Domande Orale Basi di Dati

Lorenzo Bandini

June 2024

Contents

1 SQL e Schemi

1.1 Definzione chiave e superchiave

- Un insieme K di attributi è una **superchiave** per una relazione r se ogni coppia di tuple distinte $t1$ e $t2$ in r ha valori diversi per almeno uno degli attributi in K.
- K è una **chiave** per r se è una superchiave minimale, cioè non contiene nessun altro insieme di attributi che sia superchiave di r.

Ogni relazione ha almeno una superchiave, che è l'insieme di tutti gli attributi della relazione stessa. Pertanto, ogni relazione ha almeno una chiave.
Le chiavi garantiscono l'accessibilità a ciascun dato della base di dati, fornendo un metodo per identificare univocamente le tuple. Le chiavi consentono di correlare i dati tra relazioni diverse, stabilendo relazioni e integrità referenziale nel database.

1.2 Che cos'è una gerarchia?

Una gerarchia è una struttura organizzativa in cui le classi di entità sono disposte in livelli di specializzazione e generalizzazione.
Le classi inferiori, chiamate sottoclassi, ereditano proprietà dalle superclassi, garantendo l'ereditarietà delle caratteristiche.
Gli elementi di una sottoclasse sono inclusi nell'insieme degli elementi della superclasse.
Nel contesto dei modelli a oggetti, la gerarchia tra tipi oggetto stabilisce una relazione di sottotipo asimmetrica, riflessiva e transitiva, consentendo agli elementi di un sottotipo di essere usati dove sono richiesti elementi del supertipo.
Le gerarchie possono essere semplici, con una singola superclasse per sottoclasse, o multiple, con una sottoclasse che ha più di una superclasse.

1.3 Come eliminare le gerarchie da schema concettuale a schema relazionale?

Il modello relazionale non può rappresentare direttamente le gerarchie quindi vanno eliminate e sostituirle con classi e relazioni:

- **Relazione Unica**: Le sottoclassi figlie vengono accorpate alla classe genitore. Al relazione unica viene aggiunto un attributo discriminatore per identificare di che classe si tratta e i valori che non fanno parte della classe diventano *NULL*. Non è consigliabile quando le sottoclassi figlie hanno tanti elementi diversi per le troppe colonne inutili a *NULL*.
- **Partizionamento Orizzontale**: Tutte le sottoclassi figlie accorpano gli attributi della classe genitore. Adatto per quando la classe padre ha pochi attributi. Non è possibile farlo quando la classe padre ha un vincolo referenziale.

- **Partizionamento Verticale**: Viene sostituita la gerarchia con delle relazioni uno a uno che legano le classi figlie alla classe padre. Viene però aggiunto il vincolo che ogni occorrenza della classe padre esiste al massimo una sottoclasse figlia.

1.4 Che cos'è un vincolo interrelazionale?

Sono i vincoli di integrità referenziale, mettono in corrispondenza relazioni di diverse tabelle in modo da soddisfare l'esigenza di non essere ridondante e di avere i dati sincronizzati. I valori di un attributo referenziante devono esistere nell'attributo della tabella referenziata (chiavi esterne o foreign key).

1.5 Cos'è uno schema concettuale?

Lo schema concettuale in progettazione SQL è una rappresentazione ad alto livello della struttura dei dati di un database. Include:

- **Entità e relazioni**: Definisce le entità principali del dominio di interesse e le relazioni tra di esse.
- **Attributi delle entità**: Specifica i dettagli di ciascuna entità, indicando quali attributi sono importanti e come sono collegati.
- **Relazioni tra tra entità e molteplicità**: Descrive le connessioni tra le entità e specifica la molteplicità di tali relazioni. La molteplicità indica il numero di istanze di un'entità che possono essere associate a un'istanza di un'altra entità.

Lo schema concettuale è indipendente dal modello fisico di archiviazione dei dati. Serve da base per sviluppare lo schema logico e quello fisico del database.

1.6 Quali sono gli operatori che si possono usare con le select annidate?

Tutti.

1.7 Struttura completa di una query

```
SELECT listaAttributi
FROM listaTabelle
WHERE condizione
GROUP BY listaAttributi
HAVING condizione
ORDER BY listaAttributi
```

1.8 C'è una differenza tra IN ed EXISTS? (A livello di funzionamento)

- **IN** è binario. Con IN possiamo verificare l'appartenenza a un insieme definito per enumerazione.
- **EXISTS** è unario. Restituisce `TRUE` se la sottoquery contiene una o più righe.

1.9 Differenze tra ANY, ALL, EXISTS

Gli operatori ANY, ALL e EXISTS sono utilizzati nelle query SQL per confrontare i risultati delle sottoquery con un predicato esterno.

- **ANY**: Il predicato è vero se almeno uno dei valori restituiti dalla sottoquery soddisfa la condizione.
- **ALL**: Il predicato è vero se tutti i valori restituiti dalla sottoquery soddisfano la condizione.
- **EXISTS**: Il predicato è vero se la sottoquery restituisce almeno una tupla.

Questi operatori consentono di esprimere condizioni complesse e di filtrare i risultati delle query in base a criteri specifici.

1.10 Differenza tra WHERE e HAVING

- **WHERE** è applicabile solo alle righe e diamo le specifiche condizioni per cui ci restituirà il valore `TRUE`.
- **HAVING** è analogo ma si applica ai gruppi.

1.11 Posso mettere COUNT nel WHERE?

No, `WHERE` viene utilizzata per filtrare le righe prima delle aggregazioni dei dati, `COUNT` si dovrebbe utilizzare dopo. La funzione `COUNT` si usa con `HAVING`.

1.12 Che cos'è un cursore?

È il meccanismo per ottenere uno alla volta gli elementi di una relazione. Un cursore viene definito con un'espressione SQL, poi:

- Si apre per far calcolare al DBMS il risultato e poi
- Con un opportuno comando si trasferiscono i campi delle ennuple in opportune variabili del programma.

2 Normalizzazione

2.1 Definizione formale di dipendenza funzionale?

Dato uno schema $R(T)$ e $X, Y \subseteq T$, una dipendenza funzionale (DF) fra gli attributi X e Y è un vincolo su R sulle istanze della relazione, espresso nella forma

$$X \to Y,$$

i.e. X determina funzionalmente Y o Y è determinato da X, se per ogni istanza valida r di R un valore di X determina in modo univoco un valore di Y:

$$\forall r \text{ istanza valida di } R,$$

$$\forall t_1, t_2 \in r, \text{ se } t_1[X] = t_2[X] \text{ allora } t_1[Y] = t_2[Y]$$

2.2 Definizione chiave e superchiave utilizzando le dipendenze funzionali

Si parla di **dipendenza funzionale (DF) completa** quando $X \to Y$ e per ogni $W \subset X$, non vale $W \to Y$.

- Se X è una **superchiave**, allora X determina ogni altro attributo della relazione: $X \to T$
- Se X è una **chiave**, allora $X \to T$ è una **DF completa**

2.3 Cos'è una chiave candidata?

X è chiave candidata di $R\langle T, F\rangle$ se e solo se:

- X è una superchiave, ossia $X^+ = T$.
- Nessun sottoinsieme proprio di X è una superchiave, cioè se $Y \subset X$, allora $Y^+ \neq T$.

2.4 Cos'è un attributo estraneo?

Dato $X \to Y$, $A \in X$ è estraneo se e solo se $X - \{A\} \to Y \in F^+$.

2.5 Cos'è un attributo primo?

Un attributo che appartiene ad almeno una chiave candidata.

2.6 Come si trova una chiave? Calcolare la chiusura è l'unica alternativa?

- Considero gli attributi X che appaiono solo a sinistra.
- Aggiungo a X attributi che appaiono sia a sinistra che a destra finché $X^+ \neq T$.
- La chiusura si usa per verificare se X è chiave, l'algoritmo per trovare le chiavi è un altro.

2.7 Definizione di chiave e superchiave con dipendenze funzionali

Sia dato uno schema $R < T, F >$:

- X è Superchiave se $X \to T$.
- X è Superchiave minimale o Chiave se $X \to T$ e per ogni $A_i \subset X$, $A_i \to T \notin F$.

2.8 Cos'è una copertura canonica?

F è detta una copertura canonica se e solo se:

- La parte destra di ogni dipendenza funzionale in F è un attributo.
- Non esistono attributi estranei.
- Nessuna dipendenza in F è ridondante.

Attributo estraneo: $X - \{A\} \to Y$ appartiene alla chiusura di F.
Dipendenza ridondante: se $(F - \{X \to Y\})^+ = F^+$.

2.9 Quale è l'algoritmo per calcolare una copertura canonica?

Per ogni insieme di dipendenze F esiste una copertura canonica e per trovarla possiamo:

- Trasformare le dipendenze in dipendenze atomiche della forma $X \to Y$.
- Eliminare da F gli attributi estranei
- Eliminare da F le dipendenze ridondanti

2.10 Cos'è una dipendenza ridondante?

$X \to Y$ è ridondante se e solo se $X \to Y \in (F - \{X \to Y\})^+$.

2.11 Cos'è una dipendenza atomica?

Ogni dipendenza funzionale $X \to A_1 A_2 \dots A_n$ si può scomporre nelle dipendenze funzionali $X \to A_1$, $X \to A_2$, $\dots$, $X \to A_n$. Le dipendenze funzionali del tipo $X \to A$ si chiamano **dipendenze funzionali atomiche**.

2.12 Assiomi di Armstrong

- Se $Y \subseteq X$, allora $X \to Y$ (Riflessività **R**)
- Se $X \to Y$, $Z \subseteq T$, allora $XZ \to YZ$ (Arricchimento **A**)
- Se $X \to Y$, $Y \to Z$, allora $X \to Z$ (Transitività **T**)

2.13 Parlami della decomposizione senza perdita.

La chiave è l'unico modo per avere una decomposizione senza perdita di informazione.

- Uno schema $R(X)$ si **decompone senza perdita di dati** negli schemi $R_1(X_1)$ ed $R_2(X_2)$ se, per ogni possibile istanza r di $R(X)$, il join naturale delle proiezioni di r su X_1 ed X_2 produce la tabella di partenza (cioè non contiene tuple spurie).
  Se l'insieme degli attributi comuni alle due relazioni $(X_1 \cap X_2)$ è chiave per almeno una delle due relazioni decomposte, allora la decomposizione è senza perdita.
  Inoltre la decomposizione preserva i dati se e solo se $T_1 \cap T_2 \to T_1 \in F^+$ oppure $T_1 \cap T_2 \to T_2 \in F^+$
- Una decomposizione **conserva le dipendenze** se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti, più nello specifico, la decomposizione preserva le dipendenze funzionali se e solo se l'unione delle dipendenze proiettate sui sottoinsiemi T_i, indicata come $\pi_{T_i}(F)$, è una copertura di F.

2.14 Definzione BCNF usando le dipendenze funzionali

Uno schema $R\langle T, F\rangle$ è in BCNF se e solo se $\forall(X \to Y \in F \mid X$ è superchiave di $R)$. Una relazione r è in forma normale di Boyce-Codd (BCNF) se, per ogni dipendenza funzionale (non banale) $X \to Y$ definita su di essa, X contiene una chiave K di r (è una superchiave).

2.15 Definizione 3FN

Una relazione r è in terza forma normale (3NF) se, per ogni dipendenza funzionale (non banale) $X \to Y$ definita su r, è verificata almeno una delle seguenti condizioni:

- X contiene una chiave K di r (è una superchiave) (come nella BCNF).
- **Oppure** ogni attributo in Y è contenuto in almeno una chiave K di r.

2.16 Quale FN preserva i dati e quale le dipendenze?

3FN e BCNF preservano entrambe i dati ma BCNF potrebbe non preservare le dipendenze mentre 3FN le preserva.

2.17 Perché FNBC non preserva le dipendenze?

Questo accade perché il processo di decomposizione di una relazione per portarla in FNBC può richiedere di spezzare le dipendenze funzionali in modi che non possono essere ricostruiti direttamente dalle relazioni decomposte.

2.18 Parlami dell'algoritmo di analisi

Algoritmo con complessità esponenziale per ottenere una decomposizione ρ di $R\langle T, F\rangle$ con F copertura canonica, tale che ρ preservi dati e sia in BCNF:

1. $\rho = R$.
2. while (esiste una $R_i\langle T_i, F_i\rangle \in \rho$ che non è in BCNF per una dipendenza funzionale $X \to A$)
   - $T_a = X^+$;
   - $F_a = \pi_{T_a}(F_i)$;
   - $T_b = T_i - X^+ + X$;
   - $F_b = \pi_{T_b}(F_i)$;
   - $\rho = \rho - R_i + \{R_a\langle T_a, F_a\rangle, R_b\langle T_b, F_b\rangle\}$;

2.19 Cos'è l'algoritmo di sintesi? Quali sono i passi per dedurlo?

Algoritmo con complessità polinomiale per ottenere una decomposizione ρ di $R\langle T, F\rangle$ tale che ρ preservi dati, dipendenze e sia in 3NF. I passi sono:

1. $\rho = \{\}$.
2. Determina G, copertura canonica di F.
3. Sostituisci $X \to A1, \ldots, X \to An$ con $X \to A1, \ldots, An$.
4. Per ogni $X \to Y$, aggiungi $R\langle XY, X \to Y\rangle$ a ρ.
5. Elimina ogni $R1 \subseteq R2$ in ρ.
6. Se $\exists(Ri\langle W, F\rangle \in \rho \mid W$ superchiave di $R)$ allora aggiungi $Rk\langle C, \ldots\rangle$, con C chiave di R, a ρ.

2.20 Qual è l'algoritmo che non preserva i dati e quale non preserva le dipendenze?

Sia sintesi che analisi preservano i dati. Sintesi preserva anche le dipendenze, invece per analisi non è garantito.

2.21 Quali sono i vantaggi e svantaggi della 3FN rispetto alla BCNF

Svantaggi

- La 3FN è meno restrittiva della BCNF.
- Tolleranza di alcune ridondanze e anomalie sui dati.
- Certifica meno la qualità dello schema ottenuto.

Vantaggi

- La 3FN è sempre ottenibile, qualsiasi sia lo schema di partenza.

2.22 Quando una relazione è in terza forma normale?

Uno schema $R\langle T, F\rangle$ è in 3NF se e solo se per ogni dipendenza funzionale $X \to Y \in F$, si verifica che X è una superchiave di R o Y è un attributo primo.

- La 3FN ammette una dipendenza non banale e non-da-chiave se gli attributi a destra sono primi.
- La BCNF non ammette mai nessuna dipendenza non banale e non-da-chiave.

3 DBMS

3.1 Gestore del Buffer, area delle pagine

Gestore Memoria Permanente

- Fornisce un'astrazione della memoria permanente in termini di insiemi di file logici di pagine fisiche di registrazioni (blocchi), nascondendo le caratteristiche dei dischi e del sistema operativo.

Gestore del Buffer

- Si preoccupa del trasferimento delle pagine tra la memoria temporanea e la memoria permanente, offrendo agli altri livelli una visione della memoria permanente come un insieme di pagine utilizzabili in memoria temporanea, astraendo da quando esse vengano trasferite dalla memoria permanente al buffer e viceversa.

3.2 Come funziona la gestione dell'area del buffer?

Il **Buffer Pool** nel DBMS gestisce le pagine recentemente accedute dal disco, dividendo il buffer in area libera e occupata. Ogni pagina nel buffer ha un conteggio di pin e uno stato (dirty/non dirty). Le operazioni principali includono 'getAndPinPage', 'unPinPage', 'setDirty', e 'flushPage'. Politiche come LRU (Least Recently Used) e MRU (Most Recently Used, migliore per i DB) gestiscono il rimpiazzamento delle pagine per ottimizzare le prestazioni.

3.3 Differenza tra indice primario e secondario?

- **Indice Primario**: La chiave di ordinamento del file sequenziale coincide con la chiave di ricerca dell'indice.
- **Indice Secondario**: La chiave di ordinamento e la chiave di ricerca sono diverse.

3.4 SortMerge cosa prende come parametri?

Prende le due tabelle OE e OI ordinate sugli attributi di giunzione.

3.5 NestedLoop efficienza a confronto con SortMerge (complessità)

- Algoritmo SortMerge: costo $N \log(N)$.
- NestedLoop: $O(n^2)$.

Il costo di esecuzione dipende (anche) dallo spazio a disposizione in memoria centrale.

3.6 Come funziona l'algoritmo di SortMerge?

L'algoritmo di SortMerge è comunemente utilizzato nei DBMS per ordinare grandi insiemi di dati quando la memoria centrale è limitata. Ecco come funziona:

- **Sort Interno**: Si leggono le pagine del file una alla volta e si ordinano i record di ciascuna pagina usando un algoritmo di sort interno (ad esempio QuickSort). Ogni pagina così ordinata è chiamata "run" e viene scritta su disco in un file temporaneo.
- **Merge**: Le "run" vengono fuse insieme in uno o più passaggi di fusione fino a produrre una singola "run" ordinata. Nel caso base con Z = 2 (due vie), e NB buffer disponibili, si fondono due "run" alla volta. Un buffer è utilizzato per ciascuna "run" e un terzo buffer è usato per l'output, che viene scritto una pagina alla volta.
- **Complessità**: Nel caso base con Z = 2 e NB buffer, il numero di operazioni di I/O è approssimativamente 2 * NP * ($\lceil \log_2 NP \rceil + 1$), dove NP è il numero di pagine da ordinare. Questo algoritmo ha un costo di N * Log(N) nel numero di operazioni di I/O.

Questo metodo è efficace per ordinare grandi quantità di dati.

3.7 Quando si preferisce il SortMerge rispetto al PageNestedLoop in termini di complessità?

Quando ci serve che i record risultanti siano ordinati. Se l'ordinamento non è di interesse, allora è meglio PageNestedLoop, in quanto richiede meno trasferimenti I/O.

3.8 Come funziona l'IndexNestedLoop? Come si differenzia dagli altri?

È uguale a NestedLoop ma trova i record della relazione interna usando l'indice, non il ciclo annidato.

3.9 PageNestedLoop confronto con NestedLoop

Il confronto tra **PageNestedLoop** e **NestedLoop** riguarda principalmente l'efficienza nell'accesso e nel trattamento delle pagine di dati nei database:

- **NestedLoop**: Utilizza un metodo di join basato su doppio ciclo, dove ogni tupla di una tabella è confrontata con ogni tupla dell'altra tabella. È semplice da implementare ma può essere inefficiente con grandi quantità di dati a causa del numero elevato di confronti.

- **PageNestedLoop**: Ottimizza il metodo di join riducendo il numero di accessi alle pagine di dati. Invece di confrontare singole tuple, considera intere pagine di dati (blocchi). Questo approccio può ridurre significativamente il numero di I/O necessari per eseguire il join, migliorando le prestazioni complessive in scenari con grandi volumi di dati.

In conclusione, mentre 'NestedLoop' è più semplice e universale, 'PageNestedLoop' è preferibile quando si cerca di ottimizzare le prestazioni del join in contesti di grandi dimensioni e complessità dei dati.

3.10 Cos'è un piano d'accesso fisico?

Un piano d'accesso è un algoritmo per eseguire un'interrogazione usando gli operatori fisici disponibili.

3.11 Cosa sono gli operatori fisici? Cosa fanno in più degli operatori logici?

Gli operatori fisici sono implementati come iteratori che utilizzano direttamente le risorse della macchina fisica. Rispetto agli operatori logici, offrono: Un operatore fisico è un iteratore, cioè un oggetto con metodi implementati usando gli operatori della macchina fisica. Gli operatori fisici offrono le interfacce *open*, *next*, *close*, *isDone*.

Queste caratteristiche consentono agli operatori fisici di eseguire operazioni più efficienti e dirette rispetto agli operatori logici, che operano a un livello più astratto e indipendente dalle specificità hardware.

3.12 Qual è l'input di un piano d'accesso? Come si chiama il gestore?

Un piano di accesso prende in input N tabelle. Nella macchina logica c'è il gestore delle interrogazioni che comprende l'ottimizzatore e il gestore dei piani di accesso.

3.13 Definizione della Filter

Implementa la restrizione di una tabella, senza indici.

3.14 Perché abbiamo studiato i piani di accesso fisici?

Studiamo i piani di accesso fisici nei database per ottimizzare le prestazioni delle query, minimizzare le operazioni di I/O e scegliere gli indici più efficaci. Questo ci permette di migliorare l'efficienza complessiva del sistema, adattandolo alle specifiche risorse hardware disponibili.

4 Gestione dell'affidabilità

4.1 Definizione formale di transazione?

Una transazione è una sequenza di operazioni SQL che rappresenta l'unità di lavoro fondamentale per modificare il contenuto di una base di dati che corrisponde a una serie di operazioni fisiche elementari di lettura e scrittura sul DB. È delimitata dai comandi `begin transaction` e `end transaction`, e può essere confermata (`commit work`) o annullata (`rollback work`) atomicamente per garantire la consistenza dei dati anche in caso di malfunzionamenti del sistema.

4.2 Quando facciamo una transazione, vogliamo scrivere sul disco?

Durante l'esecuzione di una transazione in un database, è fondamentale garantire che le modifiche apportate ai dati siano permanentemente salvate su disco mediante il processo di *commit*. Questo assicura che le modifiche siano resistenti anche in caso di malfunzionamenti del sistema, garantendo la loro persistenza nel tempo.

4.3 Come si garantisce l'affidabilità?

La garantiamo grazie alle proprietà **ACID**, cioè:

- **Atomicità**: Una transazione segue il principio del "tutto o niente", garantendo che tutte le operazioni siano eseguite con successo prima che le modifiche siano applicate al database o annullate in caso di fallimento.
- **Consistenza**: Una transazione deve mantenere il database in uno stato consistente, rispettando i vincoli di integrità definiti. Le modifiche che violano tali vincoli sono annullate.
- **Isolamento**: Ogni transazione è eseguita indipendentemente dalle altre, assicurando che le operazioni siano isolate e producano risultati coerenti, indipendentemente dall'interferenza di altre transazioni.
- **Persistenza**: Le modifiche confermate da una transazione tramite il "commit" devono essere permanenti, anche dopo un riavvio o guasto del sistema, garantendo che i cambiamenti siano salvati e recuperabili a lungo termine.

4.4 Regole di scrittura del log

- Regola **Write Ahead Log (WAL)**: La parte BS (before state) di ogni record di log deve essere scritta prima che la corrispondente operazione venga effettuata nella base di dati.

- Regola di **Commit Precedence**: La parte AS (after state) di ogni record di log deve essere scritta nel log prima di effettuare il commit della transazione.

4.5 Protocollo Rifare e Disfare

Il protocollo Rifare e Disfare utilizza le seguenti primitive:

- **UNDO**: Utilizzato per disfare un'azione su un oggetto O. Questo viene fatto copiando il valore BS in O, annullando così l'effetto dell'azione precedente. Ad esempio, un'operazione di inserimento può essere disfatta cancellando l'oggetto O.
- **REDO**: Utilizzato per rifare un'azione su un oggetto O. Questo viene fatto copiando il valore AS in O, ripristinando l'effetto dell'azione precedente. Ad esempio, un'operazione di cancellazione può essere rifatta ripristinando l'oggetto O.

Queste primitive sono utilizzate per gestire le azioni di undo e redo basate sui record di log associati a una transazione nel sistema di gestione del database.

4.6 Altri protocolli oltre al rifare e disfare?

- **Disfare–NonRifare**: Utilizza il log per annullare le modifiche delle transazioni, ma non ripristina le modifiche applicate. Le modifiche sono applicate solo dopo il commit della transazione.
- **NonDisfare–Rifare**: Non utilizza il log per annullare le modifiche delle transazioni, ma lo utilizza per ripristinare le modifiche applicate. Le modifiche sono applicate immediatamente.
- **NonDisfare–NonRifare**: Non utilizza il log né per annullare né per ripristinare le modifiche delle transazioni. Le modifiche sono applicate immediatamente.

4.7 Parlami del protocollo rifare e disfare (motivo per il quale funziona)

Il protocollo Rifare e Disfare è efficace perché utilizza un log delle transazioni per garantire la durabilità e la consistenza delle modifiche nel database. Funziona secondo il seguente principio:

Quando una transazione apporta modifiche al database:

- Le modifiche vengono registrate nel log prima di essere applicate al database stesso.
- Il log permette di annullare (*UNDO*) le modifiche di una transazione in caso di fallimento o annullamento della transazione stessa.

- Il log permette anche di ripetere (*REDO*) le modifiche di una transazione in caso di recupero dopo un fallimento, garantendo che tutte le modifiche siano persistenti.

4.8 Come agisci dopo aver costruito Undo e Redo?

Dopo aver ottenuto i due insiemi di UNDO e REDO:

- Ripercorriamo il log all'indietro, fino alla più vecchia azione delle transazioni in UNDO, disfacendo tutte le azioni delle transazioni in UNDO
- ripercorrere il log in avanti, rifacendo tutte le azioni delle transazioni in REDO

5 Gestione della concorrenza

5.1 Cos'è un Lock?

Blocco lettura/scrittura per ogni dato. Ogni transazione, prima di eseguire un'operazione su un dato, richiede il lock (di lettura o di scrittura, in base all'operazione), viene messa in attesa (se necessario) e al termine dell'operazione lo rilascia (nel caso di Strict-2PL i lock vengono rilasciati solo a transazione terminata). Sono permesse letture concorrenti e scritture in mutua esclusione.

5.2 Cosa si intende per risorsa? Livelli di granularità?

Una risorsa è una tabella o un valore di una cella.
La granularità si riferisce al livello di dettaglio o dimensione delle risorse a cui si applicano i lock. Nei DBMS, i lock possono essere applicati a vari livelli di granularità, come singole righe, intere tabelle o pagine di memoria. Questo approccio, chiamato multi-granularità, permette di gestire l'accesso concorrente in modo flessibile, bilanciando tra il controllo fine (lock su righe) e l'efficienza (lock su tabelle o pagine).

5.3 Che cosa si intende per serializzabilità?

Una serie di transizioni T_i si dice serializzabile se e solo se l'effetto da esse prodotto è uguale all'effetto prodotto dall'esecuzione, in qualche ordine, delle sole T_k che sono terminate normalmente. Nel caso di esecuzioni concorrenti di più transazioni, l'effetto complessivo è quello di un'esecuzione seriale.

5.4 Elenca qualche problema che possono sorgere sulla concorrenza

- **Deadlock**: $T1$ acquisisce X; $T2$ acquisisce Y; $T1$ richiede Y; $T2$ richiede X. $T1$ e $T2$ restano in attesa l'una dell'altra.

5.5 Che cos'è un protocollo pessimistico?

Protocollo di gestione della concorrenza incentrato sulla prevenzione che agisce ritardando l'esecuzione di transazioni che potrebbero generare conflitti con quella corrente.

5.6 Parlami dei protocolli ottimistici. Quando lo preferiamo rispetto a quello pessimistico?

- Protocolli **ottimistici**: Permettono l'esecuzione sovrapposta e non sincronizzata di transazioni ed effettuano un controllo sui possibili conflitti generati solo a valle del commit.

- Protocolli **pessimistici/conservativi**: Tendono a ritardare l'esecuzione di transazioni che potrebbero generare conflitti, e quindi anomalie, rispetto alla transazione corrente. Cercano quindi di prevenire i conflitti.

5.7 Come funziona il protocollo a due fasi stretto(2PL)?

- Ogni transazione, prima di effettuare un'operazione, acquisisce il blocco corrispondente (chiede il lock).
- Transazioni diverse non ottengono blocchi in conflitto.
- I blocchi/lock si rilasciano alla terminazione della transazione (cioè al commit).

Il problema è che i protocolli 2PL possono generare schedule con situazioni di deadlock

5.8 Protocolli pessimistici, cos'è il grafo delle richieste? Parlami del gestore delle concorrenze

Grafo diretto i cui nodi sono le transazioni attive. Un arco $T1 \rightarrow T2$ indica che $T1$ è in attesa di una release di $T2$. Se si crea un ciclo, viene fatta abortire la transazione coinvolta più recente, o quella che ha meno risorse, o quella il cui abort ha costo inferiore.

5.9 Quando decidiamo di mettere in lock qualcosa?

I DBMS decidono di mettere in lock una risorsa quando una transazione necessita di accedervi in modo tale da prevenire anomalie nelle operazioni concorrenti. I lock vengono utilizzati per garantire la coerenza e l'isolamento delle transazioni, seguendo il protocollo ACID (Atomicità, Consistenza, Isolamento, Durabilità). In particolare, si mette in lock qualcosa quando:

- Una transazione vuole leggere o scrivere su una risorsa.
- Si desidera prevenire conflitti tra transazioni concorrenti.
- Si devono mantenere i vincoli di integrità durante le operazioni di modifica.

5.10 Come funziona il meccanismo delle lock?

Il meccanismo delle lock nei DBMS funziona attraverso l'uso di vari livelli di granularità e modalità di lock. Ecco come funziona in sintesi:

- **Richiesta di Lock**: Una transazione richiede un lock su una risorsa (es. riga, tabella, pagina).
- **Assegnazione del Lock**: Se la risorsa è disponibile, il lock viene assegnato e la transazione può procedere. Se la risorsa è già bloccata, la transazione viene messa in coda.

- **Livelli di Granularità**: I lock possono essere applicati a vari livelli di granularità, come singole righe, intere tabelle o pagine di memoria, a seconda delle necessità di controllo e performance.
- **Modalità di Lock**: I lock possono essere di vari tipi, come:
  - **Lock di Lettura (Condiviso)**: Permette a più transazioni di leggere una risorsa contemporaneamente.
  - **Lock di Scrittura (Esclusivo)**: Permette solo a una transazione di modificare una risorsa, impedendo l'accesso ad altre transazioni.
- **Gestione dei Lock**: Una volta completate le operazioni, i lock vengono rilasciati, permettendo ad altre transazioni in coda di accedere alla risorsa.

5.11 Quando viene rilasciata la risorsa dalla lock?

Alla fine della transazione.

5.12 Cos'è la classe di priorità?

La classe di priorità è un meccanismo per prevenire deadlock assegnando livelli di priorità alle transazioni, spesso basati su timestamp, dove le transazioni con priorità più bassa rilasciano risorse a favore di quelle con priorità più alta. Tuttavia, questo può causare starvation, dove transazioni con bassa priorità vengono continuamente bloccate.

5.13 Come risolvo il deadlock?

- **Uso dei timeout**: ogni operazione di una transazione ha un timeout entro il quale deve completarsi, altrimenti la transazione viene annullata (abort).
- **Deadlock avoidance**: prevenire configurazioni che possono causare deadlock:
  - Lock/Unlock simultaneo di tutte le risorse.
  - Uso di time-stamp o classi di priorità tra transazioni (rischio di starvation).
- **Deadlock detection**: utilizzo di algoritmi per identificare situazioni di deadlock e meccanismi di recovery:
  - Grafo delle richieste/risorse per identificare cicli (deadlock).
  - Abort delle transazioni coinvolte nel ciclo per risolvere il deadlock.

5.14 Timeout e Timestamp

Un metodo alternativo al 2PL per la gestione della concorrenza in un DBMS prevede l'utilizzo dei time-stamp delle transazioni (metodo TS):

- Ad ogni transazione si associa un timestamp che rappresenta il momento di inizio della transazione.
- Ogni transazione non può leggere o scrivere un dato scritto da una transazione con timestamp maggiore.
- Ogni transazione non può scrivere su un dato già letto da una transazione con timestamp maggiore.