Politecnico di Torino
Sistemi di gestione di basi di dati

8 Luglio 2011

1. (6 punti) Sono date le relazioni seguenti (le chiavi primarie sono sottolineate):

   UTENTE(UId, Nome, Cognome, Città, Nazione, Data_di_nascita)
   FOTO(FId, UId, Risoluzione, KBs)
   UPLOAD(FId, Data, Ora, Descrizione)
   TAG(FId, Tag)

   Si ipotizzino le seguenti cardinalità per le tabelle:

   - card(UTENTE)= 10^6 tuple,
     MIN(Data_di_nascita) = 1-1-1941, MAX(Data_di_nascita) = 31-12-1990,
     numero di Città $\simeq 10^2$,
     numero di Nazioni $\simeq 10$,
   - card(FOTO)= 10^8 tuple,
     numero di Risoluzioni $\simeq 10$,
     MIN(KBs) = 10^2, MAX(KBs) = $2 \cdot 10^3$,
   - card(UPLOAD)= $5 \cdot 10^8$ tuple,
     MIN(Data) = 01-01-2010, MAX(Data) = 31-12-2010,
   - card(TAG)= 10^9 tuple

   Inoltre si ipotizzi il seguente fattore di riduzione per la condizione di group by:

   - having count(distinct tag) $\leq$ 10 $\simeq \frac{99}{100}$.

   Si consideri la seguente query SQL:

```
select UId, Nome, Cognome
from UTENTE U, FOTO F
where U.UId=F.FId and Data_di_nascita > 1-1-1981
      and Città ='Roma'
      and FId NOT IN (select UP.FId
                      from UPLOAD UP, TAG T, FOTO F1
                      where UP.FId=T.FId and UP.FId=F1.FId
                      and Data ≥ 01-06-2010 and Data ≤ 30/06/2010
                      and Risoluzione <> '1280x720'
                      group by UP.FId
                      having count(distinct Tag) ≤10);
```

   Per l'interrogazione SQL

   (a) Si scriva l'espressione algebrica corrispondente, indicando le operazioni svolte, la cardinalità e la selettività di ogni operazione. Dove necessario, si ipotizzi la distribuzione dei dati. Discutere la possibilità di anticipare l'operatore GROUP BY.

   (b) Si scelgano le strutture fisiche accessorie per migliorare le prestazioni dell'interrogazione. Si motivi la scelta e si definisca il piano di esecuzione (ordine e tipo dei join, accesso alle tabelle e/o indici, etc.).

2. (7 Punti) Sono date le relazioni seguenti (le chiavi primarie sono sottolineate, gli attributi opzionali hanno l'asterisco):

```
PERSONA (Matricola, Mansione)
TURNO (Matricola, Data, CodT)
TIPO-TURNO (CodT, OraInizio, Durata)
RICHIESTA-FERIE(CodN, Matricola, Data)
NOTIFICA(Matricola, Data, EsitoRichiesta)
```

Scrivere il trigger per gestire le richieste di un giorno di ferie da parte delle persone che lavorano in un ospedale (inserimento nella tabella `RICHIESTA-FERIE`).

La richiesta di ferie viene accettata se la persona che la inoltra non è di turno nel giorno di ferie richiesto (tabella `TURNO`). Altrimenti, se la persona è di turno in quel giorno, la richiesta di ferie è accettata solo se esiste un'altra persona che può sostituirla per il turno. In caso contrario, la richiesta di ferie viene respinta. Una persona può sostituire un'altra persona per un turno in un certo giorno se ha la stessa mansione della persona che deve sostituire, e non è già di turno in quel giorno.

Deve essere notificato l'esito della richiesta (accettata o respinta) mediante un inserimento nella tabella `NOTIFICA-TURNO`.

3. Progettazione Data Warehouse

   Un'azienda internazionale di sviluppo software tiene traccia delle statistiche legate al codice sviluppato nelle sue sedi relativo alle diverse applicazioni che compongono la sua offerta commerciale.

   Ogni sede ha un proprio sistema di gestione del codice sviluppato (repository), dove gli sviluppatori condividono i progetti.

   Sul repository gli sviluppatori possono prelevare una copia dell'ultima revisione di codice disponibile e, dopo aver apportato le modifiche necessarie, aggiornano il repository con la nuova revisione di codice (operazione chiamata commit).

   Ad ogni commit, il repository tiene traccia di:

   - quale sviluppatore ha svolto l'operazione,
   - quante righe di codice sono state modificate, aggiunte e rimosse,
   - quali file sono stati cambiati,
   - quale tipo di modifica è stata apportata (può essere una combinazione di *bugfix*, *new feature*, *refactor* e *improvement*, ciascuno dei quali assume valore vero o falso),
   - data e ora dell'operazione.

   Inoltre ad ogni commit il repository assegna un numero di revisione univoco e crescente, che permette agli sviluppatori di sapere quando altri hanno aggiornato il codice. Eventuali conflitti per aggiornamenti concorrenti sono risolti localmente dagli sviluppatori prima di fare il commit sul repository, in modo che l'ultima revisione incorpori sempre tutte le modifiche precedenti.

   Gli sviluppatori sono raggruppati in team di sviluppo. Ogni sviluppatore appartiene a un solo team, e ogni team fa riferimento ad una specifica sede dell'azienda. Inoltre ogni sviluppatore ha un ruolo specifico, per esempio *tester*, *performance*, ecc.

   I vari progetti in sviluppo sono costituiti da diversi moduli separati. I file sono raggruppati in package, e ogni modulo è costituito da diversi package. Ogni applicazione nel catalogo dei prodotti sviluppati dall'azienda è costituita da un insieme di progetti. Ogni progetto è incluso in una sola applicazione. Gli sviluppatori lavorano su diversi progetti contemporaneamente (potenzialmente su tutti i progetti).

   Le varie revisioni del codice sono raggruppate in milestone. Le milestone, in base al grado di maturità raggiunto, possono essere divise in versioni stabili, release candidate, versioni beta, o build di sviluppo. Inoltre ogni milestone appartiene a una major release.

   L'azienda vuole creare un datawarehouse centrale per l'analisi dell'attività di sviluppo software, raccogliendo le informazioni disponibili nei database delle sue sedi. Deve essere possibile analizzare il numero totale di commit, il numero totale di righe di codice aggiunte, il numero totale di righe di codice rimosse, il numero totale di righe di codice modificate, e il numero medio di righe di codice cambiate (aggiunte, rimosse o modificate) per ogni commit in funzione di:

   - giorno della settimana (lunedì-domenica), giorno del mese (1-31), giorni feriali o festivi,
   - data, mese, bimestre, trimestre, anno,
   - sviluppatore, ruolo dello sviluppatore, team di sviluppo, sede e nazione in cui si trova la sede,
   - applicazione, progetto, modulo, package, file,
   - milestone, grado di maturità del codice, major release,
   - tipo di commit (*bugfix*, *new feature*, *refactor*, *improvement*).

   Nel data warehouse saranno contenuti i dati relativi agli anni 2006-2010 (inclusi). Sono inoltre note le seguenti statistiche (le informazioni ritenute necessarie ma non presenti in questa lista possono essere ipotizzate e stimate dal candidato):

   - l'azienda ha circa 2000 sviluppatori;

- un team è composto mediamente da 10 sviluppatori;
- gli sviluppatori sono suddivisi secondo 5 ruoli diversi;
- l'azienda ha 20 sedi in 10 nazioni;
- i repository delle varie sedi comprendono complessivamente circa 500 milestone di codice, 10 mila file, 1000 package, 500 moduli, 100 progetti e 20 applicazioni.

Sono riportate di seguito alcune delle interrogazioni frequenti di interesse per l'azienda, alle quali il datawarehouse deve poter rispondere in modo efficiente:

(a) Calcolare quante righe di codice sono state aggiunte in media al giorno dagli sviluppatori, separatamente per ogni ruolo dello sviluppatore, per ogni progetto e per ogni anno.

(b) Per ogni team e per ogni mese, calcolare in media quante righe di codice sono state aggiunte ad ogni commit, separatamente per ogni combinazione di tipo di commit (*bugfix*, ecc.).

(c) Mensilmente e separatamente per ogni team, calcolare la percentuale di commit effettuati da ogni sviluppatore rispetto al totale dei commit effettuati dal suo team. Considerare nel conteggio solo i commit che riguardano un *bugfix*.

Progettazione

(a) (7 Punti) Progettare il data warehouse in modo da soddisfare le richieste descritte nelle specifiche del problema. Il data warehouse progettato deve inoltre permettere di rispondere in modo efficiente a tutte le interrogazioni frequenti indicate.

(b) (4 Punti) Esprimere l'interrogazione frequente (a) utilizzando il linguaggio SQL esteso.

(c) (*Opzionale:* 5 Punti) Esprimere l'interrogazione frequente (c) utilizzando il linguaggio SQL esteso.