RISPOSTE DOMANDE BASI TEMI ESAMI PASSATI

DEFINIZIONE DI CONFLICT-SERIALIZZABILITA':

Uno schedule S e' conflict serializzabile se esiste uno schedule S' che e' conflict-equivalente ad S.

Due schedule sono conflict-equivalenti se possiedono le stesse operazioni e gli stessi conflitti e le operazioni in conflitto sono nello stesso ordine nei due schedule.

INDICE PRIMARIO DENSO: STRUTTURA, RICERCA, INSERIMENTO E CANCELLAZIONE

L'indice primario denso e' una struttura ad accesso dati che aumenta la prestazione negli accessi alle tuple memorizzate in memoria secondaria.

Nell'indice primario la chiave di ricerca e' la chiave di ordinamento (tipicamente la chiave primaria)

Ogni indice ha un record dove per ogni chiave viene associato un puntatore alla prima occorenza della tupla con quella chiave.

Nell'indice primario denso per ogni occorrenza della chiave presente nel file esiste un corrispondente record nell'indice.

RICERCA DI UNA TUPLA CON CHIAVE DI RICERCA K:

Nell'indice primario k e' sicuramente presente tra gli indici quindi accedo al record negli indici con chiave k e accedo al file tramite il puntatore associato pk e scorro nelle tuple del blocco puntato finche' non trovo la tupla che cercavo. Il costo e' 1 accesso indice + 1 accesso blocco dati.

INSERIMENTO DI UN RECORD NELL'INDICE

L'inserimento nell'indice avviene solo se la tupla inserita nel file ha un valore di chiave k che non è già presente.

CANCELLAZIONE DI UN RECORD NELL'INDICE

La cancellazione nell'indice avviene solo se la tupla cancellata nel file è l'ultima tupla con valore di chiave k.

DEFINIZIONE DI VIEW-EQUIVALENZA

Uno schedule S si dice view-equivalente ad uno schedule S' se questi hanno lo stesso insieme di relazioni *legge_da* e lo stesso insieme di *scritture_finali.*

INDICE B+-TREE: STRUTTURA, RICERCA TUPLA CON CHIAVE K, INSERIMENTO, CANCELLAZIONE

Il B+-tree è una struttura di dati ad albero bilanciato utilizzata comunemente nei sistemi di database per gestire grandi quantità di dati.

L'albero è sempre bilanciato, il che garantisce che l'altezza rimanga logaritmica rispetto al numero di chiavi, permettendo operazioni efficienti di ricerca, inserimento e cancellazione.

Nei B+-tree, i nodi interni contengono solo chiavi che guidano la ricerca, mentre le foglie contengono le chiavi effettive e i puntatori ai dati.

Le foglie sono collegate tra loro, consentendo una scansione sequenziale efficiente dei dati.

Ricerca di una tupla con chiave K

l'algoritmo parte dalla radice e confronta K con le chiavi presenti nel nodo corrente. In base a questo confronto, decide quale ramo seguire, scendendo progressivamente attraverso i nodi interni fino a raggiungere un nodo foglia. Una volta raggiunto il nodo foglia, si cerca linearmente K al suo interno. Se la chiave è trovata, si restituisce il puntatore al record di dati; se non è trovata, la ricerca termina senza successo.

Inserimento

L'inserimento in un B+-tree segue un processo di ricerca simile per trovare il nodo foglia appropriato dove inserire la nuova chiave. Se il nodo foglia ha spazio, la chiave viene inserita direttamente. Se il nodo è pieno, si esegue uno split: il nodo viene diviso in due nodi.

Se necessario(se supero il vincolo di riempimento dei puntatori del nodo intermedio) lo split si propaga al livello superiore, mantenendo l'albero bilanciato.

Cancellazione

La cancellazione di una chiave da un B+-tree inizia anche essa con la ricerca della chiave. Una volta trovata, se la chiave si trova in un nodo foglia, viene semplicemente rimossa.

Se la rimozione di una chiave causa una violazione del vincolo di riempimento minimo di chiavi in un nodo foglia si esegue un'operazione di merge con un nodo fratello per mantenere l'albero bilanciato, se facendo il merge vengono violati altri vincoli (anche nei nodi intermedi), si ridistrbuiscono le chiavi opportunamente con merge/split in modo che vengano rispettati i vincoli di riempimento minimo e massimo.

ILLUSTRARE 2PL STRETTO

Il protocollo di locking a due fasi (2PL) è un meccanismo per la gestione della concorrenza nei sistemi di database, che si basa sull'uso di lock per controllare l'accesso alle risorse durante le operazioni di lettura e scrittura delle transazioni. Ogni risorsa può essere in uno dei seguenti stati: libera, lettura lock (r_lock) o scrittura lock (w_lock). Le transazioni possono acquisire un r_lock su una risorsa per consentire letture concorrenti, ma un w_lock per la scrittura impedisce l'accesso concorrente da parte di altre transazioni.

Il 2PL garantisce la serializzabilità, ovvero l'equivalenza dell'esecuzione delle transazioni a una esecuzione seriale, imponendo che una transazione non può acquisire nuovi lock dopo averne rilasciato uno.

Questo comporta che ogni transazione passa attraverso due fasi: una fase di acquisizione dei lock e una fase di rilascio dei lock.

Nel caso del 2PL stretto, si aggiunge un ulteriore livello di sicurezza richiedendo che una transazione mantenga tutti i lock acquisiti fino al completamento della transazione stessa, cioè fino al commit o al rollback.

Questo assicura che nessuna altra transazione possa vedere dati parzialmente aggiornati, prevenendo inconsistenze dovute a operazioni non confermate.

DEFINIZIONE DI VIEW-SERIALIZZABILITA'

Uno schedule S e' view-serializzabile se esiste uno schedule seriale S' che e' view-equivalente allo schedule S.

RELAZIONE TRA VSR E CSR

Relazione tra VSR e CSR: L'insieme degli schedule conflict-serializzabili (CSR) è un sottoinsieme degli schedule view-serializzabili (VSR). Questo significa che ogni schedule CSR è anche VSR, ma non tutti gli schedule VSR sono CSR.

Dimostrazione della relazione CSR ⊆ VSR:

1. **Osservazione preliminare:** La conflict-serializzabilità implica la view-serializzabilità.

Controesempio per la non necessità: Consideriamo lo schedule: r1(x) w2(x) w1(x) w3(x)
LEGGE_DA = ∅
SCR_FIN = {w3(x)}
r1(x) w1(x) w2(x) w3(x)
LEGGE_DA = ∅
SCR_FIN = {w3(x)}

Questo schedule è view-serializzabile perché view-equivalente allo schedule seriale:
Tuttavia, non è conflict-serializzabile perché il grafo dei conflitti è ciclico.

Dimostrazione che CSR ⇒ VSR:
Supponiamo che S1 ≈C S2 e dimostriamo che S1 ≈V S2. Poiché S1 ≈C S2, i due schedule hanno:
Stesse scritture finali: Se non fosse così, ci sarebbero almeno due scritture sulla stessa risorsa in ordine diverso, e quindi non sarebbero ≈C.
Stessa relazione "legge_da": Se non fosse così, ci sarebbero scritture o coppie lettura-scrittura in ordine diverso, violando la ≈C.

Quindi, ogni schedule conflict-serializzabile è anche view-serializzabile, dimostrando che CSR ⊆ VSR.

INDICE SECONDARIO: STRUTTURA, RICERCA CON CHIAVE K

L'indice secondario e' una struttura ad accesso dati che aumenta la prestazione negli accessi alle tuple memorizzate in memoria secondaria.

In questo caso invece la chiave di ordinamento e la chiave di ricerca sono diverse.

Ogni indice ha un record dove per ogni chiave viene associato un puntatore alla prima occorenza della tupla con quella chiave.

Nell'indice secondario per ogni occorrenza della chiave presente nel file esiste un corrispondente record nell'indice.

RICERCA DI UNA TUPLA CON CHIAVE DI RICERCA K:

Nell'indice primario k e' sicuramente presente tra gli indici quindi accedo al record negli indici con chiave k e accedo al file tramite il puntatore associato pk e scorro nelle tuple del blocco puntato finche' non trovo la tupla che cercavo. Il costo e' 1 accesso indice + 1 accesso blocco dati.

ILLUSTRARE L'ALGORITMO DI RIPRESA A CALDO, INDICANDO ANCHE QUANDO VIENE UTILIZZATO E DA QUALE MODULO DEL DBMS VIENE ESEGUITO.

L'algoritmo di ripresa a caldo viene eseguito dal **gestore dell'affidabilità** del DBMS, il modulo responsabile di garantire la consistenza e l'integrità dei dati in caso di guasti. Il gestore dell'affidabilità include i seguenti sottosistemi principali:

1. **Modulo di Gestione del Log**: Questo modulo è responsabile della registrazione di tutte le operazioni eseguite sul database, compresi gli inizi (B(T)) e i commit (C(T)) delle transazioni. Il log serve come registro persistente delle modifiche, fondamentale per il ripristino a caldo.
2. **Modulo di Ripristino**: Utilizza le informazioni nel log per eseguire il ripristino del database. Implementa l'algoritmo di ripresa a caldo per garantire che il database ritorni a uno stato coerente dopo un guasto. Questo modulo svolge le operazioni di undo e redo necessarie per recuperare le transazioni attive e committate.

Ecco un riepilogo dettagliato dell'algoritmo di ripresa a caldo:

Algoritmo di Ripresa a Caldo

Passaggi dell'Algoritmo

1. **Accesso all'Ultimo Blocco del LOG**:
   - Il gestore dell'affidabilità accede all'ultimo blocco del log e ripercorre all'indietro fino a trovare il record di checkpoint (CK) più recente. Il checkpoint rappresenta uno stato salvato del database, utile per limitare l'analisi del log necessaria per il ripristino.
2. **Identificazione delle Transazioni**:
   - Il gestore dell'affidabilità identifica le transazioni da rifare o disfare, inizializzando due insiemi: **UNDO** e **REDO**.
     - **UNDO**: Popolato con le transazioni attive al momento del checkpoint.
     - **REDO**: Inizialmente vuoto.
3. **Scansione del LOG in Avanti**:
   - Il gestore dell'affidabilità scansiona il log in avanti, dal checkpoint fino al momento del guasto. Durante questa scansione:
     - Per ogni record **B(T)** (inizio di una transazione T), T viene aggiunto all'insieme **UNDO**.
     - Per ogni record **C(T)** (commit di una transazione T), T viene spostato dall'insieme **UNDO** all'insieme **REDO**.
4. **Disfarsi delle Operazioni delle Transazioni UNDO**:
   - Il gestore dell'affidabilità ripercorre all'indietro il file di log dal momento del guasto fino alla prima azione della transazione più vecchia. Durante questa scansione, si disfano (undo) le operazioni eseguite dalle transazioni presenti nell'insieme **UNDO**.
5. **Rifare le Operazioni delle Transazioni REDO**:
   - Una volta completata la fase di undo, il gestore dell'affidabilità rifà (redo) le operazioni delle transazioni presenti nell'insieme **REDO**. Questa operazione garantisce che tutte le modifiche effettuate dalle transazioni committate siano effettivamente applicate al database.

Scenari di Utilizzo

L'algoritmo di ripresa a caldo è utilizzato nei seguenti scenari:

- Dopo un arresto anomalo del sistema o un guasto del DBMS.
- In seguito a una perdita di potenza che causa un'interruzione del servizio.
- In caso di errori hardware o software che richiedono il ripristino dello stato coerente del database.

In sintesi, il gestore dell'affidabilità, tramite i moduli di gestione del log e di ripristino, garantisce che il database possa recuperare rapidamente e correttamente in caso di guasti, mantenendo l'integrità e la consistenza dei dati.

ILLUSTRARE L'ALGORITMO DI ORDINAMENTO Z-WAY SORT-MERGE.

Lo Z-way Sort-Merge è un algoritmo di ordinamento utilizzato nei database per gestire grandi quantità di dati. Questo metodo impiega una tecnica di suddivisione e fusione delle tuple per ordinare i risultati di un'interrogazione.

Descrizione dello Z-way Sort-Merge

1. **Fase di Sort Interno**:
   - Le tuple vengono divise in gruppi più piccoli chiamati "run".
   - Ogni run è ordinato individualmente utilizzando un algoritmo di sort interno, come il QuickSort.
   - Ogni run ordinato viene scritto su memoria secondaria in un file temporaneo.
2. **Fase di Merge**:
   - I run ordinati vengono uniti attraverso uno o più passaggi di fusione.
   - Durante ogni passaggio di fusione, due run vengono letti contemporaneamente e mescolati in un unico run ordinato.
   - Questo processo continua fino a quando non si ottiene un'unica run contenente tutte le tuple ordinate.

ILLUSTRARE LE PROPRIETÀ DELLE TRANSAZIONI; SI INDICHI INOLTRE QUALI MODULI DI UN DBMS GARANTISCONO CIASCUNA DI TALI PROPRIETÀ.

Le proprietà delle transazioni sono queste quattro proprietà chiamate acide:

A→ATOMICITA': UNA TRANSAZIONE E' UN'UNITA' DI LAVORO INDIVISIBILE O VIENE ESEGUITA COMPLETAMENTE O NON VIENE ESEGUITA.
GARANTITA DAL GESTORE DEI METODI DI ACCESSO, DAL GESTORE DELL'AFIDABILITA' E DAL GESTORE DEL BUFFER

C→CONSISTENZA: L'ESECUZIONE DI UNA TRANSAZIONE NON DEVE VIOLARE I VINCOLI DI INTEGRITA'.

GARANTITA DAL GESTORE DEI METODI DI ACCESSO,DAL GESTORE DELLA CONCORRENZA, DAL GESTORE DELLE INTERROGAZIONI.

I→ISOLAMENTO: L'ESECUZIONE DI UNA TRANSAZIONE DEVE ESSERE INDIPENDENTE DALLA CONTEMPORANEA ESECUZIONE DI ALTRE TRANSAZIONI.
GARANTITA DAL GESTORE DELLA CONCORRENZA

D→DURABILITY = PERSISTENZA: L'EFFETTO DI UNA TRANSAZIONE CHE HA ESEGUITO COMMIT NON DEVE ANDARE PERSO
GARANTITA DAL GESTORE DELL'AFFIDABILITA', DAL GESTORE DEL BUFFER E DA GESTORE DELLA MEMORIA SECONDARIA

ILLUSTRARE L'ARCHITETTURA DI UN DMBS DESCRIVENDO IN PARTICOLARE IL MODULO DI GESTIONE DEI BUFFER; SI INDICHI INOLTRE QUALI MODULI GARANTISCONO LE PROPRIETÀ DI PERSISTENZA E CONSISTENZA DELLE TRANSAZIONI.

Architettura di un DBMS

Un Database Management System (DBMS) è composto da vari moduli che collaborano per gestire i dati in modo efficiente e sicuro. Questi moduli includono:

- **Gestore delle interrogazioni:** E' responsabile della gestione e ottimizzazione delle query SQL inviate dagli utenti.
- **Gestore dei Metodi di Accesso:** Si occupa di come i dati vengono fisicamente letti e scritti sul disco
- **Gestore di Concorrenza**: Controlla l'accesso simultaneo ai dati per evitare conflitti e garantire la consistenza.
- **Gestore dell'Affidabilità**: Si occupa del ripristino dei dati in caso di guasti.
- **Gestore dei Buffer**: Gestisce la memoria temporanea per migliorare le prestazioni delle operazioni di I/O.
- **Gestore della memoria secondaria**: Gestisce lo spazio di archiviazione dei dati sul disco.

Modulo di Gestione dei Buffer

Il modulo di gestione dei buffer ha il compito di ottimizzare l'accesso ai dati utilizzando una zona dedicata della memoria principale chiamata buffer per ridurre il numero di accessi alla memoria secondaria.

Funzioni Principali:

1) Mantiene in memoria i blocchi di dati più frequentemente accessibili per accelerare le operazioni di lettura e scrittura.
2) Utilizza principio di località ed euristiche per decidere quali blocchi di dati rimuovere dalla memoria quando è piena.
3) Con un indice i segna i blocchi di dati che sono in uso attivo per evitare che vengano sostituiti.
4) Con un indice j tiene traccia dei blocchi modificati (dirty) che devono essere scritti sul disco per garantire la persistenza dei dati.

SI PRESENTI IN DETTAGLIO LA DEFINIZIONE DI CONFLICT-EQUIVALENZA TRA DUE SCHEDULE.

Due schedule S1 ed S2 sono conflict-equivalenti se possiedono le stesse operazioni, gli stessi conflitti e le operazioni in conflitto sono nello stesso ordine nei due schedule.

Il conflitto si verifica quando una coppia di operazioni nello stesso SCHEDULE (ai,aj) hanno le seguenti caratteristiche:

- Appartengono a transazioni diverse

- operano sulla stessa risorsa

-almeno una delle due è un' operazione di scrittura

- ai compare in S prima di aj

LO STUDENTE ILLUSTRI LA STRUTTURA DI ACCESSO AI DATI DENOMINATA STRUTTURA AD ACCESSO CALCOLATO (HASHING), SI DESCRIVANO IN PARTICOLARE I SEGUENTI PUNTI: (i) LE CARATTERISTICHE DELLA STRUTTURA DI ACCESSO, (ii) L'ALGORITMO DI RICERCA DI UNA TUPLA CON CHIAVE K USANDO L'INDICE.

(i)

Gli indici hash sono una struttura fisica che permette l'indicizzazione veloce ed efficiente delle tuple in una base di dati.

Si basano su una funzione di hash che mappa i valori delle chiavi nei bucket i quali contengono i puntatori agli indirizzi di memorizzazione dove sono contenute le tuple.

Le funzioni di hash devono distribuire in modo uniforme e casuale le chiavi nei bucket per limitare le collisioni(*).

(*)Una collisione si verifica se per due diversi valori di chiave K1 e K2 si ha h(f(k1)) = h(f(k2)).

Questo perché se c'è un numero elevato di collisioni si saturerebbero i bucket e saturando i bucket si renderebbero necessari i bucket di overflow vanificando i vantaggi in termini di accesso garantiti dalla struttura hash.

(ii)

Algoritmo di ricerca di una tupla con chiave k utilizzando l'indice.

1) Calcolo l'hash della chiave di ricerca K utilizzando la funzione di hash :
   b= h(f(k)) (costo zero)
2) Accedo al bucket b (costo: 1 accesso a pagina)
3) Accedo alle tuple attraverso i puntatori contenuti nel bucket finché non trovo la tupla con la chiave K
   (costo: se n sono il numero dei puntatori contenuti nel bucket b, m<= n è il numero di accessi a pagina)