultima corsa, frequenza delle corse; Lo schema $FERMATA$ contiene la descrizione delle fermate: codice della fermata, nome della fermata e codice della zona in cui si trova la fermata; Lo schema $COMPOSIZIONE$ indica quali fermate sono presenti nelle linee e l'ordine (ORD è un intero) in cui le fermate compaiono nelle linee. Lo schema $CORSE$ indica i tempi delle effettive percorrenze effettuate dagli autobus nelle varie date per ogni tratta fermata-fermata: COD-F si riferisce alla fermata di partenza, OraP all'orario di partenza dalla fermata COD-P e OraA a quella di arrivo alla fermata successiva della linea.

Esercizio 01 *(Punti 7) Si scriva una espressione in algebra relazionale che, se valutata, fornisce il codice delle linee che hanno fermate in un'unica zona.*

Esercizio 02 *(Punti 7) Si scriva una vista che per ogni giornata e per ogni linea restituisca il numero di corse fatte nella giornata in quella linea, il tempo minimo di percorrenza complessiva (capolinea-capolinea) della corsa nella giornata, il tempo massimo di percorrenza complessiva.*

Esercizio 03 *(Punti 8) Si scriva una procedura in PL SQL che riceve in ingresso il codice di due linee. La procedura verifica che le due linee abbiano una fermata in comune (coincidenza). Se non vi è coincidenza la procedura restituisce una stringa di caratteri vuota. Se invece vi è coincidenza la procedura restituisce una stringa di caratteri che contiene nell'ordine la sequenza dei nomi delle fermate (separate da ';') della prima linea fino alla coincidenza (compresa) e a seguire i nomi delle fermate della seconda linea a partire dalla coincidenza.*

Esercizio 04 *(Punti 7) Si consideri la relazione*
$R(A, B, C, D, E)$ *con le seguenti dipendenze funzionali:*

- $A \rightarrow B, E$;
- $B \rightarrow A$;
- $A, C \rightarrow D$;
- $B, C \rightarrow D$;

Trovare le chiavi. Trovare una copertura canonica per le dipendenze. Dire quali dipendenze di R soddisfano i requisiti della terza forma normale e quali soddisfano i requisiti della forma normale di Boyce-Code.
Se la tabella non è in terza forma normale scomporla applicando l'algoritmo di normalizzazione.

Esercizio 05 *(Punti 7) Si scriva una procedura PLSQL che riceve in ingresso il nome una tabella e una stringa di coppie attributo-valore di attributi della tabella e di valori per gli attibuti (della forma attr@val1@attr2@val2@..... @attrk@valk). La procedura usando sql dinamico inserisce una riga nella tabella passata per parametro usando la stringa di coppie attributo-valore. Si ricorda che in Oracle la funzione SUBSTR(stringa, pos, lungh) restituisce la porzione della sottostringa di stringa che ha inizio in pos ed ha lunghezza lungh) e che la funzione INSTR(stringa, search, occ) restituisce la posizione iniziale della occorrenza occ (occ = 1 per la prima occorrenza) della stringa search in stringa se esistente e -1 altrimenti.*

Basi di Dati e Sitemi informativi I, 5 Dicembre 2014

Struttura della base di dati comune a tutte le tracce

Si consideri il seguente schema relazionale che descrive un frammento della base di dati per gestire un'applicazione tipo social network per la condivisione di post (di tipo testuale).

Post(idPost, Autore, Data, Ora, Testo)
Commento(idPost, Ordine, Testo, Autore, Data, Ora)
Key (parola, Tema)
Presenza (IdPost, tema)
Sottoscrizione (IdUtente, Tema, Data)
Notifica (idPost, idUtente, Data, Ora)
Amici (idUtente1, idUtente2)

Post descrive gli interventi nel social network;
Commento indica i commenti ai Post, ordine indica l'ordine di inserimento del commento al post, (IdPost e Ordine sono chiave). *Key* indica parole chiave rilevanti per individuare il tema del psot ed il Tema generale a cui sono riconducibili (ad esempio, parola programmazione, tema informatica). Gli utenti possono esprimere il loro interesse per alcuni temi in modo da ricevere notifiche quando vengono pubblicati post correlati al tema (tabella *Sottoscrizione*). *Notifica* indica le notifiche dei post agli utenti amici e interessati. *Amici* indica la relazione di amicizia tra coppie di utenti.

Prova A

Esercizio 01:
Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce l'identificativo di post che sono stati commentati da tutti gli amici dell'autore.

Esercizio 02:
Si scriva una interrogazione SQL che fornisce l'identificativo di post e del tema del post tale che nel testo del psot occorrono due parole diverse entrambe associate allo stesso tema

Esercizio 03:
Si esprimano nel modo più opportuno i seguenti Vincoli.

1. La relazione Amici è simmetrica (se Ut1, Ut2) si trova nella relazione anche (ut2, ut1) si trova nella relazione.
2. Se un utente ha ricevuto una notifica o è amico dell'autore del post o ha sottoscritto le notifiche del tema del post
3. Le notifiche di un post ad un utente vengono fatte una sola volta
4. I commenti fanno riferimento a post presenti non cancellati

Esercizio 04:

Si scriva una vista che per ogni utente ed ogni mese fornisca un riepilogo dell'attività dell'utente del tipo:
(Utente, anno, mese, **N_post**, **N_nocom**, **N_com_amici**).
N_post il numero di post dell'utente del mese, **N_com_amici** il numero di commenti di amici al post del mese
N_nocom numero di post senza commenti nel mese).

Prova B

Esercizio 01:
Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce l'identificativo di utenti autori di post che sono stati commentati solo da amici (mai da utenti non inclusi tra gli amici)

Esercizio 02:
Si scriva una interrogazione SQL che fornisce l'identificativo di post che sono stati commentati da tutti i sottoscrittori del tema del post

Esercizio 03:
Si esprimano nel modo più opportuno i seguenti Vincoli.

1. L'ordinamento dei commenti deve rispettare l'ordine di data e ora di pubblicazione (non può comparire prima nell'ordine ciò che è stato pubblicato dopo).
2. Se viene indicata la presenze di un tema in un post ci deve essere l'occorrenza nel testo di un post di una parola associata al tema
3. Un utente può sottoscrivere solo una volta un tema
4. I commenti fanno riferimento a post presenti e non cancellati

Esercizio 04:
Si scriva una vista che per ogni utente ed ogni mese fornisca un riepilogo dell'attività dell'utente del tipo:
(Utente, anno, mese, **N_post**, **N_nocom**, **N_com_nonamici**).
N_post il numero di post dell'utente del mese, **N_com_nonamici** il numero di commenti a post dell'utente nel mese fatti da utenti che non sono amici, **N_nocom** numero di post senza commenti).

Prova C

Esercizio 01:
Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce l'identificativo di post che sono stati commentati da tutti i sottoscrittori del tema del post.

Esercizio 02:
Si scriva una interrogazione SQL che fornisce l'identificativo di post che e due parole associate a temi distinti e il post sia tale che nel suo testo occorrono entrambe le parole

Esercizio 03:
Si esprimano nel modo più opportuno i seguenti Vincoli.

1. Se un utente ha ricevuto una notififa o è amico dell'autore del posto o ha sottoscritto le notifiche del tema del post
2. L'ordinamento dei commenti deve rispettare l'ordine di data e ora di pubblicazione (non può comparire prima nell'ordine
3. Le notifiche di un post ad un utente vengono fatte una volta sola.
4. Le Notifiche fanno riferimento a post presenti e non cancellati

Esercizio 04:
Si scriva una vista che per ogni utente ed ogni mese fornisca un riepilogo dell'attività dell'utente del tipo:
(Utente, anno, mese, **N_post**, **N_nocom**, **N_com_sottoscrittori**).
N_post il numero di post dell'utente del mese, **N_com_sottoscrittori** il numero di commenti di sottoscrittori al tema del post, **N_nocom** numero di post senza commenti nel mese).

Basi di Dati e Sistemi Informativi I, 26 gennaio 2015

(Seconda prova intercorso: ultimi 3 esercizi)

* NB: per ogni classe la superclasse, se esiste, è unica.

Facoltà di Scienze M.F.N., Corso di Laurea in Informatica

Si consideri il seguente schema relazionale che descrive la strutturazione di classi in un class diagram di UML.
$CLASS(\underline{Cod}, Nome, Descrizione)$
$REFINE(SuperClass, SubClass)$
$ATTRIBUTI(Class, Nome, Tipo, Posizione)$
$BASICTYPE(Nome)$
$METODO(\underline{CodM}, Class, Nome, TipoOut, Posizione)$
$PARAMETRI(CodM, Class, Nome, Tipo, Posizione)$

$CLASS$ descrive le classi. $REFINE$ descrive la relazione di specializzazione delle classi (SubClass è la specializzazione della classe SuperClass; entrambi gli attibuti sono totali). $ATTRIBUTI$ descrive gli attributi vengono associati ad una classe. Il tipo di un attributo può essere di tipo basico (contenuto nello schema $BASICTYPE$) oppure il nome di una classe. Posizione indica l'ordine in cui compare l'attributo nella classe. $METODO$ descrive i metodi associati ad una classe. Il *Tipout* di ritorno di un metodo può essere di tipo basico (contenuto nello schema $BASICTYPE$) oppure il nome di una classe oppure la costante *Void*. Posizione indica l'ordine in cui compare il metodo nella classe. $PARAMETRI$ descrive i patrametri associati ad un metodo. Il *Tipout* di un attributo può essere di tipo basico (contenuto nello schema $BASICTYPE$) oppure il nome di una classe. Posizione indica l'ordine in cui compare il parametro nel metodo. Si dice che due metodi hanno la stessa segnatura se hanno lo stesso nome, lo stesso dipo di ritorno e gli stessi parametri (in ogni posizione stesso nome e stesso tipo).

Esercizio 01 *(7 punti, II prova 0) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce codice e nome delle classi che prevedono solo metodi senza parametri.*

Esercizio 02 *(7 punti, II prova 0) Si scriva una vista SQL che restituisca per ogni classe il numero di attributi della classe di tipo BASICTYPE, il numero di metodi della classe cha hanno solo parametri di tipo BASICTYPE e il numero di classi che la specializzano.*

Esercizio 03 *(8 punti, II prova 12) Si scriva un metodo PLSQL che riceve in ingresso il codice di una classe e il codice di un metodo e che: se in una superclasse della classe data si trova un metodo con la stessa segnatura (del metodo dato in parametro) restituisce il codice del medodo trovato; altrimenti restituisce -1.*

Esercizio 04 *(8 punti, II prova 12) Utilizzando SQL dinamico, si scriva una funzione PLSQL che riceve in ingresso una sequenza di parole separate dal carattere < (ad esempio < parola1 < parola2 <) La funzione restituisce una stringa contenente i codici delle classi separati da < che contengono almeno una delle parole della lista di input nella loro descrizione.*

Esercizio 05 *(6 punti, II prova 10) Si consideri la relazione* $R(A, B, C, D, E)$ *che soddisfa le seguenti dipendenze logiche:*

- $A, B \rightarrow C$;
- $B, C \rightarrow D$;
- $B \rightarrow E$;
- $C \rightarrow A$;

R

Trovare le chiavi. Dire se ~~ORGANICO~~ è in terza forma normale e se è in forma normale di Boyce-Code. Se la tabella non è in forma normale di Boyce-Code scomporla applicando l'algoritmo di normalizzazione.

$LOG(\underline{Cod}, Operazione, CodRisorsa, ValorePrima, ValoreDopo, CodTransazione, Timestamp)$
$RISORSA(\underline{CodRisorsa}, Locazione, Valore, Stato)$
$RICHIESTE(CodTransazione, Tempo, tipoAccesso, CodRisorsa)$
$ASSEGNAZIONE(CodTransazione, Tempo, CodRisorsa, tipoAccesso)$

Esercizio 01 *(Punti 8) Si implementi il seguente trigger. Quando una transazione registra una operazione di ABORT sul log, tutte le scritture fatte dalla transazione e riportate sul LOG devono essere annullate in ordine inverso a quelle in cui sono state fatte. Per annullare le scritture si deve consultare il log e si deve assegnare ad ogni risorsa scritta dalla transizione il valore ValorePrima riportato nel LOG. Inoltre, le risorse assegnate alla transazione devono tornare libere: si rimuovono le assegnazioni alla transazione e lo stato della risorsa assume valore UNLOCK.*

```
CREATE TRIGGER abort_transaz
AFTER INSERT INTO log
FOR EACH ROW
WHEN NEW.operazione = 'ABORT'


BEGIN
DECLARE
CURSOR rollback_op
FOR
SELECT L.codRisorsa, L.ValorePrima, L.Operazione
FROM LOG L
WHERE L.CodTransazione = NEW.CodiceTransazione
ORDER BY L.Timestamp DESC

BEGIN
FOR operazione IN rollback_op
LOOP
    IF operazione.operazione = 'MODIFICA'
    THEN
        UPDATE Risorse R
        SET R.Valore = Operazione.ValorePrima
        R.stato = 'UNLOCK'
        WHERE R.codRisorsa = operazione.CodRisorsa
    ENDIF

    IF operazione.operazione = 'CREA'
    THEN
    DELETE FROM Risorse R
        WHERE R.codRisorsa = operazione.CodRisorsa
    ENDIF
```

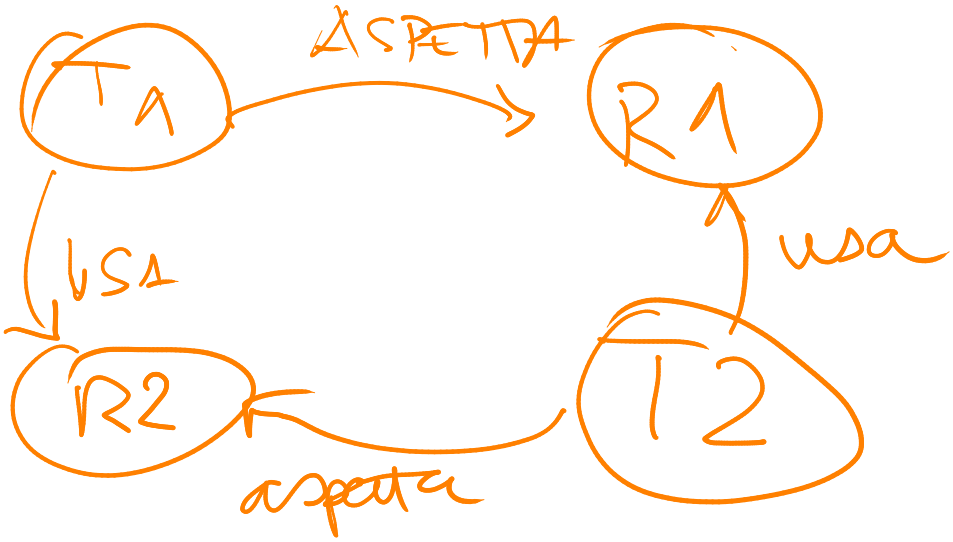
```
IF operazione.operazione = 'CANCELLA'
        THEN
                INSERT INTO Risorse R
                SET (codRisorsa, Valore, Stato)
                VALUES (operazione.CodRisorsa, operazione.ValorePrima,
                        'UNLOCK')
ENDIF
ENDLOOP


  DELETE FROM ASSEGNAZIONE A
         WHERE A.codAssegnazione = NEW.codTransazione

 END
```

$LOG(\underline{Cod}, Operazione, CodRisorsa, ValorePrima, ValoreDopo, CodTransazione, Timestamp)$
$RISORSA(\underline{CodRisorsa}, Locazione, Valore, Stato)$
$RICHIESTE(CodTransazione, Tempo, tipoAccesso, CodRisorsa)$
$ASSEGNAZIONE(CodTransazione, Tempo, CodRisorsa, tipoAccesso)$

Esercizio 02 *(Punti 8) Si scriva una funzione con parametro intero Tout che per tutte le transazioni T1 che sono in attesa per una risorsa per un tempo superiore a Tout (differenza tra il tempo di registrazione della richiesta e il tempo corrente) controlli se ci sia un deadlock (cioè se esiste un'altra transazione T2 che occupa la risorsa richiesta e la transazione T2 richiede una risorsa asegnata alla transazione T1). La funzione restituisce una stringa coi codici delle transazioni T1 in deadlock cosi trovate.*



-- creo e scorro un cursore per recuperare le transazioni bloccate da più di to

-- T1 è bloccata -> recupero le risorse T1.risorsa
-- recupera a quale transazione T2 è assegnata T1.risorsa
-- recupero le risorse richieste da T2
-- verifico se T2 è in attesa della risorsa assegnata a T1

```
CREATE FUNCTION timeout (TO TIME)
RETURN VARCHAR (1000)

DECLARE
        res VARCHAR (1000) = ''
        CURSOR tr_bloccate IS
                SELECT  R.codTrnasazione, R.codRichiesta
                        FROM RICHIESTE R
                        WHERE SYSTIME - R.Tempo >= TO


 T2 Log.CodTransazione%TYPE
 R_used Assegnazione.codRisorsa%TYPE
```

```
BEGIN
FOR T1 IN tr_bloccate
LOOP
        SELECT codTransazione
                INTO T2
                FROM Assegnazione A
                WHERE A.codRisorsa = T1.CodRisorsa

        SELECT COUNT(*)
                INTO R_used
                FROM richiesta AS R1 JOIN Aseegnazione A1
                        ON R1.codRisorsa = A.codRisorsa
                WHERE R1.transazione = T2
                AND A1.codAssegnata = T1.codTransazione

IF R-used > 0
        res = res || T1.codTransazione
ENDIF
ENDLOOP
RETURN risultato
END
```

$$Post(\underline{IdPost}, Autore, Data, Ora, Testo)$$
$$Commento(\underline{IdPost, Ordine}, Testo, Autore, Data, Ora)$$
$$Key(parola, Tema)$$
$$Presenza(IdPost, Tema)$$
$$Sottoscrizione(IdUtente, Tema, Data)$$
$$Notifica(IdPost, IdUtente, Data, Ora)$$
$$Amici(IdUtente1, IdUtente2)$$

Esercizio 01 *Si scriva un trigger che, quando viene inserito un nuovo post, controlla se tra le parole del testo sono presenti delle parole chiave (parole presenti nella tabella Key). Se una parola chiave è presente deve essere inviata una notifica (aggiunta una riga nella tabella Notifica) a tutti gli utenti interessati al tema collegato. Una notifica deve essere inviata anche a tutti gli amici dell'autore del post.*

```
-- apro un cursore per scorrere le parole
-- per ogni parola controllo se è contenuta nel testo
-- se è contenuta recupero il tema associato
-- inserisco la notifica per i sottoscrittori del tema

CREATE TRIGGER new_post
AFTER INSERT ON Post
FOR EACH ROW
BEGIN
DECLARE
BEGIN
    FOR K IN (SELECt * FROM KEY)
    LOOP
        IF INSTR(New.testo, k.parola) > 0
            THEN
                INSERT INTO Notifica
                    VALUES (
                    SELECT New.idPost, s.Utente, SYSDATE, SYSTIME
                        FROM Sottoscrizione S
                        WHERE S.Tema = K.Tema
                    )
    ENDLOOP
```

```
INSERT INTO NOTIFICA
        VALUES ((SELECT New.idPost, A.idUtente1, SYSDATE,SYSTIMe
                FROM Amici A
                WHERE idUtente2= New.Autore
            )
          UNION  (SELECT New.idPost, A.idUtente2, SYSDATE,SYSTIMe
                FROM Amici A
                WHERE idUtente1= New.Autore
            )
END
```

$$Post(\underline{IdPost}, Autore, Data, Ora, Testo)$$
$$Commento(\underline{IdPost, Ordine}, Testo, Autore, Data, Ora)$$
$$Key(parola, Tema)$$
$$Presenza(IdPost, Tema)$$
$$Sottoscrizione(IdUtente, Tema, Data)$$
$$Notifica(IdPost, IdUtente, Data, Ora)$$
$$Amici(IdUtente1, IdUtente2)$$

Esercizio 02 *Si scriva una funzione che prende in ingresso il codice di un post e un parametro k intero e che restituisce in una stringa separati da virgole i codici dei k post più simili al post dato. Il grado di somiglianza tra due post è dato dal numero di parole chiave che hanno in comune.*

-- dato un post recupero tutte le parole colegate e mi prendo poi tutti gli id dei post che hanno quelle stesse parole. Raggruppo poi per idpost e conto
--Ordino per count decrescente e limito con k.
- uso il cursore per scorrere il risultato

```
SELECT P.idPost, COUNT(*) AS Num_simili
        FROM Presenza P JOIN Key AS K
              ON  P.Tema = K.Tema
        WHERE K.Parola IN (
                                SELECT K2.parola
                                      FROM Presenza P2 JOIN KEY K2
                                            ON P2.Tema = K2.Tema
                                      WHERE P2.idPost = InIdPost
                          )
        GROUP BY P.IdPost
        ORDER BY Num_simili DESC
        LIMIT by k
```

var. input

```
CREATE FUNCTION post_simili (InIdPost Post.idPost%TYPE, K INTEGER)
RETURN VARCHAR2(1000)
DECLARE
        my_cursor SYS_REF.CURSOR Post.idPost%TYPE, INTEGER

        risultato VARCHAR2(1000)
        comando VARCHAR(1000)
BEGIN
```

NOTA BENE: questa soluzione funziona ma è più corretto farla con SQL STATICO (vedi fine esercizio)

```
comando := 'SELECT P.idPost, COUNT(*) AS Num_simili FROM Presenza
                P JOIN Key AS K ON  P.Tema = K.Tema WHERE K.Parola IN (
                        SELECT K2.parola FROM Presenza P2 JOIN KEY K2
                        ON P2.Tema = K2.Tema WHERE P2.idPost =' || idPost
comando :=  ' ) GROUP BY P.IdPost ORDER BY Num_simili DESC LIMIT '
comando := comando || k ';'

OPEN my_cursor FOR comando
LOOP
      FETCH my_cursor INTO idPostC, K_c
            EXIT WHEN my_cursor%NOT_FOUND
            risultato = risultato || idPost_C || ', '
ENDLOOP
 RETURN RISULTATO
END
```

Perchè in questo caso possiamo usare SQL statico? Perchè
IL CURSORE dichiarato staticamente PUÒ FARE RIFERIMENTO AL
CONTENUTO di una variabile PURCHÈ SIA NEL SUO SCOPE.
La bind delle variabili viene fatta a RUNTIME NEL
MOMENTO IN CUI IL CURSORE VIENE APERTO. Questo implica però
che il valore assegnato al momento della open è STATICO e non
può essere variato durante la fetch dei risultati.
Per ulteriori dettagli ed esempi si veda il paragrafo
7.2.2.4 Variables in Explicit Cursor Queries della refernce guide di Oracle v 21.

```
CREATE FUNCTION post_simili (InIdPost Post.idPost%TYPE, K INTEGER)
RETURN VARCHAR2(1000)
DECLARE
      CURSOR my_cursor
      IS
      SELECT P.idPost, COUNT(*) AS Num_simili
      FROM Presenza P JOIN Key AS K
      ON P.Tema = K.Tema
      WHERE K.Parola IN (
            SELECT K2.parola
                  FROM Presenza P2 JOIN KEY K2
                  ON P2.Tema = K2.Tema
                  WHERE P2.idPost = InIdPost
)
GROUP BY P.IdPost
ORDER BY Num_simili DESC
LIMIT by K;

risultato VARCHAR2(1000)
comando VARCHAR(1000)
```

```
BEGIN
	FOR riga IN my_cursor
		LOOP
		risultato = risultato || riga.idPost_C || ', '
		ENDLOOP
RETURN risultato
END
```