Basi di Dati e Sistemi Informativi I, Prove scritte AA 2010/11

Adriano Peron

Facoltà di Scienze M.F.N., Corso di Laurea in Informatica, Dipartimento di Scienze Fisiche,
Università di Napoli 'Federico II', Italy
E-mail: peron@na.infn.it

1 Scritto del 20 dicembre 2010 - Intercorso A

Si consideri il seguente schema relazionale che descrive operazioni di acquisto on-line con carrello (il cliente mette nel carrello gli articoli scelti; alla fine dell'operazione decide se procedere all'ordine indicando che il carrello è completo; quando il carrello è completo il carrello viene trasformato in ordine)
$CLIENTE(\underline{CodCl}, Nome, Cognome, Punti)$
$ORDINI(\underline{CodO}, CodCl, Data)$
$COMPORD(CodO, CodA, Quantita)$
$CARRELLO(CodO, Data, CodCl, Completo)$
$COMPCARRELLO(CodO, CodA, Quantita)$
$ARTICOLO(\underline{CodA}, Costo, Punti, Scorta)$.
$CLIENTE$, fornisce la descrizione del cliente (Punti fornisce i punti cumulati dal cliente per gli acquisti fatti); $ORDINI$ forisce la descrizione di un ordine completato (CodCl indica il codice cliente). $COMPORD$ indica la composizione dell'ordine (Codice dell'ordine, codice dell'articolo e quantità dell'articolo). $CARRELLO$ descrive un ordine non ancora consolidato (Completo è NULL se il carrello non è stato completato e NOT NULL altrimenti). $COMPCARRELLO$ indica la composizione del carrello (Codice dell'ordine, codice dell'articolo e quantità dell'articolo). $ARTICOLO$ descrive gli articoli in vendita (Codice, Costo unitario, Punti guadagnati per ogni singolo acquisto, scorta in magazzino dell'articolo).

Esercizio 11 *Si scriva una espressione in algebra relazionale (senza usare operazioni di conteggio) che, se valutata, fornisce il codice degli utenti che in tutti i loro ordini hanno comperato sempre almeno due articoli.*

Esercizio 12 *Si scriva una interrogazione SQL (preferibilmente senza uso di viste) che, se valutata, fornisce il codice degli utenti che (complessivamente nei vari ordini) hanno comperato tutti gli articoli disponibili.*

Almeno uno dei due seguenti esercizi.

Esercizio 13 *Si fornisca l'implementazione più ragionevole per i seguenti vincoli:*

1. *La scorta di un articolo non deve essere inferiore alla quantità complessiva dell'articolo presente nella composizione dei carrelli (ci possono essere più carrelli);*
2. *Ogni cliente può avere al più un carrello;*
3. *Gli ordini e i carrelli sono associati solo a clienti noti.*

Esercizio 14 *Si supponga che il carrello di codice 'YYY' venga completato. Il completamento del carrello ha il seguente effetto: il carrello e la sua composizione viene trasformato in ordine (conservando il codice) e rimosso dalla tabella dei carrelli; Le scorte di magazzino vengono aggiornate stornando i quantitativi indicati nel carrello. I punti del cliente vengono aggiornati con i punti acquisiti dall'acquisto).*

1. *Si indichi a parole quali operazioni vengo fatte e in che ordine.*
2. *Si scriva in SQL l'operazione per l'aggiornamento della scorta in magazzino.*

2 Scritto del 20 dicembre 2010 - Intercorso B

Si consideri il seguente schema relazionale che descrive operazioni di acquisto on-line con carrello (il cliente mette nel carrello gli articoli scelti; alla fine dell'operazione decide se procedere all'ordine indicando che il carrello è completo; quando il carrello è completo il carrello viene trasformato in ordine)
$CLIENTE(\underline{CodCl}, Nome, Cognome, Punti)$
$ORDINI(\underline{CodO}, CodCl, Data)$
$COMPORD(CodO, CodA, Quantita)$
$CARRELLO(CodO, Data, CodCl, Completo)$
$COMPCARRELLO(CodO, CodA, Quantita)$
$ARTICOLO(\underline{CodA}, Costo, Punti, Scorta)$.
$CLIENTE$, fornisce la descrizione del cliente (Punti fornisce i punti cumulati dal cliente per gli acquisti fatti); $ORDINI$ forisce la descrizione di un ordine completato (CodCl indica il codice cliente). $COMPORD$ indica la composizione dell'ordine (Codice dell'ordine, codice dell'articolo e quantità dell'articolo). $CARRELLO$ descrive un ordine non ancora consolidato (Completo è NULL se il carrello non è stato completato e NOT NULL altrimenti). $COMPCARRELLO$ indica la composizione del carrello (Codice dell'ordine, codice dell'articolo e quantità dell'articolo). $ARTICOLO$ descrive gli articoli in vendita (Codice, Costo unitario, Punti guadagnati per ogni singolo acquisto, scorta in magazzino dell'articolo).

Esercizio 21 *Si scriva una espressione in algebra relazionale (senza usare operazioni di conteggio) che, se valutata, fornisce il codice degli utenti che (complessivamente nei vari ordini) hanno comperato tutti gli articoli disponibili.*

Esercizio 22 *Si scriva una interrogazione SQL (preferibilmente senza uso di viste) che, se valutata, fornisce il codice degli utenti che in tutti i loro ordini hanno comperato sempre almeno due articoli.*

Almeno uno dei due esercizi.

Esercizio 23 *Si fornisca l'implementazione più ragionevole per i seguenti vincoli:*

1. *I punti di un cliente devono esere inferiori o uguali alla somma di tutti i punti cumulati dal cliente negli ordini del 2010;*
2. *Ogni cliente può fare al più un ordine al giorno;*
3. *Le composizioni di ordini e di carrelli sono associati solo ad articoli trattati in magazzino.*

Esercizio 24 *Si supponga che il carrello di codice 'YYY' venga completato. Il completamento del carrello ha il seguente effetto: il carrello e la sua composizione viene trasformato in ordine (conservando il codice) e rimosso dalla tabella dei carrelli; Le scorte di magazzino vengono aggiornate stornando i quantitativi indicati nel carrello. I punti del cliente vengono aggiornati con i punti acquisiti dall'acquisto).*

1. *Si indichi a parole quali operazioni vengo fatte e in che ordine.*
2. *Si scriva in SQL l'operazione per l'aggiornamento del punteggio del cliente.*

3 Scritto del 2 febbraio 2011

Per il compito intero si svolgano i primi quattro esercizi.
Per la seconda prova intercorso si svolgano gli ultimi tre esercizi.

Si consideri il seguente schema relazionale che descrive la strutturazione di un documento nelle sue parti (sezioni, sottosezioni, etc) e l'associazione tra parole chiave e le sezioni dove esse sono presenti.

$SEZIONE(\underline{Documento, CodSezione}, Titolo, Testo)$
$STRUTTURA(\underline{Documento, SezContenente}, SezContenuta, Posizione)$
$PAROLE(Parola)$
$PRESENZE(Parola, Documento, Sezione, Offset)$

$SEZIONE$ descrive le sezioni dei documenti (ogni sezione ha un titolo ed un testo. Una sezione può essere composta da sezioni. Tale composizione è descritta in $STRUTTURA$ dove SezContenente è il codice della sezione composta e SezContenuta è il codice della sottosezione e Posizione è una stringa di caratteri i che rappresenta un intero e che indica che SezContenuta è la i-esima sottosezione di SezContenente.
$PAROLE$ fornisce la lista delle parole interessanti (parole chiave) per il recupero delle sezioni. Se una parola chiave è presente in una sezione, è riportata una riga corrispondente in $PRESENZE$ (Sezione in $PRESENZE$ indica il codice di una sezione e offset indica la posizione della parola nella sezione; si ricorda che in ORACLE la funzione INSTR(stringa, pattern, [inizio, [occorrenza]]) restituisce, se presente, la posizione del pattern nella stringa)

Esercizio 31 *(7 punti) Si scriva una espressione dell'algebra relazionale che se valutata fornisca tutti i documenti in cui sono presenti tutte le parole chiave.*

Esercizio 32 *(7 punti) Si scriva una interrogazione SQL che che per ogni documento fornisca tutte le coppie di parole chiavi adiacenti nel documento (si trovano nella stessa sezione e non sono separate da altra parola chiave).*

Esercizio 33 *(7 punti) Si scriva una funzione PLSQL che riceve in ingresso il codice di due documenti e che restituisce un intero che rappresenta il numero di parole che i due documenti hanno in comune.*
Si utilizzi la funzione definita per scrivere una vista SIMILITUDINE(Doc1,Doc2,NumParoleComuni) che per ogni coppia di documenti Doc1 e Doc2 riporta il numero di parole in comune.

Esercizio 34 *(6 punti un trigger, 9 due trigger) Si scriva almeno uno dei due trigger seguenti. Si scriva un trigger che quando viene inserita una nuova parola chiave nella tabella $PAROLA$ provveda ad aggiornare la tabella $PRESENZE$ con la indicazione dei documenti, delle sezioni che contengono la parola (e i relativi offset). Si scriva inoltre un trigger che quando viene inserita una nuova sezione provveda ad aggiornare la tabella $PRESENZE$ relativamente a tutte le parole chiave presenti nella nuova sezione.*

Esercizio 35 *(Esercizio facoltativo, 10 punti) Si scriva una procedura PLSQL che riceve in ingresso una stringa di parole chiave del formato @parola1@parola2@... @parolak@. Usando SQL dinamico si scriva una interrogazione che trova i documenti che contengono tutte le parole indicate nella stringa (si ricorda che in SQL la funzione SUBSTR(stringa, pos, lungh) restituisce la porzione della sottostringa di stringa che*

ha inizio in pos ed ha lunghezza lungh). Il risultato della interrogazione deve essere restituito in una stringa.

4 Scritto del 2 febbraio 2011 - bis

Per il compito intero si svolgano i primi quattro esercizi.
Per la seconda prova intercorso si svolgano gli ultimi tre esercizi.

Si consideri il seguente schema relazionale che descrive la strutturazione di un documento nelle sue parti (sezioni, sottosezioni, etc) e l'associazione tra parole chiave e le sezioni dove esse sono presenti.

$SEZIONE(\underline{Documento, CodSezione}, Titolo, Testo)$
$STRUTTURA(\underline{Documento, SezContenente}, SezContenuta, Posizione)$
$PAROLE(Parola)$
$PRESENZE(Parola, Documento, Sezione, Offset)$

$SEZIONE$ descrive le sezioni dei documenti (ogni sezione ha un titolo ed un testo. Una sezione può essere composta da sezioni. Tale composizione è descritta in $STRUTTURA$ dove SezContenente è il codice della sezione composta e SezContenuta è il codice della sottosezione e Posizione è una stringa di caratteri i che rappresenta un intero e che indica che SezContenuta è la i-esima sottosezione di SezContenente.
$PAROLE$ fornisce la lista delle parole interessanti (parole chiave) per il recupero delle sezioni. Se una parola chiave è presente in una sezione, è riportata una riga corrispondente in $PRESENZE$ (Sezione in $PRESENZE$ indica il codice di una sezione e offset indica la posizione della parola nella sezione; si ricorda che in ORACLE la funzione INSTR(stringa, pattern, [inizio, [occorrenza]]) restituisce, se presente, la posizione del pattern nella stringa)

Esercizio 41 *(10 punti) Si scriva una funzione PLSQL che riceve in ingresso il codice di due documenti e che restituisce un intero che rappresenta il numero di parole che i due documenti hanno in comune.*
Si utilizzi la funzione definita per scrivere una vista SIMILITUDINE(Doc1,Doc2,NumParoleComuni) che per ogni coppia di documenti Doc1 e Doc2 riporta il numero di parole in comune.

Esercizio 42 *(10 punti un trigger, 18 due trigger) Si scriva almeno uno dei due trigger seguenti. Si scriva un trigger che quando viene inserita una nuova parola chiave nella tabella $PAROLA$ provveda ad aggiornare la tabella $PRESENZE$ con la indicazione dei documenti, delle sezioni che contengono la parola (e i relativi offset). Si scriva inoltre un trigger che quando viene inserita una nuova sezione provveda ad aggiornare la tabella $PRESENZE$ relativamente a tutte le parole chiave presenti nella nuova sezione.*

Esercizio 43 *(Esercizio facoltativo, 15 punti) Si scriva una procedura PLSQL che riceve in ingresso una stringa di parole chiave del formato @parola1@parola2@... @parolak@. Usando SQL dinamico si scriva una interrogazione che trova i documenti che contengono tutte le parole indicate nella stringa (si ricorda che in SQL la funzione SUBSTR(stringa, pos, lungh) restituisce la porzione della sottostringa di stringa che ha inizio in pos ed ha lunghezza lungh). Il risultato della interrogazione deve essere restituito in una stringa.*

5 Scritto del 2 marzo 2011

Si consideri il seguente schema relazionale che descrive una versione elementare dei dati per un social network
$Utenti(\underline{id}, nome, cognome, mail, passwd)$
$Amici(Utente1, Utente2, Data)$
$Interessi(Utente, Categoria, Valore)$
$Richiesta(Richiedente, Invitato, Tipo, Data, Stato)$
$Post(\underline{codice}, Utente, testo, data)$
$Commento(codicePost, Utente, testo, data)$.

Amici descrive la relazione simmetrica di amicizia tra utenti e la data in cui è stata stabilita; *Interessi* descrive gli interessi di un utente mediante la categoria di interesse (ad es. Sport) e il tipo di interesse (ad es. valore tennis). *Richiesta* descrive le richieste inoltrate dall'utente Richiedente all'utente Invitato, il tipo richiesta (ad. es Amicizia), lo stato della richiesta (attesa,accettata); un elemento di *Post* indica un elemento testuale pubblicato da un utente; un elemento di *Commento* è la risposta di un utente al post individuato da codicePost.

Esercizio 51 *(8 punti) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce il nome degli utenti che hanno scritto post che hanno ricevuto sempre almeno 5 commenti.*

Esercizio 52 *(8 punti) Si implementino in modo opportuno i seguenti vincoli:*

1. *quando una richiesta di amicizia viene accettata la relazione Amici viene aggiornata stabilendo in modo simmetrico la relazione di amicizia.*
2. *Un utente non può commentare più di una volta lo stesso post;*
3. *Lo stato di una richiesta può avere solo o valore Attesa o valore Accettato o valore Rifiutato;*
4. *Non ci può essere una relazione di amicizia tra due utenti senza una richiesta di amicizia accettata.*

Esercizio 53 *(10 punti) Si scriva una funzione che prende in ingresso il codice di un utente ed un numero intero k. La funzione deve recuperare le catene di amici fino al grado k (di grado 1 sono gli amici ufficiali, di grado 2 gli amici degli amici ect.). Ci si avvalga di una tabella temporanea TMP che si suppone già creata. Una volta recuperati gli amici la funzione deve restituirli in una stringa in ordine crescente di grado di amicizia e, per gli amici dello stesso grado, in ordine lessicografico per cognome e nome.*

Esercizio 54 *(10 punti) Usando sql dinamico si scriva una funzione che riceve in ingresso una stringa di lunghezza variabile di caratteri del tipo ?categoria1?valore1?categoria2?valore2?.... e che restituisca in una stringa gli identificativi degli utenti separati da virgole che hanno interessi che soddisfano almeno una delle coppie ?categoria?valore? (si usino le funzioni INSTR e SUBSTR per la scansione della lista di ingresso).*

6 Scritto del 25 marzo 2011

Si consideri il seguente schema relazionale che descrive un database per la gestione di condivisione di file in uno schema peer to peer.
$UTENTE(\underline{UsrId}, MaxUp, MaxDown, Stato)$
$FILE(IdFile, IdUtente)$
$BLOCCHI(IdFile, IdUtente, IdBlocco)$
$SCARICO(IdFile, UsrUp, UsrDown, IdBlocco)$
$RICHIESTA(IdFile, UsrDown, Time, IdBlocco)$

$UTENTE$ formisce il numero massimo di operazioni di Upload e Download simultaneamente possibili per un utente e il suo stato che può assumere lo stato On o Off. $FILE$ indica i file completi di cui gli utenti sono in possesso. Un utente può essere in possesso di un file completo (in questo caso si ha un record in FILE) oppure può essere in possesso solo di alcuni blocchi del file e non di tutti (in questo caso per ogni blocco posseduto si ha un record in BLOCCHI). $SCARICO$ indica quali sono le operazioni di scarico in corso con indicazione dell'utente che fa l'upload ed il download e il blocco in via di trasferimento. L'unità di trasferimento il blocco. $RICHIESTA$ indica le richieste in attesa di scarico, con Time il tempo in cui la richiesta è stata inoltrata.

Esercizio 61 *(punti 8) Si scriva una espressione dell'algebra relazionale che se valutata fornisca lo userid degli utenti che hanno il maggior numero di download in corso.*

Esercizio 62 *(punti 8) Si scriva una vista SQL che per ogni utente fornisca il numero di upload e download in corso, il numero di file in attesa di dowload e il tempo massimo di attesa.*

Esercizio 63 *(punti 10) Quando un utente commuta il suo stato da Off a On i file o i blocchi di file che possiede divengono disponibili. Di conseguenza un trigger cerca di attivare il maggior numero di operazioni di scarico presenti nella tabella CODA usando le disponibilità dell'utente secondo i seguenti criteri: Hanno precedenza le richieste più vecchie; le richieste riguardano file posseduti dall'utente che ha fatto accosso (si osservi che se un utente possiede un intero file mette a disposizione tutti blocchi del file); il numero di operazioni di carico attivate non può superare il MaxUp dell'utente; una richiesta non può essere attivata se l'utente ha già in corso un numero di download pari al numero massimo a lui consentito.*
Attivare una richiesta significa inserire una riga corrispondente nella tabella $SCARICO$ e rimuovere la richiesta dalla tabella CODA. Scrivere il trigger in questione.

Esercizio 64 *(punti 8) Utilizzando SQL dimamico si scriva una funzione il cui scopo è quello di fornire in modo parametrico dei conteggi su file (Idfile). La funzione richiede tre parametri. Il primo è un Idfile, il secondo il nome di una tabella (valori possibili FILE, BLOCCHI, SCARICO, RICHIESTA), il terzo il nome di un attributo della tabella passato nel secondo parametro. La funzione restituisce il valore di una COUNT(DISTINCT) applicata all'attributo passato sul secondo parametro della tabella passata nel primo parametro per l'dentificativo di file passato nel primo parametro. Ad esempio, se vengono passati i tre parametri attuali $Id-50$, $SCARICO$, $USRUP$ la funzione restituisce il valore degli upload distinti (COUNT DISTINCT su UsrUp) sulla tabella $SCARICO$ per lo $IdFile = Id-50$.*

7 Scritto del 28 maggio 2011

Si consideri il seguente schema relazionale che descrive la strutturazione di classi in un class diagram di UML.

$CLASS(\underline{Nome}, Descrizione, DefVal)$
$REFINE(SuperClass, SubClass)$
$ATTRIBUTI(Class, Nome, Tipo)$
$BASICTYPE(Nome, DefVal)$
$OBJ(\underline{Oid}, NomeClass)$
$OBJATT(\underline{Oid, NomeAtt}, ValBasic, ValObj)$.

$CLASS$ descrive le classi: l'attributo DefVal contiene l'oid di una istanza di default per la classe (chiave esterna verso la tabella OBJ). $REFINE$ descrive la relazione di specializzazione delle classi (SubClass è la specializzazione della classe SuperClass); $ATTRIBUTI$ indica quali attributi vengono associati ad una classe. Il tipo di un attributo può essere di tipo basico (contenuto nello schema $BASICTYPE$ dove l'attributo $defVal$ indica il valore di default per il tipo basico) oppure il nome di una classe. OBJ descrive le istanze delle classi (oggetti) con Oid l'identificativo dell'istanza. I valori degli attributi associati agli oggetti sono memorizzati in $OBJATT$: se l'attributo ha tipo basico allora $ValBasic$ è non vuoto (ValObj è invece vuoto) e contiene il valore dell'attributo; se l'attributo ha come tipo una classe allora $ValBasic$ è vuoto e ValObj contiene l'oid di un oggetto.

Esercizio 71 *(7 punti) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce il nome delle classi non specializzate che prevedono solo attributi di tipo BASICTYPE.*

Esercizio 72 *(8 punti) Si scriva una interrogazione SQL che recupera coppie di nomi di classi in modo che per ogni attributo della seconda classe vi sia un attributo della prima classe che abbia lo stesso nome e lo stesso tipo.*

Esercizio 73 *(11 punti) Si scriva un metodo PLSQL che riceve in ingresso il nome di una classe e un nuovo identificativo di oggetto e che crea la istanziazione di un oggetto default per quella classe. L'oggetto creato dovr essere inserito nelle opportune tabelle (OBJ e OBJATT). Per gli attributi di tipo basico si assoceranno agli attributi i valori di default in dicati in BASICTYPE. Per gli attributi di tipo tipo classe si assoceranno agli attributi i valori di default in dicati in CLASS. Si presti attenzione al fatto che se la classe da istanziare risulta essere il raffinamento di un'altra classe anche gli attributi non sovrascritti di tutte le classi antenate (fino alla radice della gerarchia di specializzazione) vanno istanziati.*

Esercizio 74 *(8 punti) Si scriva usando SQL dinamico una procedura PSQL che riceve in ingreasso il nome di una classe e una stringa del formato 'NomeAttributo1@Valoreattributo1@NomeAttributo2@Valoreattributo2@...' (numero arbitrario di coppie attibuto valore attributo). L'effetto della procedura di restituire il numero di oggetti di quella classe per cui per tutte le coppie valga nomeattributo = valoreattributo.*

(si ricorda che in PSQL la funzione SUBSTR(stringa, pos, lungh) restituisce la porzione della sottostringa di stringa che ha inizio in pos ed ha lunghezza lungh e che

INSTR(instringa,stringa,pos) restituisce la prima posizione di instringa in cui compare a partire da pos la prima occorrenza della sottostringa stringa).

8 Scritto del 29 giugno 2011

Si consideri il seguente schema relazionale che descrive un database contenente domande per test a risposta multipla, test e risultati di somministrazione di test.

$QUESITO(\underline{CodQ}, Testo, Tipo, Livello)$
$RISPOSTE(\underline{CodQ, CodR}, Risposta, Ok)$
$TEST(\underline{CodT}, Data, Luogo, PuntE, PuntC, PuntN, Elab)$
$COMPTEST(codT, CodQ)$
$RISULTATO(codT, Utente, CodQ, CodR)$
$VALUTAZIONE(Utente, CodT, Punti)$
$STORICO(Utente, NTest, Punteggio)$

Lo schema $QUESITO$ descrive domande con risposte a scelta multipla. Ogni domanda appartiene ad una tipologia ed ha associato un livello di difficoltà. Le risposte multiple alle domande si trovano in $RISPOSTE$: l'attributo *ok* assume valori $\{0, 1\}$ (0 per la risposta sbagliata e 1 per quella corretta). Ogni domanda ha più risposte e tra queste solo una è corretta. Lo schema $TEST$ descrive i test effettuati (un test è una collezione di domande somministrata a uno o più utenti). PuntE indica i punti per una risposta errata, PuntC i punti per una risposta corretta, PuntN i punti per una risposta non data, Elab un flag che assume valori $\{0, 1\}$ e che indica se i risultati del test sono stati elaborati.

La scelta delle domande presenti in un test è indicata nella tabella $COMPTEST$. Nello schema $RISULTATO$ si ha descrizione della risposta data da un utente per una domanda in un test. Se l'utente non ha fornito risposta ad una domanda presente in un test l'attributo $CodR$ assume valore nullo. La tabella $VALUTAZIONE$ contiene il punteggio conseguito da un utente in un test. La tabella $STORICO$ tiene traccia del numero di test sostenuto da ciascun utente e del punteggio medio conseguito nei test.

Esercizio 81 *(7 punti) Si scriva una interrogazione in algebra relazionale che fornisce coppie codice test - utente relativi a test per i quali l'utente ha fornito tutte le risposte correttamente.*

Esercizio 82 *(7 punti) Si scriva una interrogazione SQL fornisce coppie codice test - utente relativi a test per i quali l'utente ha fornito il maggior numero di risposte corrette (rispetto agli utenti che hanno sostenuto lo stesso test).*

Esercizio 83 *(9 punti) Si scriva un trigger SQL che elabora i risultati del test quando il flag Elab passa da 0 a 1. IL trigger deve calcolare il punteggio conseguito da ciascun utente nel test e memorizzarlo nella tabella $VALUTAZIONE$. La tabella $STORICO$ deve essere aggiornata di conseguenza.*

Esercizio 84 *(9 punti) Si scriva usando SQL dinamico una procedura PSQL che riceve in ingresso il codice di un test e una stringa contenente un numero variabile di codici di domande che debbono comporre il test (i codici di domande sono separati dal carattere @, ad es 'Cod1@Cod2@Cod3...'). La procedura elabora la stringa creando un'unica istruzione INSERT che inserisce le coppie codice test - codice domanda nella tabella COMPTEST e la amnda in esecuzione.*

(si ricorda che in PSQL la funzione SUBSTR(stringa, pos, lungh) restituisce la porzione della sottostringa di stringa che ha inizio in pos ed ha lunghezza lungh e che

INSTR(instringa,stringa,pos) restituisce la prima posizione di instringa in cui compare a partire da pos la prima occorrenza della sottostringa stringa).

9 Scritto del 21 luglio 2011

Si consideri il seguente schema relazionale che descrive un frammento della base di dati per gestire un gioco da schacchiera

$Caselle(\underline{CodC}, X, Y)$
$Inizio(Partita, Pezzo, CodC)$
$Mosse(\underline{Partita, Ordine}, Pezzo, Partenza, Arrivo, Ultima)$
$Stop(Partita, Pezzo, Ordine)$

Caselle descrive le coordinate delle caselle della scacchiera occupabili dalle pedine. *Inizio* tiene traccia della posizione iniziale delle pedine in una partita. *Mosse* memorizza le sequenze (ordinate secondo l'attributo *Ordine*) di mosse in una partita. Per ogni mossa viene indicata la casella di partenza e quella di arrivo. L'attributo ultima (con valori 0,1) indica se la mossa rappresenta l'ultima mossa della partita. $Stop(Partita, Pezzo, Ordine)$ tiene traccia delle pedine eliminate (ordine indica la mossa che ha portao all'eliminazione).

Esercizio 91 *Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce codice di partita e pezzo per pezzi che nella partita hanno fatto solo o mosse orizzontali o mosse verticali.*

Esercizio 92 *Si supponga che oltre agli schemi già elencati vi sia un altro schema riassuntivo per ogni partita con la seguente struttura:*
$Storico(Partita, Pezzo, NMosse, NCaselle)$ *dove* $NMosse$ *indica il numero di mosse del pezzo nella partita e* $NCaselle$ *indica indica il numero di caselle diverse attraversate dal pezzo. Quando viene inserita l'ultima mossa della partita, un trigger aggiorna lo schema Storico con i dati della partita. Scrivere il trigger.*

Esercizio 93 *Si scriva ua procedura PLSQL che prende in ingresso una partita e un numero d'ordine di mossa N e restituisca in uscita la situazione della scacchiera dopo che si sono giocate le prime N mosse della partita. La situazione della scacchiera vine descritta mediante una stringa che contiene una sequenza di terne -X,Y,pezzo- (separate da un carattere separatore) ad indicare quali sono le caselle occupate dai pezzi (le caselle vuote non vengono elencate nella stringa).*

Esercizio 94 *Si scriva una procedura PLSQL che riceve in ingresso il nome per una nuova tabella e una stringa di coppie attributo - tipo (della forma attributo1@tipo1@attributo2@tipo2..... @attributok@tipok). Usando SQL dinamico si scriva un comando (e lo si esegua) per la creazione della tabella che abbia come nome il nome indicato nel primo parametro e come attributi quelli indicati nella stringa passata come secondo parametro. 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.*

10 Scritto del 3 marzo 2010

Si consideri il seguente schema relazionale che descrive un frammento della base di dati per gestire un sistema di posta elettronica

$User(\underline{email}, Nome, Cognome, Note)$
$Mail(\underline{Idmail}, Timestamp, Stato, Oggetto, Testo, Dimensione)$
$Legame(Idmail, utente, tipo)$
$Folder(\underline{IdFolder}, Utente, Nome, Dimensione, Contenente)$
$Contiene(IdFolder, Idmail)$

User descrive gli utenti noti che possono essere o mittenti o destinatari di una email. *Mail* descrive il messaggio di posta ricevuto: stato indica se il messaggio è stato letto (READ), non è stato letto (UNREAD); dimensione indica la dimensione in byte del messaggio. *Legame* indica gli utenti interessati ad una email; *tipo* indica se l'utente è mittente (FROM), destinatario (TO), per conoscenza (CC); un messaggio ha un solo mittente ma possibilmente più destinatari. *Folder* indica il nome di un folder di un utente dove sono organizzate i messaggi: un forder appartiene ad un utente; fa parte di una organizzazione ad albero (directory/sottodirectory come in un file system); Contenente indica il folder contenete (se il folder è la radice il valore è NULL); Dimensione indica la dimensione cumulata di tutti i messaggi contenuti nel folder e nei suoi sottofolder. *Contiene* associa i messaggi ai folder.

Esercizio 101 *Si scriva una espressione in algebra relazionale che se valutata fornisca coppie di utenti mittente - destinatario tali che l'utente mittente abbia inviato dei messaggi al destinatario ma il destinatario non abbia mail letto nessuno dei messaggi inviati da quel mittente.*

Esercizio 102 *Si scriva una vista che raccolga la seguente informazione aggregata relativa ai messaggi inviati/ricevuti nel solo mese corrente (si usi SYSDATE per la data corrente). Per ogni utente, il numero di messaggi inviati, il numero di messaggi ricevuti, il numero medio di messaggi ricevuti per utente mittente.*

Esercizio 103 *Si scriva un trigger che viene attivato quando viene aggiunto un messaggio ad un folder (tabella Contenente). L'effetto previsto è quello di incrementare la dimensione del folder di una quantità pari alla dimensione del messaggio il folder che lo contiene e la dimensione di tutti i folder (a risalire fino alla radice della gerarchia di contenimento) che lo contengono.*

Esercizio 104 *Si scriva una procedura PLSQL che riceve in ingresso il nome di due tabelle Tab1 e Tab2, il nome di un vincolo Vinc, e una stringa di coppie di attributi di Tab1 e Tab2, rispettivamente, (della forma attr1Tab1@attr1Tab2@attr2Tab1@attr2Tab2..... @attrkTab1@attrkTab2). L'idea è che in Tab2 deva essere inserita un vincolo di chiave esterna per gli attributi (attr1Tab2, ..., attrkTab2) che referenziano la chiave primaria (attr1Tab1, ..., attrkTab1) di Tab1. Usando SQL dinamico si scriva un comando (e lo si esegua) che implementi le richieste di cui sopra. 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.*

11 Scritto del 24 novembre 2011