Esami - Basi di dati

VR443470

giugno 2023

Indice

1 Domande di teoria - Primo parziale

Le domande più frequenti che si possono incontrare nel primo parziale di Basi di dati sono:

1. Si illustri il concetto/costrutto di **entità** nel modello Entità-Relazione.
2. Si illustri il concetto/costrutto di **relazione** nel modello Entità-Relazione.
3. Si illustri il concetto/costrutto di **generalizzazione** nel modello Entità-Relazione.
4. Si illustri il concetto/costrutto di **identificatore** nel modello Entità-Relazione.
5. Si illustri il concetto/costrutto di **superchiave** nel modello Entità-Relazione.
6. Si illustri il concetto/costrutto di **attributo multivalore** nel modello Entità-Relazione.

La risposta, per essere considerata perfetta, deve includere le seguenti caratteristiche:

- Semantica
- Sintassi grafica con esempio
- Istanza
- Eventuali proprietà

Qui di seguito vengono date le possibili risposte alle domande di teoria:

1. *Si illustri il concetto/costrutto di **entità** nel modello Entità-Relazione.*

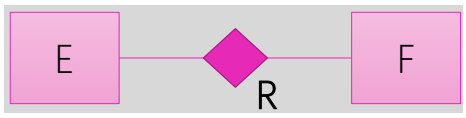
   Semantica. L'entità rappresenta una classe di oggetti che hanno proprietà comuni ed esistenza "autonoma" ai fini dell'applicazione di interesse. Il nome dato ad ogni entità è identificativo di quella determinata classe di oggetti e deve essere univoco all'interno dello schema.
   Sintassi grafica. Per esempio, l'entità studenti rappresenta la classe di oggetti degli studenti di un'università e gli attributi possibili possono essere: matricola, nome, cognome, data di nascita, ecc. La sintassi grafica è la seguente:

   Istanza. L'istanza di un'entità è un oggetto della classe che lo rappresenta e non un unico valore. Per esempio, nell'entità studenti, lo studente Mario Rossi (in carne ed ossa) rappresenta un'istanza dell'entità.

2. *Si illustri il concetto/costrutto di **relazione** nel modello Entità-Relazione.*

Semantica. La relazione rappresenta legami logici tra una o più entità. Ogni relazione deve avere un nome univoco all'interno dello schema e non può avere identificatori. Esistono due tipi di relazioni: *ricorsive*, cioè in cui è coinvolta una sola entità, *n-arie*, in cui sono coinvolte n entità. Esse nascono solo quando le entità coinvolte contengono almeno una tupla.
Sintassi grafica. Un esempio di relazione è la "Residenza" tra le entità "Città" e "Impiegato". La sua sintassi grafica è un rombo:



Istanza. Data una relazione R tra n entità $E_1, E_2, ..., E_n$, un'istanza è composta da una ennupla del tipo:

$$(e_1, e_2, ..., e_i) \text{ dove } e_i \in I(E_i), 1 \leq i \leq n$$

Inoltre, esiste un'importante proprietà che afferma:

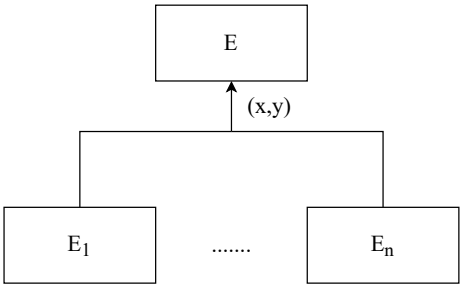
$$I(R) \subseteq I(E_1) \times I(E_2) \times ... \times I(E_n)$$

3. *Si illustri il concetto/costrutto di **generalizzazione** nel modello Entità-Relazione.*

Semantica. Le generalizzazioni rappresentano legami logici tra un'entità E, chiamata genitore, e più entità $E_1, ..., E_n$, chiamate figlie. Quindi, si dice che l'entità E (genitore) è la generalizzazione delle entità $E_1, ..., E_n$ (figlie) e quest'ultime vengono chiamate specializzazioni. Inoltre, ogni occorrenza dell'entità figlia è anche un'occorrenza dell'entità padre, e ogni proprietà dell'entità padre è anche una proprietà dell'entità figlia.
La classificazioni sono coppie di valori che hanno diverso significato:

- (totale, esclusiva)
- (totale, sovrapposta)
- (parziale, esclusiva)
- (parziale, sovrapposta)

Con totale, il genitore ha ogni occorrenza posseduta da almeno un'entità figlia. In caso contrario è parziale.
Con esclusiva, il genitore ha ogni occorrenza che si ripete solamente in una delle entità figlie. In caso un'occorrenza del genitore sia di più entità figlie, si dice sovrapposta.
Sintassi grafica. Un esempio è la generalizzazione "Persona" con le specializzazioni "Uomo" e "Donna". La sintassi grafica:

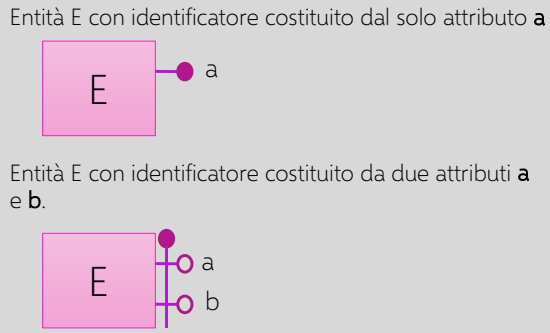


4. *Si illustri il concetto/costrutto di **identificatore** nel modello Entità-Relazione.*

Semantica. Gli identificatori descrivono i concetti (attributi/entità) dello schema che consentono di identificare in maniera univoca le occorrenze delle entità. Devono essere specificati per ogni entità e non possono apparire all'interno di relazioni. Un identificatore può essere:

- Interno, ovvero viene scelto un attributo dell'entità;
- Esterno, viene scelto un identificatore di un'altra identità;

È possibile utilizzare sia identificatori interni ed esterni insieme.
Sintassi grafica. Un esempio è l'entità "Studente" che possiede come identificatore la "Matricola" poiché unica. La sintassi grafica:



5. *Si illustri il concetto/costrutto di **superchiave** nel modello Entità-Relazione.*

Semantica. Una superchiave è un'insieme di attributi che non contiene tuple duplicate al suo interno. Una superchiave è una chiave prima se e solo se è una superchiave minimale. Invece, una chiave primaria è sempre superchiave (non viceversa!).
Sintassi grafica. Non esiste una sintassi grafica poiché è un concetto, ma un esempio:

Tabella

Matricola	Cognome	Nome	Data di nascita	Ufficio
2231	Rossi	Mario	22/08/1984	marketing
2232	Verdi	Paolo	11/03/1990	marketing
2233	Bianchi	Mario	07/05/1995	vendite
2234	Rossi	Giovanni	16/01/1978	personale

superchiave

Nessuna tupla si ripete, quindi "Matricola, Cognome, Nome" è una superchiave valida. Non è minimale poiché esiste "Matricola" che è chiave primaria e superchiave minimale.

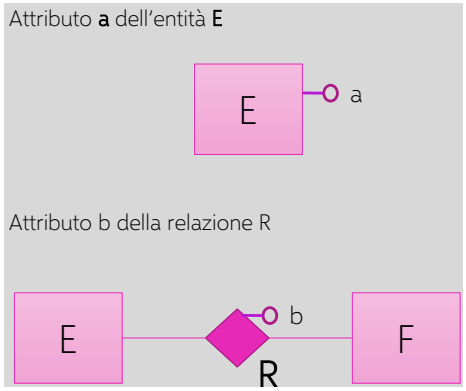
6. *Si illustri il concetto/costrutto di **attributo** nel modello Entità-Relazione.*

Semantica. Gli attributi descrivono le proprietà elementari di entità o relazioni che sono di interesse ai fini dell'applicazione. Ogni attributo ha un suo dominio e quindi può essere visto come una funzione che ha come dominio le istanze dell'entità/relazione e come codominio l'insieme dei valori ammissibili:

$$f_a : I(E) \to D$$

Dove a è un attributo dell'entità E, $I(E)$ è l'insieme delle istanze di E e D è l'insieme dei valori ammissibili.

Sintassi grafica. Un esempio di attributo è "Cognome", "Stipendio" ed "Età" dell'entità "Impiegato". La sintassi grafica è la seguente:



Istanza. L'istanza si ottiene tramite una funzione che data un'istanza dell'entità E (o relazione R), restituisce l'attributo a:

$$\text{valore di } a \text{ su } e = f_a(e)$$

2 Esercizi terzo parziale

2.1 B$^+$-tree

2.1.1 Esercizio 1

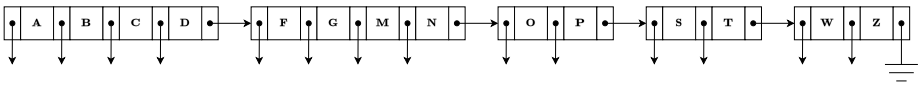
Costruire un B$^+$-tree di fan-out = 5 con i seguenti nodi foglia: (A, B, C, D), (F, G, M, N), (O, P), (S, T), (W, Z). I vincoli di riempimento sono:

- $2 \leq$ #chiavi ≤ 4
- $3 \leq$ #puntatori ≤ 5

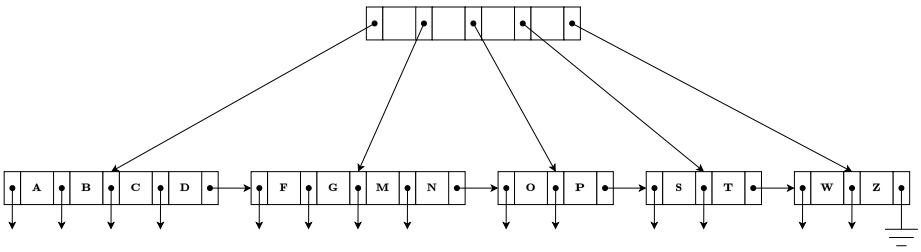
Dopodiché, inserire il valore chiave H nel B$^+$-tree ottenuto precedentemente. Infine, l'esercizio si conclude eseguendo una rimozione del valore chiave Z ottenuto precedentemente.

Soluzione

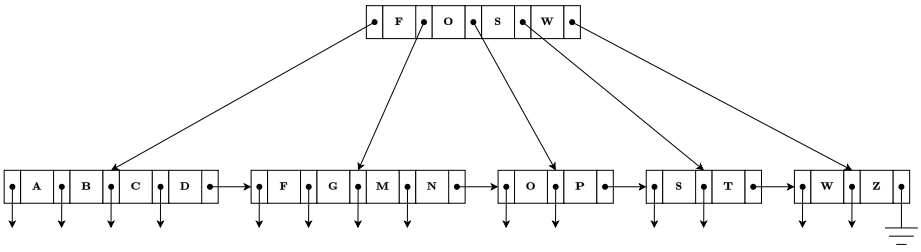
Il primo passo è costruire i vari livelli dei nodi foglia:



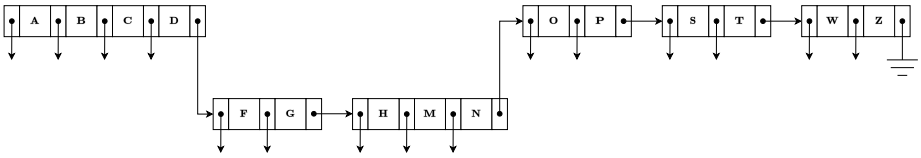
Adesso è necessario costruire tutti i puntatori richiesti. Fan-out è uguale a 5 quindi viene costruito un nodo intermedio con 5 puntatori e si collegano tutti i nodi:



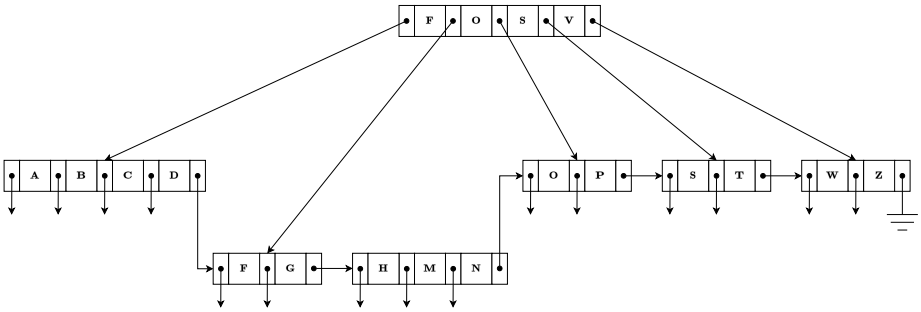
Adesso si aggiungono le lettere che devono essere raggiunte dopo aver visitato ogni nodo:



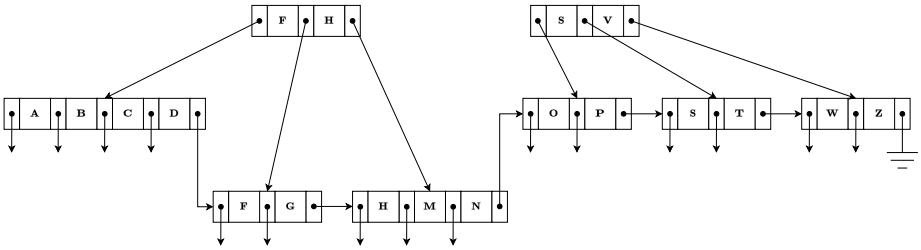
Per inserire il valore chiave è necessario avere a disposizione una posizione libera. Tuttavia, questo non è possibile, per cui viene applicato uno split. Viene divisa la radice contenente (F, G, M, N) così da inserire la chiave H tra la G e la M:



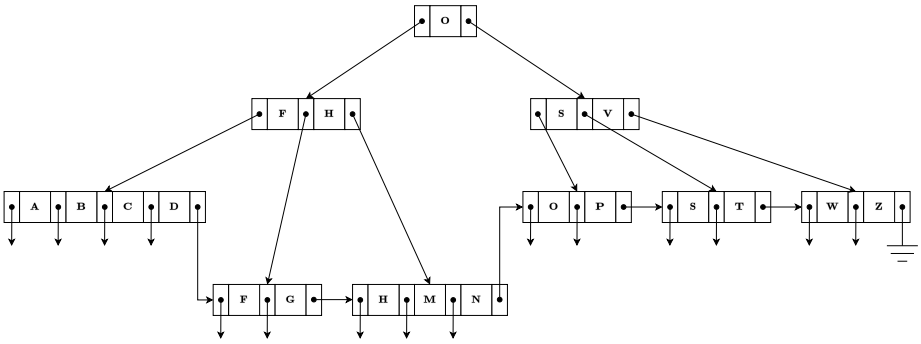
A questo punto è necessario riadattare il nodo radice che attualmente punta ad un nodo errato (attenzione c'è un errore, il nodo V in realtà è il nodo W):



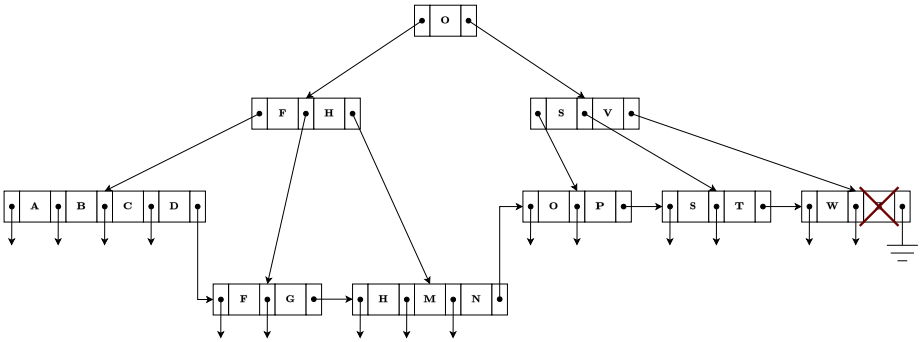
Per farlo, è necessario eseguire una divisione anche nel nodo radice aggiustando i valori delle chiavi (attenzione c'è un errore, il nodo V in realtà è il nodo W):



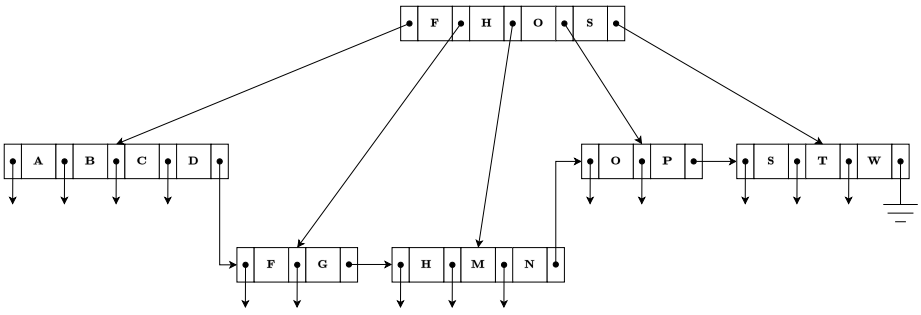
E infine, collegare i due nodi divisi con un nodo di congiunzione. Inoltre, quest'ultimo viene riempito con un valore chiave (attenzione c'è un errore, il nodo V in realtà è il nodo W):



La rimozione della chiave Z comporta che l'ultimo nodo abbia come chiave solo il valore W. Questo comporta un'irregolarità poiché il numero minimo di ogni chiave in ogni nodo deve essere minimo di due e massimo di quattro. Per cui è necessario effettuare un merge:



Eliminazione della chiave Z.



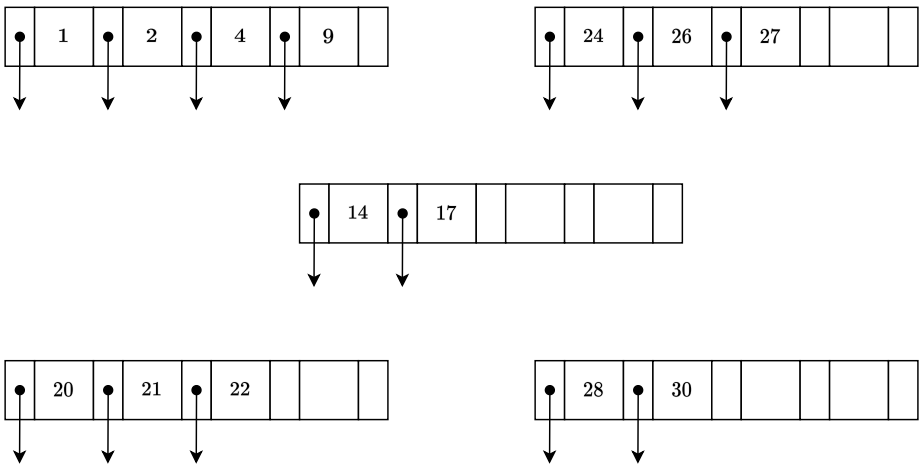
Merge degli ultimi due nodi.

2.1.2 Esercizio 2

Data la seguente lista di possibili valori chiave:

$$L = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30)$$

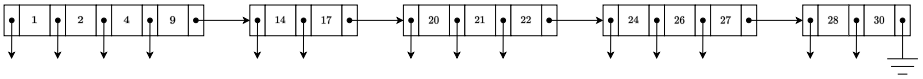
Costruire un B+-tree (*fan-out* = 5) che contenga i seguenti nodi foglia:



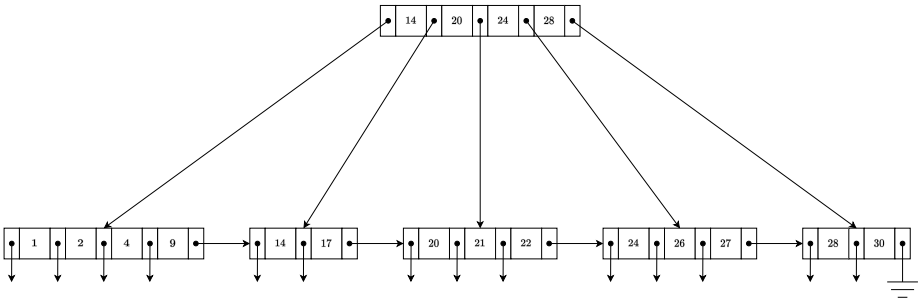
Mostrare l'albero dopo l'inserimento del valore chiave 5 e partendo da questo risultato, mostrare l'albero dopo la cancellazione del valore chiave 28.

Soluzione

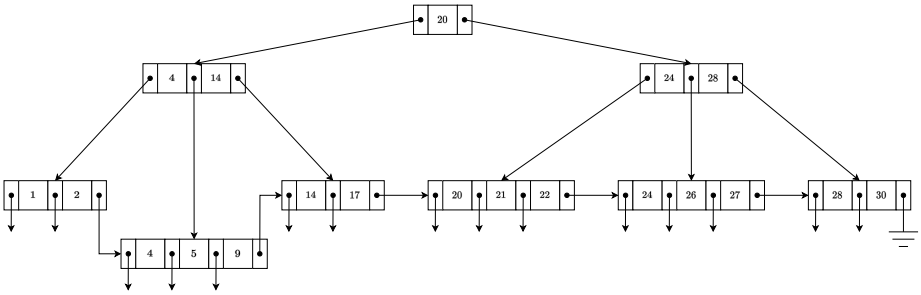
Si costruisce la lista:



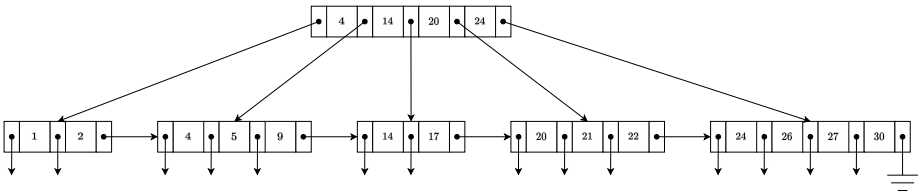
E poi l'albero:



Si inserisce il valore 5:



E infine si rimuove il valore 28:



2.2 Verificare che uno schedule sia VSR (View-serializzabile)

2.2.1 Esercizio 1 - Perdita di aggiornamento

Date due transazioni T_1 e T_2 di seguito descritte:

$$\begin{array}{rcl} T_1 & : & r_1(x)\ w_1(x) \\ T_2 & : & r_2(x)\ w_2(x) \end{array}$$

Lo schedule che rappresenta l'anomalia è il seguente:

$$S_{PA} = r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)$$

Per verificare che uno schedule sia VSR o meno, è necessario caratterizzare S_{PA} calcolando l'insieme delle relazioni LeggeDa e l'insieme delle ScrittureFinali.

Quindi, per l'insieme LeggeDa viene cercato per ogni operazione di lettura, una precedente scrittura sulla stessa risorsa fatta da un'altra transazione. In questo caso, l'insieme è vuoto poiché nessuna risorsa scrive prime di una lettura.

Invece, per l'insieme ScrittureFinali, per ogni risorsa indicata nello schedule si specifica l'ultima scrittura eseguita. In questo caso, l'unica risorsa è x e l'ultima scrittura è $w_1(x)$.

Quindi, gli insiemi sono composti da:

$$\begin{array}{rcl} \text{LeggeDa}(S_{PA}) & = & \emptyset \\ \text{ScrittureFinali}(S_{PA}) & = & \{w_1(x)\} \end{array}$$

Adesso si generano tutti i possibili schedule seriali che eseguono le due transazioni. Essi si ottengono generando le possibili permutazioni dell'insieme di transazioni che partecipano allo schedule. In questo caso, date solo due transazioni T_1 e T_2, le possibili combinazioni sono:

$$\begin{array}{rcl} S_1 = T_1\,T_2 & = & r_1(x)\ w_1(x)\ r_2(x)\ w_2(x) \\ S_2 = T_2\,T_1 & = & r_2(x)\ w_2(x)\ r_1(x)\ w_1(x) \end{array}$$

Adesso, si verifica che almeno uno dei due schedule seriali è view-equivalente a S_{PA}.

Verifica partendo dallo schedule S_1:

1. Creazione dell'insieme LeggeDa(S_1). Data la sequenza:

$$S_1 = r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)$$

   L'unica scrittura che precede una lettura è $w_1(x)$. Quindi, l'insieme è composto dalla scrittura che avviene prima della lettura e da quest'ultima:

$$\text{LeggeDa}(S_1) = \{(r_2(x), w_1(x))\}$$

2. Creazione dell'insieme ScrittureFinali(S_1). Data la sequenza:

$$S_1 = r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)$$

   L'unica risorsa x ha come ultima scrittura $w_2(x)$, quindi l'insieme è composto da:

$$\text{ScrittureFinali}(S_1) = \{w_2(x)\}$$

3. Si esegue il confronto degli insiemi ottenuti da S_1 e dagli insiemi ottenuti da S_{PA}:

$$\begin{array}{lcl} \text{LeggeDa}(S_{PA}) & = & \emptyset \\ \text{LeggeDa}(S_1) & = & \{(r_2(x), w_1(x))\} \\ \text{ScrittureFinali}(S_{PA}) & = & \{w_1(x)\} \\ \text{ScrittureFinali}(S_1) & = & \{w_2(x)\} \end{array}$$

   Come è evidente, nessuno dei due insiemi è equivalente:

$$\begin{array}{lcl} \text{LeggeDa}(S_{PA}) & \not\equiv & \text{LeggeDa}(S_1) \\ \text{ScrittureFinali}(S_{PA}) & \not\equiv & \text{ScrittureFinali}(S_1) \end{array}$$

   Quindi, è possibile concludere che S_{PA} non è view-equivalente a S_1.

Verifica partendo dallo schedule S_2:

1. Creazione dell'insieme LeggeDa(S_2). Data la sequenza:

$$S_2 = r_2(x)\ w_2(x)\ r_1(x)\ w_1(x)$$

   L'unica scrittura che precede una lettura è $w_2(x)$. Quindi, l'insieme è composto dalla scrittura che avviene prima della lettura e da quest'ultima:

$$\text{LeggeDa}(S_2) = \{(r_1(x), w_2(x))\}$$

2. Creazione dell'insieme ScrittureFinali(S_2). Data la sequenza:

$$S_2 = r_2(x)\ w_2(x)\ r_1(x)\ w_1(x)$$

   L'unica risorsa x ha come ultima scrittura $w_2(x)$, quindi l'insieme è composto da:

$$\text{ScrittureFinali}(S_2) = \{w_1(x)\}$$

3. Si esegue il confronto degli insiemi ottenuti da S_1 e dagli insiemi ottenuti da S_{PA}:

$$\begin{array}{lcl} \text{LeggeDa}(S_{PA}) & = & \emptyset \\ \text{LeggeDa}(S_1) & = & \{(r_1(x), w_2(x))\} \\ \text{ScrittureFinali}(S_{PA}) & = & \{w_1(x)\} \\ \text{ScrittureFinali}(S_1) & = & \{w_1(x)\} \end{array}$$

   Come è evidente, soltanto uno dei due insiemi è equivalente:

$$\begin{array}{lcl} \text{LeggeDa}(S_{PA}) & \not\equiv & \text{LeggeDa}(S_1) \\ \text{ScrittureFinali}(S_{PA}) & \equiv & \text{ScrittureFinali}(S_1) \end{array}$$

   Quindi, è possibile concludere che S_{PA} non è view-equivalente a S_1 poiché entrambi gli insiemi non sono equivalenti.

L'esercizio si conclude qua. Nessuna combinazione è view-equivalente allo schedule di partenza S_{PA}. Quindi, si conclude affermando che S_{PA} non è VSR.

2.2.2 Esercizio 2 - Lettura inconsistente

Date due transazioni T_1 e T_2 di seguito descritte:

$$\begin{array}{lcl} T_1 & : & r_1(x)\ r'_1(x) \\ T_2 & : & r_2(x)\ w_2(x) \end{array}$$

Lo schedule che rappresenta l'anomalia è il seguente:

$$S_{LI} = r_1(x)\ r_2(x)\ w_2(x)\ r'_1(x)$$

Per verificare che uno schedule sia VSR o meno, è necessario caratterizzare S_{LI} calcolando l'insieme delle relazioni LeggeDa e l'insieme delle ScrittureFinali.

Quindi, per l'insieme LeggeDa viene cercato per ogni operazione di lettura, una precedente scrittura sulla stessa risorsa fatta da un'altra transazione. In questo caso, l'insieme è composto da $w_2(x)$ perché precede $r'_1(x)$.

Invece, per l'insieme ScrittureFinali, per ogni risorsa indicata nello schedule si specifica l'ultima scrittura eseguita. In questo caso, l'unica risorsa è x e l'ultima scrittura è $w_2(x)$.

Quindi, gli insiemi sono composti da:

$$\begin{array}{rcl} \text{LeggeDa}(S_{LI}) & = & \{(r'_1(x), w_2(x))\} \\ \text{ScrittureFinali}(S_{LI}) & = & \{w_2(x)\} \end{array}$$

Adesso si generano tutti i possibili schedule seriali che eseguono le due transazioni. Essi si ottengono generando le possibili permutazioni dell'insieme di transazioni che partecipano allo schedule. In questo caso, date solo due transazioni T_1 e T_2, le possibili combinazioni sono:

$$\begin{array}{lcl} S_1 = T_1\,T_2 & = & r_1(x)\ r'_1(x)\ r_2(x)\ w_2(x) \\ S_2 = T_2\,T_1 & = & r_2(x)\ w_2(x)\ r_1(x)\ r'_1(x) \end{array}$$

Adesso, si verifica che almeno uno dei due schedule seriali è view-equivalente a S_{LI}.

Verifica partendo dallo schedule S_1:

1. Creazione dell'insieme LeggeDa(S_1). Data la sequenza:

$$S_1 = r_1(x)\ r'_1(x)\ r_2(x)\ w_2(x)$$

   L'unica scrittura che precede una lettura è $w_1(x)$. Quindi, l'insieme è composto dalla scrittura che avviene prima della lettura e da quest'ultima:

$$\text{LeggeDa}(S_1) = \emptyset$$

2. Creazione dell'insieme ScrittureFinali(S_1). Data la sequenza:

$$S_1 = r_1(x)\ r'_1(x)\ r_2(x)\ w_2(x)$$

   L'unica risorsa x ha come ultima scrittura $w_2(x)$, quindi l'insieme è composto da:

$$\text{ScrittureFinali}(S_1) = \{w_2(x)\}$$

3. Si esegue il confronto degli insiemi ottenuti da S_1 e dagli insiemi ottenuti da S_{LI}:

$$\begin{array}{lcl} \text{LeggeDa}(S_{LI}) & = & \{(r'_1(x), w_2(x))\} \\ \text{LeggeDa}(S_1) & = & \emptyset \\ \text{ScrittureFinali}(S_{LI}) & = & \{w_2(x)\} \\ \text{ScrittureFinali}(S_1) & = & \{w_2(x)\} \end{array}$$

   Come è evidente, soltanto uno dei due insiemi è equivalente:

$$\begin{array}{lcl} \text{LeggeDa}(S_{LI}) & \not\equiv & \text{LeggeDa}(S_1) \\ \text{ScrittureFinali}(S_{LI}) & \equiv & \text{ScrittureFinali}(S_1) \end{array}$$

   Quindi, è possibile concludere che S_{LI} non è view-equivalente a S_1.

Verifica partendo dallo schedule S_2:

1. Creazione dell'insieme LeggeDa(S_2). Data la sequenza:

$$S_2 = r_2(x)\ w_2(x)\ r_1(x)\ r'_1(x)$$

   L'unica scrittura che precede due letture è $w_2(x)$. Quindi, l'insieme è composto dalla scrittura che avviene con due letture:

$$\text{LeggeDa}(S_2) = \{(r'_1(x), w_2(x)), (r_1(x), w_2(x))\}$$

2. Creazione dell'insieme ScrittureFinali(S_2). Data la sequenza:

$$S_2 = r_2(x)\ w_2(x)\ r_1(x)\ r'_1(x)$$

   L'unica risorsa x ha come ultima scrittura $w_2(x)$, quindi l'insieme è composto da:

$$\text{ScrittureFinali}(S_2) = \{w_2(x)\}$$

3. Si esegue il confronto degli insiemi ottenuti da S_1 e dagli insiemi ottenuti da S_{PA}:

$$\begin{array}{lcl} \text{LeggeDa}(S_{LI}) & = & \{(r'_1(x), w_2(x))\} \\ \text{LeggeDa}(S_2) & = & \{(r'_1(x), w_2(x)), (r_1(x), w_2(x))\} \\ \text{ScrittureFinali}(S_{LI}) & = & \{w_2(x)\} \\ \text{ScrittureFinali}(S_2) & = & \{w_2(x)\} \end{array}$$

   Come è evidente, soltanto uno dei due insiemi è equivalente:

$$\begin{array}{lcl} \text{LeggeDa}(S_{PA}) & \not\equiv & \text{LeggeDa}(S_1) \\ \text{ScrittureFinali}(S_{PA}) & \equiv & \text{ScrittureFinali}(S_1) \end{array}$$

   Quindi, è possibile concludere che S_{LI} non è view-equivalente a S_2 poiché entrambi gli insiemi non sono equivalenti.

L'esercizio si conclude qua. Nessuna combinazione è view-equivalente allo schedule di partenza S_{LI}. Quindi, si conclude affermando che S_{LI} non è VSR.

2.2.3 Sintesi dell'algoritmo

In sintesi l'algoritmo per capire se uno schedule è VSR:

1. Si tiene bene in considerazione lo schedule che rappresenta l'anomalia, ovvero quello che viene dato;
2. Si compongono i due insiemi:
   (a) Creazione insieme LeggeDa cercando per ogni operazione di lettura (r_i (risorsa)) una precedente operazione di scrittura sulla stessa risorsa fatta da un'altra transazione. Nel caso in cui si trovi, si aggiunge all'insieme la scrittura incriminata e la relativa lettura;
   (b) Creazione insieme ScrittureFinali cercando per ogni risorsa indicata nello schedule l'ultima scrittura eseguita.
3. Date le varie transazioni, si creano tutti i possibili schedule creando così una lista;
4. Si verifica che almeno uno schedule della lista sia view-equivalente allo schedule dato al punto 1. Per farlo si esegue questo piccolo algoritmo:
   (a) Creazione dell'insieme LeggeDa (vedi punto 2.a);
   (b) Creazione dell'insieme ScrittureFinali (vedi punto 2.b);
   (c) Confronto degli insiemi creati precedente con quelli creati per lo schedule dato al punto 1. Se non sono uguali tutti uguali, allora lo schedule creato tramite combinazione non è equivalente allo schedule dato al punto 1. Altrimenti, è possibile affermare di aver trovato una combinazione view-equivalente.
5. Al termine della creazione degli insiemi e dei vari confronti, se esiste almeno una combinazione che è view-equivalente allo schedule del punto 1, allora è possibile affermare che lo schedule di partenza è VSR.

2.3 Verificare che uno schedule sia CSR (Conflict-serializzabile)

2.3.1 Esercizio 1 - Perdita di aggiornamento

Date due transizioni T_1 e T_2:

$$\begin{array}{lcl} T_1 & : & r_1(x)\ w_1(x) \\ T_2 & : & r_2(x)\ w_2(x) \end{array}$$

Lo schedule che rappresenta l'anomalia è il seguente:

$$S_{PA} = r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)$$

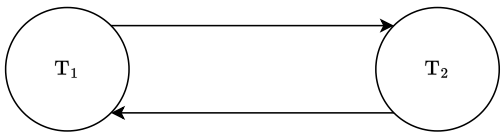
Per verificare CSR è necessario caratterizzare S_{PA} calcolando l'insieme dei conflitti. Si ricorda che due azioni sono in conflitto se operano sullo stesso oggetto e se almeno una di esse è in scrittura (quindi le combinazioni: rw, wr, ww). Quindi, si calcola l'insieme dei conflitti di S_{PA}:

1. $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)$
2. $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)$
3. $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)$

L'insieme è quindi così costituito:

$$\text{Conflitti}(S_{PA}) = \{(r_1(x), w_2(x)), (r_2(x), w_1(x)), (w_2(x), w_1(x))\}$$

Si costruisce il grafo nel seguente modo. Si rappresentano tanti nodi quanti sono le transazioni e ogni arco (orientato) viene tracciato da t_i a t_j se vengono rispettate due condizioni: se c'è almeno un conflitto fra un'azione a_i e un'azione a_j tale che a_i precede a_j. Quindi:



Se il grafo è aciclico allora S_{PA} è CSR. In questo caso, il grafo non è aciclico ma ciclico, per cui S_{PA} non è CSR.

2.3.2 Esercizio 2 - Lettura inconsistente

Date due transizioni T_1 e T_2:

$$\begin{array}{lcl} T_1 & : & r_1(x)\ r_1'(x) \\ T_2 & : & r_2(x)\ w_2(x) \end{array}$$

Lo schedule che rappresenta l'anomalia è il seguente:

$$S_{LI} = r_1(x)\ r_2(x)\ w_2(x)\ r_1'(x)$$

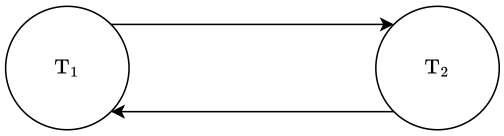
Per verificare CSR è necessario caratterizzare S_{LI} calcolando l'insieme dei conflitti. Si ricorda che due azioni sono in conflitto se operano sullo stesso oggetto e se almeno una di esse è in scrittura (quindi le combinazioni: rw, wr, ww). Quindi, si calcola l'insieme dei conflitti di S_{LI}:

1. $r_1(x)\ r_2(x)\ w_2(x)\ r_1'(x)$
2. $r_1(x)\ r_2(x)\ w_2(x)\ r_1'(x)$

L'insieme è quindi così costituito:

$$\text{Conflitti}(S_{LI}) = \{(r_1(x), w_2(x)), (w_2(x), r_1'(x))\}$$

Si costruisce il grafo nel seguente modo. Si rappresentano tanti nodi quanti sono le transazioni e ogni arco (orientato) viene tracciato da t_i a t_j se vengono rispettate due condizioni: se c'è almeno un conflitto fra un'azione a_i e un'azione a_j tale che a_i precede a_j. Quindi:



Se il grafo è aciclico allora S_{LI} è CSR. In questo caso, il grafo non è aciclico ma ciclico, per cui S_{LI} non è CSR.

2.3.3 Sintesi dell'algoritmo

In sintesi l'algoritmo per capire se uno schedule è CSR:

1. Si calcola l'insieme dei conflitti. Un conflitto si manifesta quando due azioni **differenti** operano sullo stesso oggetto e quando almeno una di esse è in scrittura. Quindi, le combinazioni che possono esserci sono: rw, wr, ww;
2. Si costruire il grafo dall'insieme dei conflitti. I nodi rappresentano le transizioni e gli archi si disegnano solo se due azioni non riguardano la stessa transizione.

2.4 Verificare che uno schedule sia NonSR, VSR e/o CSR

2.4.1 Testo esercizio

Classificare i seguenti schedule (come: NonSR, VSR, CSR); nel caso uno schedule sia VSR oppure CSR, indicare tutti gli schedule seriali a esso equivalenti.

1. $S_1 = r_1(x)\ w_1(x)\ r_2(z)\ r_1(y)\ w_1(y)\ r_2(x)\ w_2(x)\ w_2(z)$
2. $S_2 = r_1(x)\ w_1(x)\ w_3(x)\ r_2(y)\ r_3(y)\ w_3(y)\ w_1(y)\ r_2(x)$
3. $S_3 = r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
4. $S_4 = r_1(x)\, r_3(y)\, w_1(y)\, w_4(x)\, w_1(t)\, w_5(x)\, r_2(z)\, r_3(z)\, w_2(z)\, w_5(z)\, r_4(t)\, r_5(t)$

2.4.2 Schedule 1

Dato il seguente schedule:

$$S_1 = r_1(x)\ w_1(x)\ r_2(z)\ r_1(y)\ w_1(y)\ r_2(x)\ w_2(x)\ w_2(z)$$

Le transizioni sono:

$$\begin{array}{lcl} T_1 & : & r_1(x)\ w_1(x)\ r_1(y)\ w_1(y) \\ T_2 & : & r_2(z)\ r_2(x)\ w_2(x)\ w_2(z) \end{array}$$

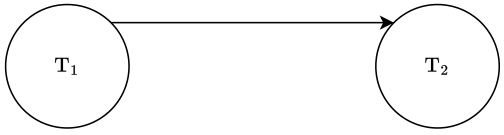
Si verifica se è CSR. Quindi, si crea l'insieme dei conflitti:

1. $r_1(x)\ w_1(x)\ r_2(z)\ r_1(y)\ w_1(y)\ r_2(x)\ w_2(x)\ w_2(z)$
2. $r_1(x)\ w_1(x)\ r_2(z)\ r_1(y)\ w_1(y)\ r_2(x)\ w_2(x)\ w_2(z)$
3. $r_1(x)\ w_1(x)\ r_2(z)\ r_1(y)\ w_1(y)\ r_2(x)\ w_2(x)\ w_2(z)$

Quindi l'insieme è:

$$\text{Conflitti}(S_1) = \{(r_1(x)\ w_2(x)), (w_1(x)\ r_2(x)), (w_1(x)\ w_2(x))\}$$

Si costruisce il grafo:



Il ciclo è aciclico quindi è S_1 è CSR. Dato che CSR $\subset$ VSR, allora S_1 è anche VSR.

2.4.3 Schedule 2

Dato il seguente schedule:

$$S_2 = r_1(x)\ w_1(x)\ w_3(x)\ r_2(y)\ r_3(y)\ w_3(y)\ w_1(y)\ r_2(x)$$

Le transazioni sono:

$$\begin{array}{lcl} T_1 & : & r_1(x)\ w_1(x)\ w_1(y) \\ T_2 & : & r_2(y)\ r_2(x) \\ T_3 & : & w_3(x)\ r_3(y)\ w_3(y) \end{array}$$

Si verifica se è VSR. Si inizia analizzando l'insieme S_2:

$$\begin{array}{lcl} \text{LeggeDa}(S_2) & = & \{(r_2(x), w_3(x))\} \\ \text{ScrittureFinali}(S_2) & = & \{w_3(x), w_1(y)\} \end{array}$$

Dato che è impossibile provare tutte le combinazioni (3 transazioni e quindi $3! = 6$), si fanno alcune considerazioni. Per esempio, dato che nelle LeggeDa si deve mantenere l'ordine $(r_2(x), w_3(x))$, e sapendo che r_2 appartiene a T_2 e w_3 a T_3, si può concludere che T_3 deve per forza precedere T_2. Quindi, le combinazioni si riducono a:

- $T_1\ T_3\ T_2$
- $T_3\ T_1\ T_2$
- $T_3\ T_2\ T_1$

Tuttavia, se T_3 anticipa T_2, tutte le combinazioni avranno come insieme LeggeDa almeno i due valori:

$$\text{LeggeDa}(S_2) = \{(r_2(x), w_3(x)), (r_2(y)\ w_3(y))\}$$

Quindi, è possibile concludere che nessuna combinazione ha un insieme LeggeDa equivalente al LeggeDa di S_2. È possibile concludere che S_2 non è VSR.

2.4.4 Schedule 3

Dato il seguente schedule:

$$S_3 = r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$$

Le transazioni sono:

$$\begin{array}{lcl} T_1 & : & r_1(x)\ w_1(x)\ w_1(y)\ w_1(z) \\ T_2 & : & r_2(x)\ w_2(x) \\ T_3 & : & r_3(x)\ w_3(y)\ w_3(x) \\ T_4 & : & r_4(z) \\ T_5 & : & w_5(x)\ w_5(y)\ r_5(z) \end{array}$$

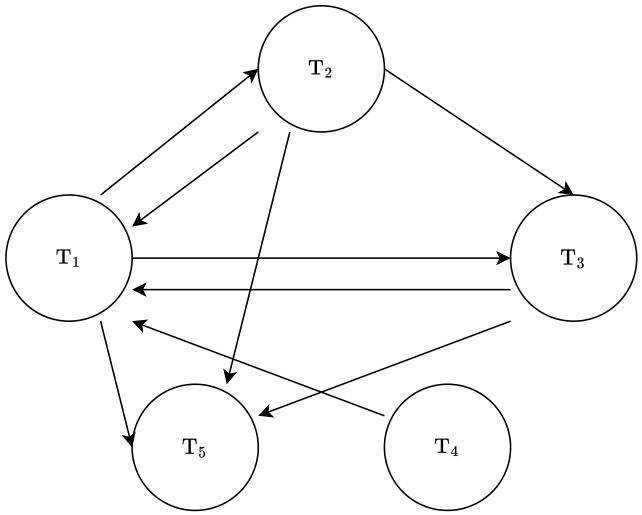
Si verifica se CSR. Quindi, si cerca l'insieme di conflitti:

1. $\boldsymbol{r_1(x)}\, r_2(x)\, \boldsymbol{w_2(x)}\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
2. $\boldsymbol{r_1(x)}\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, \boldsymbol{w_3(x)}\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
3. $\boldsymbol{r_1(x)}\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, \boldsymbol{w_5(x)}\, w_1(z)\, w_5(y)\, r_5(z)$
4. $r_1(x)\, \boldsymbol{r_2(x)}\, w_2(x)\, r_3(x)\, r_4(z)\, \boldsymbol{w_1(x)}\, w_3(y)\, w_3(x)\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
5. $r_1(x)\, \boldsymbol{r_2(x)}\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, \boldsymbol{w_3(x)}\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
6. $r_1(x)\, \boldsymbol{r_2(x)}\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, \boldsymbol{w_5(x)}\, w_1(z)\, w_5(y)\, r_5(z)$
7. $r_1(x)\, r_2(x)\, \boldsymbol{w_2(x)}\, \boldsymbol{r_3(x)}\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
8. $r_1(x)\, r_2(x)\, \boldsymbol{w_2(x)}\, r_3(x)\, r_4(z)\, \boldsymbol{w_1(x)}\, w_3(y)\, w_3(x)\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
9. $r_1(x)\, r_2(x)\, \boldsymbol{w_2(x)}\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, \boldsymbol{w_3(x)}\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
10. $r_1(x)\, r_2(x)\, \boldsymbol{w_2(x)}\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, \boldsymbol{w_5(x)}\, w_1(z)\, w_5(y)\, r_5(z)$
11. $r_1(x)\, r_2(x)\, w_2(x)\, \boldsymbol{r_3(x)}\, r_4(z)\, \boldsymbol{w_1(x)}\, w_3(y)\, w_3(x)\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
12. $r_1(x)\, r_2(x)\, w_2(x)\, \boldsymbol{r_3(x)}\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, \boldsymbol{w_5(x)}\, w_1(z)\, w_5(y)\, r_5(z)$
13. $r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, \boldsymbol{r_4(z)}\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, w_5(x)\, \boldsymbol{w_1(z)}\, w_5(y)\, r_5(z)$
14. $r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, \boldsymbol{w_1(x)}\, w_3(y)\, \boldsymbol{w_3(x)}\, w_1(y)\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
15. $r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, \boldsymbol{w_1(x)}\, w_3(y)\, w_3(x)\, w_1(y)\, \boldsymbol{w_5(x)}\, w_1(z)\, w_5(y)\, r_5(z)$
16. $r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, \boldsymbol{w_3(y)}\, w_3(x)\, \boldsymbol{w_1(y)}\, w_5(x)\, w_1(z)\, w_5(y)\, r_5(z)$
17. $r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, \boldsymbol{w_3(y)}\, w_3(x)\, w_1(y)\, w_5(x)\, w_1(z)\, \boldsymbol{w_5(y)}\, r_5(z)$
18. $r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, \boldsymbol{w_3(x)}\, w_1(y)\, \boldsymbol{w_5(x)}\, w_1(z)\, w_5(y)\, r_5(z)$
19. $r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, \boldsymbol{w_1(y)}\, w_5(x)\, w_1(z)\, \boldsymbol{w_5(y)}\, r_5(z)$
20. $r_1(x)\, r_2(x)\, w_2(x)\, r_3(x)\, r_4(z)\, w_1(x)\, w_3(y)\, w_3(x)\, w_1(y)\, w_5(x)\, \boldsymbol{w_1(z)}\, w_5(y)\, \boldsymbol{r_5(z)}$

L'insieme dei conflitti è quindi formato da:

$$\begin{aligned}\text{Conflitti}\,(S_3) = \{\quad & (r_1\,(x)\ w_2\,(x))\,,(r_1\,(x)\ w_3\,(x))\,,(r_1\,(x)\ w_5\,(x))\,,\\ & (r_2\,(x)\ w_1\,(x))\,,(r_2\,(x)\ w_3\,(x))\,,(r_2\,(x)\ w_5\,(x))\,,\\ & (w_2\,(x)\ r_3\,(x))\,,(w_2\,(x)\ w_1\,(x))\,,(w_2\,(x)\ w_3\,(x))\,,(w_2\,(x)\ w_5\,(x))\,,\\ & (r_3\,(x)\ w_1\,(x))\,,(r_3\,(x)\ w_5\,(x))\,,\\ & (r_4\,(z)\ w_1\,(z))\,,\\ & (w_1\,(x)\ w_3\,(x))\,,(w_1\,(x)\ w_5\,(x))\,,\\ & (w_3\,(y)\ w_1\,(y))\,,(w_3\,(y)\ w_5\,(y))\,,\\ & (w_3\,(x)\ w_5\,(x))\,,\\ & (w_1\,(y)\ w_5\,(y))\,,\\ & (w_1\,(z)\ r_5\,(z)) \qquad\qquad \}\end{aligned}$$

Adesso si crea il grafo dei conflitti. Le transazioni sono 5, quindi ci saranno 5 nodi:



Il grafo non è aciclico, quindi S_3 non è CSR, quindi si verifica se può essere VSR. Si inizia calcolando l'insieme LeggeDa e ScrittureFinali:

$$\begin{aligned}\text{LeggeDa}\,(S_3) &= \{(r_3\,(x)\ w_2\,(x))\,,(r_5\,(z)\ w_1\,(z))\}\\ \text{ScrittureFinali}\,(S_3) &= \{w_5\,(x)\,,w_1\,(z)\,,w_5\,(y)\}\end{aligned}$$

Adesso si cerca almeno una combinazione tra le 5 transizioni, tale per cui una di esse sia view-equivalente a S_3. Con una considerazione rapida è possibile già ottenere il risultato finale. Guardando l'insieme delle ScrittureFinali, è possibile vedere che la 5^a transizione dovrebbe avere tra la scrittura sulla risorsa x e tra la scrittura sulla risorsa y, la scrittura sulla risorsa z. Questo non è possibile poiché quest'ultima appartiene alla transazione 1. Per cui, è già possibile affermare che nessuna combinazione tra le 5 transazioni, può dare una combinazione view-equivalente a S_3. Si conclude che S_3 non è VSR e quindi è NonSR.

2.4.5 Schedule 4

Dato il seguente schedule:

$$S_4 = r_1(x)\, r_3(y)\, w_1(y)\, w_4(x)\, w_1(t)\, w_5(x)\, r_2(z)\, r_3(z)\, w_2(z)\, w_5(z)\, r_4(t)\, r_5(t)$$

Le transazioni sono:

$$\begin{array}{lcl} T_1 & : & r_1(x)\ w_1(y)\ w_1(t) \\ T_2 & : & r_2(z)\ w_2(z) \\ T_3 & : & r_3(y)\ r_3(z) \\ T_4 & : & w_4(x)\ r_4(t) \\ T_5 & : & w_5(x)\ w_5(z)\ r_5(t) \end{array}$$

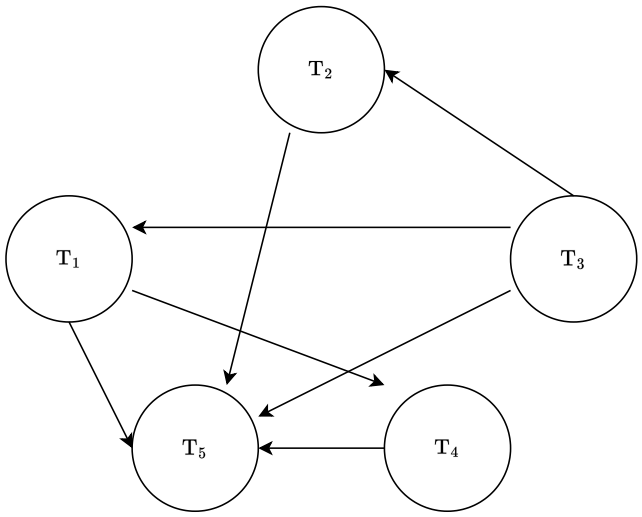
Si verifica CSR prima di tutto. Quindi, si inizia costruendo l'insieme dei conflitti:

1. $\boldsymbol{r_1(x)}\, r_3(y)\, w_1(y)\, \boldsymbol{w_4(x)}\, w_1(t)\, w_5(x)\, r_2(z)\, r_3(z)\, w_2(z)\, w_5(z)\, r_4(t)\, r_5(t)$
2. $\boldsymbol{r_1(x)}\, r_3(y)\, w_1(y)\, w_4(x)\, w_1(t)\, \boldsymbol{w_5(x)}\, r_2(z)\, r_3(z)\, w_2(z)\, w_5(z)\, r_4(t)\, r_5(t)$
3. $r_1(x)\, \boldsymbol{r_3(y)\, w_1(y)}\, w_4(x)\, w_1(t)\, w_5(x)\, r_2(z)\, r_3(z)\, w_2(z)\, w_5(z)\, r_4(t)\, r_5(t)$
4. $r_1(x)\, r_3(y)\, w_1(y)\, \boldsymbol{w_4(x)}\, w_1(t)\, \boldsymbol{w_5(x)}\, r_2(z)\, r_3(z)\, w_2(z)\, w_5(z)\, r_4(t)\, r_5(t)$
5. $r_1(x)\, r_3(y)\, w_1(y)\, w_4(x)\, \boldsymbol{w_1(t)}\, w_5(x)\, r_2(z)\, r_3(z)\, w_2(z)\, w_5(z)\, \boldsymbol{r_4(t)}\, r_5(t)$
6. $r_1(x)\, r_3(y)\, w_1(y)\, w_4(x)\, \boldsymbol{w_1(t)}\, w_5(x)\, r_2(z)\, r_3(z)\, w_2(z)\, w_5(z)\, r_4(t)\, \boldsymbol{r_5(t)}$
7. $r_1(x)\, r_3(y)\, w_1(y)\, w_4(x)\, w_1(t)\, w_5(x)\, \boldsymbol{r_2(z)}\, r_3(z)\, w_2(z)\, \boldsymbol{w_5(z)}\, r_4(t)\, r_5(t)$
8. $r_1(x)\, r_3(y)\, w_1(y)\, w_4(x)\, w_1(t)\, w_5(x)\, r_2(z)\, \boldsymbol{r_3(z)\, w_2(z)}\, w_5(z)\, r_4(t)\, r_5(t)$
9. $r_1(x)\, r_3(y)\, w_1(y)\, w_4(x)\, w_1(t)\, w_5(x)\, r_2(z)\, \boldsymbol{r_3(z)}\, w_2(z)\, \boldsymbol{w_5(z)}\, r_4(t)\, r_5(t)$
10. $r_1(x)\, r_3(y)\, w_1(y)\, w_4(x)\, w_1(t)\, w_5(x)\, r_2(z)\, r_3(z)\, \boldsymbol{w_2(z)\, w_5(z)}\, r_4(t)\, r_5(t)$

Quindi, l'insieme dei conflitti è:

$$\begin{aligned} \text{Conflitti}(S_4) = \{ \quad & (r_1(x)\ w_4(x)), (r_1(x)\ w_5(x)), \\ & (r_3(y)\ w_1(y)), \\ & (w_4(x)\ w_5(x)), \\ & (w_1(t)\ r_4(t)), (w_1(t)\ r_5(t)) \\ & (r_2(z)\ w_5(z)), \\ & (r_3(z)\ w_2(z)), (r_3(z)\ w_5(z)), \\ & (w_2(z)\ w_5(z)) \qquad\qquad\qquad\qquad \} \end{aligned}$$

Adesso si costruisce il grafo. I nodi sono 5 poiché il numero di transazioni è 5:



Il grafo risulta aciclico, quindi è possibile affermare con certezza che S_4 è CSR e per definizione anche VSR ($CSR \subset VSR$). Per concludere l'esercizio, è necessario calcolare gli schedule seriali equivalenti. Per capire quali combinazioni sono accettate, si fanno alcune considerazioni guardando il grafo:

- Il nodo T_5 non ha archi orientati in uscita, ma solo in entrata. Quindi, è possibile affermare che la transazione T_5 deve essere l'ultima.
- Il nodo T_3 non ha archi orientati in entrata, ma solo in uscita. Quindi, è possibile affermare che la transazione T_3 deve essere la prima.
- I nodi T_2 e T_1 sono gli unici che vengono raggiunti da T_3, ovvero dal nodo d'inizio. Quindi, è possibile affermare che le uniche combinazioni potranno essere:
  1. T_3, T_1, T_2, T_4, T_5
  2. T_3, T_1, T_4, T_2, T_5
  3. T_3, T_2, T_1, T_4, T_5
  4. T_3, T_2, T_4, T_1, T_5
- Dal nodo T_1 è possibile andare al nodo T_4 senza problemi ma non viceversa poiché altrimenti si creerebbe una ciclo. Quindi le uniche combinazioni ammesse sono:
  1. T_3, T_1, T_2, T_4, T_5
  2. T_3, T_1, T_4, T_2, T_5
  3. T_3, T_2, T_1, T_4, T_5
  4. T_3, T_2, T_4, T_1, T_5

2.5 Ottimizzazione e stima di costo

2.5.1 Esercizio 1

Si consideri il seguente schema relazionale contenete le ricette di una catena di ristoranti:

$$\begin{aligned} \text{INGREDIENTE} &\quad (\underline{\text{Codice}}, \text{Nome}, \text{Calorie})\,; \\ \text{COMPOSIZIONE} &\quad (\underline{\text{Ricetta, Ingrediente}}, \text{Quantità})\,; \\ \text{RICETTA} &\quad (\underline{\text{CodiceRicetta}}, \text{Nome}, \text{Regione}, \text{TempoPreparazione})\,. \end{aligned}$$

N.B.: la quantità della tabella COMPOSIZIONE è espressa in grammi.

Vincoli di integrità referenziale:

$$\begin{aligned} \text{COMPOSIZIONE.Ricetta} &\rightarrow \text{RICETTA} \\ \text{COMPOSIZIONE.Ingrediente} &\rightarrow \text{INGREDIENTE} \end{aligned}$$

Data la seguente interrogazione SQL che consente di: *trovare gli ingredienti usati in ricette della Regione Veneto, riportando, il codice della ricetta e il nome e le calorie dell'ingrediente.*

```sql
SELECT  R.CodiceRicetta, I.Nome, I.Calorie
FROM    RICETTA R JOIN COMPOSIZIONE C
        ON R.CodiceRicetta = C.Ricetta
        JOIN INGREDIENTE I
        ON C.Ingrediente = I.Codice
WHERE   R.Regione = 'Veneto'
```

Calcolare il costo dell'interrogazione in termini di numero di accessi a memoria secondaria sotto le seguenti ipotesi:

- La selezione su ricetta richiede una scansione sequenziale della tabella RICETTA;
- L'ordine di esecuzione del join è:

$$(\text{RICETTA} \bowtie \text{COMPOSIZIONE}) \bowtie \text{INGREDIENTE}$$

- Le operazioni di join vengono eseguite con la tecnica Nested Loop join con una pagina di buffer disponibile per ogni tabella;
- Il risultato intermedio del primo join viene interamente memorizzato nel buffer;
- $\text{NP}(\text{INGREDIENTE}) = 40$, $\text{NP}(\text{COMPOSIZIONE}) = 200$, $\text{NP}(\text{RICETTA}) = 12$;
- $\text{NR}(\text{INGREDIENTE}) = 1200$, $\text{NR}(\text{COMPOSIZIONE}) = 13000$, $\text{NR}(\text{RICETTA}) = 260$;
- $\text{VAL}(\text{Regione}, \text{RICETTA}) = 20$, $\text{VAL}(\text{Ricetta}, \text{COMPOSIZIONE}) = 260$.

Come cambia il costo se è disponibile un indice B+-tree sull'attributo Codice della tabella INGREDIENTE che ha profondità 2.

Il *nested loop join* è un algoritmo di join che unisce due set usando due cicli nidificati. Quindi, i cicli sono:

1. Viene selezionata la regione Veneta (WHERE R.Regione = 'Veneto') e successivamente eseguito il primo Join:

   $$\text{seleziona\_veneto}\,(\text{RICETTA}) \bowtie \text{COMPOSIZIONE}$$

   Quindi il costo risulta essere una lettura della tabella esterna RICETTA poiché deve essere applicato una condizione WHERE e infine viene eseguita una moltiplicazione:

   $$\begin{aligned}\text{Costo} = \quad & \text{NP}\,(\text{RICETTA}) + \\ & \text{NR}\,(\text{RICETTA con selezione Regione='Veneto'}) \times \\ & \text{NP}\,(\text{COMPOSIZIONE})\end{aligned}$$

2. Il secondo Join non prevede condizioni WHERE e il costo non tiene in considerazione una lettura da una tabella esterna poiché la tabella interessata è già in buffer:

   $$\begin{aligned}\text{Costo} = \quad & \text{NR}\,(\text{COMPOSIZIONE per le ricette con selezione Regione='Veneto'}) \times \\ & \text{NP}\,(\text{INGREDIENTE})\end{aligned}$$

I dati dell'esercizio sui costi sono i seguenti:

Tabella	NP	NR	VAL(Regione)	VAL(Ricetta)
Ricetta	12	260	20	✗
Ingrediente	40	1200	✗	✗
Composizione	200	13000	✗	260

I dati presi sono stati acquisiti dai requisiti elencati, niente di difficile. Adesso si cerca di ottenere un valore tangibile, quindi si sostituiscono i valori:

$$\text{Costo} = 12 + \text{NR}\,(\text{RICETTA}) \div \text{VAL}\,(\text{Regione}, \text{RICETTA}) \times 200$$

La selezione della regione Veneto sulla tabella RICETTA è eseguita con la condizione:

$$\text{NR}\,(\text{RICETTA}) \div \text{VAL}\,(\text{Regione}, \text{RICETTA})$$

Andando a sostituire nuovamente i dati:

$$\text{Costo} = 12 + 260 \div 20 \times 200 = 2'612$$

Il costo del secondo Join è possibile calcolarlo allo stesso modo.

$$\text{Costo} = \text{NR}(\mathsf{COMPOSIZIONE}) \div \text{VAL}(\text{Ricetta}, \mathsf{COMPOSIZIONE}) \times 13 \times 40$$

- $\text{NR}(\mathsf{COMPOSIZIONE}) \div \text{VAL}(\text{Ricetta}, \mathsf{COMPOSIZIONE})$, rappresenta il calcolo da eseguire poiché nella tabella COMPOSIZIONE la Ricetta è una chiave esterna;
- 13, il valore ottenuto dal costo precedente, ovvero rappresenta il numero di regioni del veneto sulla tabella RICETTA;
- 40, rappresenta $\text{NP}(\mathsf{INGREDIENTE})$, come da formula.

Quindi, il costo totale del secondo Join:

$$\text{Costo} = 13'000 \div 260 \times 13 \times 40 = 26'000$$

E infine, il costo totale delle operazioni:

$$\text{Costo totale} = 2'612 + 26'000 = 28'612$$

Qua di seguito, viene riportato il costo totale nel caso in cui sia disponibile un indice B+-tree sull'attributo Codice della tabella INGREDIENTE che ha profondità 2. Il costo sul primo Join non viene influenzato poiché non riguarda la tabella INGREDIENTE e quindi rimane uguale a $2'612$. Invece, il secondo Join viene influenzato poiché al posto di utilizzare $\text{NP}(\mathsf{INGREDIENTE})$, si utilizza la profondità dell'albero, ovvero:

$$\text{Costo secondo Join} = (13'000 \div 260) \times 13 \times (2 + 1) = 650 \times 3 = 1'950$$

E quindi il costo totale:

$$\text{Costo totale} = 2'612 + 1'950 = 4'562$$

2.5.2 Esercizio 2

Si consideri il seguente schema relazionale contenete le ricette di una catena di ristoranti:

$$\begin{aligned} \mathsf{RICETTA} &\quad (\text{Nome, Descrizione, Regione})\,; \\ \mathsf{IN\_RICETTA} &\quad (\text{Ricetta, Ingrediente, Quantità})\,; \\ \mathsf{INGREDIENTE} &\quad (\text{Nome, Descrizione})\,. \end{aligned}$$

N.B.: la quantità della tabella COMPOSIZIONE è espressa in grammi.

Vincoli di integrità referenziale:

$$\begin{aligned} \mathsf{IN\_RICETTA}.\text{Ricetta} &\rightarrow \mathsf{RICETTA} \\ \mathsf{IN\_RICETTA}.\text{Ingrediente} &\rightarrow \mathsf{INGREDIENTE} \end{aligned}$$

Data la seguente interrogazione SQL:

```sql
SELECT  IR.Ingrediente, IR.Quantita, IR.Ricetta, R.Regione
FROM    IN_RICETTA IR, RICETTA R
WHERE   IR.Ricetta = R.Nome AND
        R.Regione = 'Veneto' AND
        IR.Quantita = 4
```

Calcolare il costo dell'interrogazione in termini di numero di accessi a memoria secondaria sotto le seguenti ipotesi:

- La selezione sulla tabella IN_RICETTA eseguita attraverso una scansione della tabella medesima, il risultato della selezione viene salvato in memoria secondaria in 180 pagine e riutilizzato per il join;
- La selezione delle ricette viene eseguita attraverso una scansione della tabella RICETTA, il risultato della selezione viene mantenuto nel buffer;
- L'ordine di esecuzione del join è:

  $$\mathsf{RICETTA} \bowtie \mathsf{IN\_RICETTA}$$

  E le operazioni di join vengono eseguite con la tecnica "Nested Loop Join" con una pagina di buffer disponibile per ogni tabella;
- $\text{NP}(\mathsf{INGREDIENTE}) = 30$, $\text{NP}(\mathsf{IN\_RICETTA}) = 780$, $\text{NP}(\mathsf{RICETTA}) = 150$;
- $\text{NR}(\mathsf{INGREDIENTE}) = 270$, $\text{NR}(\mathsf{IN\_RICETTA}) = 19'800$, $\text{NR}(\mathsf{RICETTA}) = 900$;
- $\text{VAL}(\text{Regione}, \mathsf{RICETTA}) = 18$, $\text{VAL}(\text{Ricetta}, \mathsf{IN\_RICETTA}) = 90$, $\text{VAL}(\text{Quantita}, \mathsf{IN\_RICETTA}) = 50$.

Come cambia il costo se è disponibile un indice B+-tree sull'attributo Ricetta della tabella IN_RICETTA che ha profondità 2.

I dati dell'esercizio sui costi sono i seguenti:

Tabella	NP	NR	VAL(Regione)	VAL(Ricetta)	VAL(Quantita)
Ricetta	150	900	18	✗	✗
In_ricetta	780	19800	✗	90	50
Ingrediente	30	270	✗	✗	✗

Come viene esplicitato dall'esercizio, la selezione sulla tabella IN_RICETTA viene eseguita:

1. Facendo una scansione della tabella medesima:

$$\mathrm{NP}(\mathsf{IN\_RICETTA}) = 780$$

2. Il risultato della selezione viene salvato in memoria secondaria in 180 pagine, quindi:

$$\mathrm{NP}(\mathsf{IN\_RICETTA}) = 780 \Longrightarrow 180$$

3. Viene utilizzato il valore per il Join. Quindi, dato che non vi sono letture di tabelle poiché il buffer contiene già i valori d'interesse, il costo è:

$$\begin{aligned} \text{Costo} &= \mathrm{NR}(\mathsf{RICETTA}) \div \mathrm{VAL}(\text{Regione}, \mathsf{RICETTA}) \times \mathrm{NP}(\mathsf{IN\_RICETTA}) \\ &= 900 \div 18 \times 180 \\ &= 50 \times 180 \\ &= 9'000 \end{aligned}$$

   Ricordando che si utilizzano 180 pagine e non 780 (punto 2).

Quindi, il costo totale è dato, rispettivamente, dalla somma delle pagine di IN_RICETTA, dalle pagine scritte nella memoria secondaria che si riferiscono a IN_RICETTA, dal Join eseguito qua sopra e infine dalla scansione della tabella RICETTA:

$$\text{Costo totale} = 780 + 180 + 9'000 + 150 = 10'110$$

Per quanto riguarda il caso in cui fosse presente un indice B+-tree, è necessario modificare il Join. Quindi diventa la formula:

$$\begin{aligned} \text{Costo Join} &= \mathrm{NR}(\mathsf{RICETTA} \text{ con selezione}) \div \mathrm{VAL}(\text{Regione}, \mathsf{RICETTA}) \times \\ &\quad (\text{Prof. Indice} + \mathrm{NR}(\mathsf{IN\_RICETTA} \text{ con selezione}) \div \mathrm{VAL}(\text{q.ta}, \mathsf{IN\_RICETTA})) \\ &= 900 \div 18 \times (19'800 \div 50) \div 90 \\ &= 220 \end{aligned}$$

Il totale quindi diventa:

$$\text{Costo totale} = 780 + 180 + 150 + 220 = 1'330$$

2.5.3 Esercizio 3

Si consideri il seguente schema relazionale:

COMUNE	(CodISTAT, Nome, Abitanti, Superficie, Prov, TipoTerritorio)
ADIACENTE	(Comune1, Comune2)
PROVINCIA	(Codice, NomeProv, SuperficieProv, Regione)

Con il dato TipoTerritorio: {montagna, mare, pianura, collina}.

La tabella ADIACENTE rappresenta la relazione di adiacenza tra comuni: poiché la relazione è simmetrica per rappresentare l'adiacenza tra i comuni A e B si memorizza sia la tupla (A,B) sia la tupla (B,A).

Vincoli d'integrità:

$$\begin{aligned} \text{ADIACENTE.Comune1} &\rightarrow \text{COMUNE} \\ \text{ADIACENTE.Comune2} &\rightarrow \text{COMUNE} \\ \text{COMUNE.Prov} &\rightarrow \text{PROVINCIA} \end{aligned}$$

Dato lo schema dell'esercizio, si consideri la seguente interrogazione:

```
SELECT  Nome, Abitanti, Superficie, Comune2
FROM    COMUNE JOIN ADIACENTE ON CodISTAT = Comune1
WHERE   TipoTerritorio <> 'pianura'
```

Indicare una stima del costo dell'interrogazione in termini di numero di accessi a memoria secondaria sapendo che: (i) la selezione dei comuni viene eseguita attraverso una scansione della tabella COMUNE e il risultato viene mantenuto nel buffer:

- L'ordine di esecuzione del join è:

  $$\text{COMUNE} \bowtie \text{ADIACENTE}$$

  E viene applicata la tecnica "Nested Loop Join" e con indice B+-tree di profondità 3 sull'attributo Comune1 della tabella ADIACENTE;

- I dati sono:

  $$\begin{aligned} \text{NP}(\text{COMUNE}) &= 15; \\ \text{NR}(\text{COMUNE}) &= 1'900; \\ \text{NP}(\text{ADIACENTE}) &= 155; \\ \text{NR}(\text{ADIACENTE}) &= 38'000; \\ \text{VAL}(\text{TipoTerritorio}, \text{COMUNE}) &= 4; \\ \text{VAL}(\text{Comune1}, \text{ADIACENTE}) &= 1'900; \end{aligned}$$

I dati messi a disposizione sono i seguenti:

Tabella	NP	NR	VAL(TipoTerritorio)	VAL(Comune1)
COMUNE	15	1'900	4	✗
ADIACENTE	155	38'000	✗	1'900
PROVINCIA	✗	✗	✗	✗

Viene richiesto il costo per:

- Eseguire una scansione della tabella COMUNE poiché viene eseguita una selezione dei comuni. Questo risultato è semplice poiché corrisponde a:

$$\text{Costo} = \text{scrittura} + \text{lettura} = \text{NP}(\text{COMUNE}) + 0 = 15$$

- Eseguire il Join con la tecnica "Nested Loop Join" senza albero B+-tree. Quindi, viene calcolato nel seguente modo il costo:

  $$\text{Costo} = 0 + \text{NR}(\text{COMUNE}) \div \text{VAL}(\text{TipoTerritorio}, \text{COMUNE}) \times \text{NP}(\text{ADIACENTE})$$

  Il valore zero indica che nel buffer non viene eseguita nessuna scrittura poiché è già stata fatta in precedenza (punto precedente). Quindi, andando a sostituire i valori:

$$\text{Costo} = 0 + 1'900 \div 4 \times 155 = 475 \times 155 = 73'625$$

- Eseguire il Join con la tecnica "Nested Loop Join" con albero B+-tree di profondità 3 sull'attributo Comune1. Quindi, viene calcolato nel seguente modo il costo:

$$\begin{aligned} \text{Costo} &= 0 + \text{NR}(\text{COMUNE}) \div \text{VAL}(\text{TipoTerritorio}, \text{COMUNE}) \times \\ &\quad (\text{Profondità Indice} + \text{NR}(\text{ADIACENTE}) \div \text{VAL}(\text{Comune1}, \text{ADIACENTE})) \\ &= 0 + 1'900 \div 4 \times (3 + 38'000 \div 1'900) \\ &= 475 \times 23 \\ &= 10'925 \end{aligned}$$

2.6 XML

2.6.1 Esercizio 1

Dato il seguente frammento XML:

```xml
<banca>
    <conto>
        <num_conto>101</num_conto>
        <agenzia>Citta</agenzia>
        <saldo>555</saldo>
        <cliente>
            <nome>Mario</nome>
            <cognome>Rossi</cognome>
            <via>Garibaldi</via>
            <citta>Verona</citta>
        </cliente>
    </conto>
    <conto>
        <num_conto>1010</num_conto>
        <agenzia>Desenzano</agenzia>
        <saldo>5550</saldo>
        <cliente>
            <nome>Mario</nome>
            <cognome>Rossi</cognome>
            <via>Garibaldi</via>
            <citta>Verona</citta>
        </cliente>
    </conto>
    <conto>
        <num_conto>301</num_conto>
        <agenzia>Negrar</agenzia>
        <saldo>1020</saldo>
        <cliente>
            <nome>Paolo</nome>
            <cognome>Verdi</cognome>
            <via>Roma</via>
            <citta>Verona</citta>
        </cliente>
    </conto>
</banca>
```

- Generare il corrisponde XSD, supponendo che ogni conto abbia uno o più clienti intestatari.
- Modificare la struttura dell'XML per ridurre la ridondanza e rappresentare una sola volta gli elementi ripetuti.

Soluzione pt.1

Qua di seguito il codice soluzione dell'esercizio supponendo che ogni conto abbia uno o più intestatari:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.banca.it"
            xmlns="http://www.banca.it">
<xsd:element name="banca">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="conto" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="conto">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="num_conto" type="xsd:unsignedInt"/>
            <xsd:element name="agenzia" type="xsd:string"/>
            <xsd:element name="saldo" type="xsd:unsignedInt"/>
            <xsd:element ref="cliente"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:complexType name="ClienteType">
    <xsd:sequence>
        <xsd:element name="nome" type="xsd:string"/>
        <xsd:element name="cognome" type="xsd:string"/>
        <xsd:element name="via" type="xsd:string"/>
        <xsd:element name="citta" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="cliente" type="ClienteType"/>
</xsd:schema>
```

Soluzione pt.2

Qua di seguito il codice soluzione dell'esercizio supponendo che ogni conto abbia uno o più intestatari:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.banca.it"
            xmlns="http://www.banca.it">
<xsd:element name="banca">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="citta" maxOccurs="unbounded"/>
            <xsd:element ref="agenzia" maxOccurs="unbounded"/>
            <xsd:element ref="cliente" maxOccurs="unbounded"/>
            <xsd:element ref="conto" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="citta">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="ct_id"
                               type="xsd:ID" use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="agenzia">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="ag_id"
                               type="xsd:ID" use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:complexType name="ClienteType">
    <xsd:sequence>
        <xsd:element name="nome" type="xsd:string"/>
        <xsd:element name="cognome" type="xsd:string"/>
        <xsd:element name="via" type="xsd:string"/>
        <xsd:element name="citta_cliente">
            <xsd:complexType>
                <xsd:attribute name="citta_id"
                               type="xsd:IDREF" use="required"/>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="cl_id" type="xsd:ID" use="required"/>
</xsd:complexType>
<xsd:element name="cliente" type="ClienteType"/>
<xsd:element name="conto">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="num_conto" type="xsd:unsignedInt"/>
            <xsd:element name="saldo" type="xsd:unsignedInt"/>
            <xsd:element name="agenzia_conto">
                <xsd:complexType>
                    <xsd:attribute name="agenzia_id"
```

```
                                    type="xsd:IDREF"
                                    use="required"/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="intestatari"
                       type="xsd:IDREFS" use="required"/>
        <xsd:attribute name="condo_id"
                       type="xsd:ID" use="required"/>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

A seguito delle modifiche di questo schema, l'istanza del documento diventa:

```
<banca>
    <citta ct_id="CT1">Verona</citta>
    <agenzia ag_id="AG1">Citta</agenzia>
    <agenzia ag_id="AG2">Desenzano</agenzia>
    <agenzia ag_id="AG3">Negrar</agenzia>

    <cliente cl_id="CL1">
        <nome>Mario</nome>
        <cognome>Rossi</cognome>
        <via>Garibaldi</via>
        <citta_cliente citta_id="CT1"/>
    </cliente>
    <cliente cl_id="CL2">
        <nome>Paolo</nome>
        <cognome>Verdi</cognome>
        <via>Roma</via>
        <citta_cliente citta_id="CT1"/>
    </cliente>

    <conto condo_id="CC23" intestatari="CL1">
        <num_conto>101</num_conto>
        <saldo>555</saldo>
        <agenzia_conto agenzia_id="AG1"/>
    </conto>
    <conto condo_id="CC39" intestatari="CL1">
        <num_conto>1010</num_conto>
        <saldo>5550</saldo>
        <agenzia_conto agenzia_id="AG2"/>
    </conto>
    <conto condo_id="CC22" intestatari="CL2">
        <num_conto>301</num_conto>
        <saldo>1020</saldo>
        <agenzia_conto agenzia_id="AG3"/>
    </conto>
</banca>
```

2.6.2 Esercizio 2

Dato il seguente documento XML, produrre una specifica XML schema alla quale tale documento sia conforme:

```
<?xml version="1.0"?>
<RubricaTelefonica  xmlns="http://www.pagineBianche.org"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.pagineBianche.org
                                        pagineBianche.xsd">
    <Utente id="UT00032">
        <tipo>privato</tipo>
        <numero>045802777</numero>
        <indirizzo>
            <via civico="23">Lombardia</via>
            <citta>Milano</citta>
        </indirizzo>
    </Utente>
    <Utente id="UT00123">
        <tipo>attivita commerciale</tipo>
        <numero>048372839</numero>
        <indirizzo>
            <piazza civico="8">Duomo</piazza>
            <citta>Brescia</citta>
        </indirizzo>
    </Utente>
    <Utente id="UT0003">
        <tipo>attivita professionale</tipo>
        <numero>04832222839</numero>
        <indirizzo>
            <piazzale civico="2">Stazione</piazzale>
            <citta>Brescia</citta>
        </indirizzo>
    </Utente>
    <!-- Etc. -->
</RubricaTelefonica>
```

I requisiti sono i seguenti. Il tipo può assumere solo uno dei seguenti valori: {privato, attivita commerciale, attivita professionale, servizio pubblico}. L'attributo id è obbligatorio. Supponiamo per semplicità che il civico sia sempre un intero positivo. Si desuma il tipo per il numero di telefono dalle istanze nel documento XML.

Soluzione

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.pagineBianche.org"
            xmlns="http://www.pagineBianche.org">
<xsd:element name="RubricaTelefonica">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="Utente" minOccurs="1"
                                      maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:simpleType name="utenteTipo">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="privato"/>
        <xsd:enumeration value="attivita commerciale"/>
        <xsd:enumeration value="attivita professionale"/>
        <xsd:enumeration value="servizio pubblico"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="numeroTipo">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{9}"/>
        <xsd:pattern value="\d{11}"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="viaTipo">
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute name="civico" type="xsd:unsignedShort"
                           use="required"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="Utente">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="tipo" type="utenteTipo"/>
            <xsd:element name="numero" type="numeroTipo"/>
            <xsd:element name="indirizzo">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:choice>
                            <xsd:element name="via"
                                         type="viaTipo"/>
                            <xsd:element name="piazza"
                                         type="viaTipo"/>
                            <xsd:element name="piazzale"
                                         type="viaTipo"/>
                        </xsd:choice>
                        <xsd:element name="citta"
                                     type="xsd:string"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:ID" use="required"/>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

3 Domande di teoria - Terzo parziale

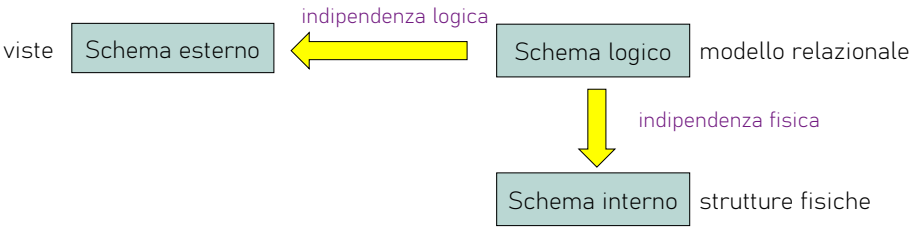
Domande di teoria tratte dalla terza prova intermedia dell'esame 06/2015.

1. ***(3 punti) Illustrare l'architettura di un DBMS descrivendo in particolare il modulo di gestione dei buffer; si indichi inoltre, per ogni modulo dell'architettura, quali sono le proprietà delle transazioni che contribuisce a garantire.***

   L'architettura di un DBMS è strutturata su 3 livelli differenti:

   - Le viste, che appartengono allo schema esterno;
   - Il modello relazionale, che appartiene allo schema logico;
   - Le strutture fisiche, che appartengono allo schema interno.

   Lo schema logico comunica con lo schema esterno attraverso l'indipendenza logica e con lo schema interno attraverso l'indipendenza fisica.

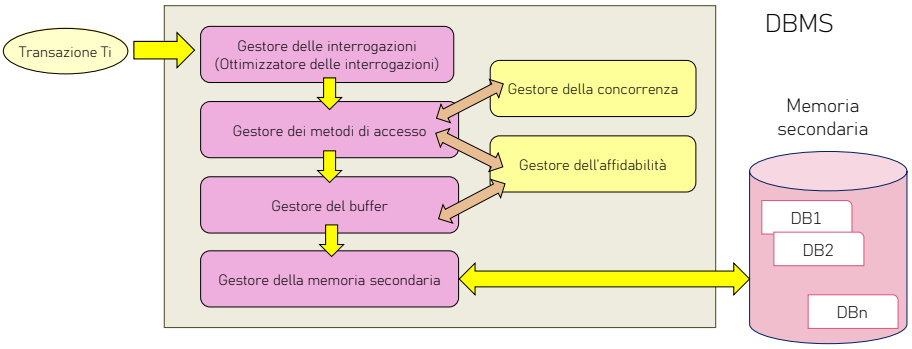


   All'interno dello schema logico ci sono molteplici moduli, i quali garantiscono anche alcune proprietà delle transazioni:

   - Gestore delle **interrogazioni**;
   - Gestore dell'**esecuzione concorrente**, il quale garantisce le proprietà di atomicità e isolamento;
   - Gestore dei **metodi d'accesso**, il quale garantisce la proprietà di consistenza;
   - Gestore dell'**affidabilità**, il quale garantisce atomicità e persistenza;
   - Gestore del **buffer**;
   - Gestore della **memoria secondaria**.

   Ricapitolando:

Proprietà	Garantita da
Atomicità	Gestore dell'**esecuzione concorrente** Gestore dell'**affidabilità**
Consistenza	Gestore dei **metodi d'accesso**
Isolamento	Gestore dell'**esecuzione concorrente**
Persistenza	Gestore dell'**affidabilità**



Il **modulo di gestione del buffer** è fondamentale nella **comunicazione tra memoria centrale e secondaria**. Il suo obbiettivo è quello di **evitare accessi multipli alla memoria di massa** così da velocizzare le operazioni.
Il buffer è organizzato in pagine di dimensione pari a un numero intero di blocchi. Si occupa di caricare (lettura)/scaricare (scrittura) pagine dalle memoria centrale alla memoria di massa:

Operazione	Descrizione
Lettura	Lettura dal buffer se il dato è presente, altrimenti lettura fisica dalla memoria di massa.
Scrittura	Scrittura in memoria di massa se e solo se è garantita la proprietà di affidabilità del sistema, quindi se vi è la certezza che l'operazione vada a buon fine.

Il modulo della gestione del buffer segue un'importante proprietà: il **principio di località dei dati**. Ovvero, **i dati referenziati di recente hanno maggior probabilità di essere referenziati nuovamente nel futuro**.
Inoltre, esiste una **legge empirica** che afferma che il 20% dei dati è tipicamente acceduto dall'80% delle applicazioni.

Infine, il gestore del buffer memorizza le seguenti **informazioni per ogni pagina**:

- File fisico
- Numero di blocco
- (variabile di stato) Contatore, indica quanti programmi utilizzano la pagina
- (variabile di stato) Bit di stato, indica se la pagina è stata modificata

Concetto	Descrizione
Architettura DBMS	L'architettura DBMS è strutturata su **3 livelli**: schema **esterno**, **logico** e **interno**. Lo schema **logico**, **il più importante**, ha al suo interno una serie di **moduli** (gestori) importanti: delle **interrogazioni**, dell'**esecuzione concorrente**, dei **metodi d'accesso**, dell'**affidabilità**, del **buffer**, della **memoria secondaria**.
Proprietà garantite	A sinistra le proprietà e a destra i moduli che le garantiscono: **Atomicità** → Gestore esecuzione concorrente, Gestore dell'affidabilità **Consistenza** → Gestore dei metodi d'accesso **Isolamento** → Gestore esecuzione concorrente **Persistenza** → Gestore dell'affidabilità
Descrizione	Modulo fondamentale nella **comunicazione tra memoria centrale e di massa**, ed è **organizzato in pagine** di dimensione pari ad un numero intero di blocchi.
Obbiettivo	**Evitare accessi multipli alla memoria di massa** così da velocizzare le operazioni.
Cosa svolge	Esegue il **caricamento** (lettura) e **scaricamento** (scrittura) **delle pagine dalla memoria centrale alla memoria di massa**.
Principio importante	I **dati referenziati di recente** hanno maggior probabilità di essere **referenziati nuovamente nel futuro**.
Legge importante	Il 20% dei dati è **acceduto** dall'80% delle applicazioni.

Tabella 1: Riepilogo dei concetti della domanda sulla gestione del buffer.

2. **(*2 punti*)** ***Si presenti in dettaglio la definizione di Conflict-Serializzabilità (CSR).***

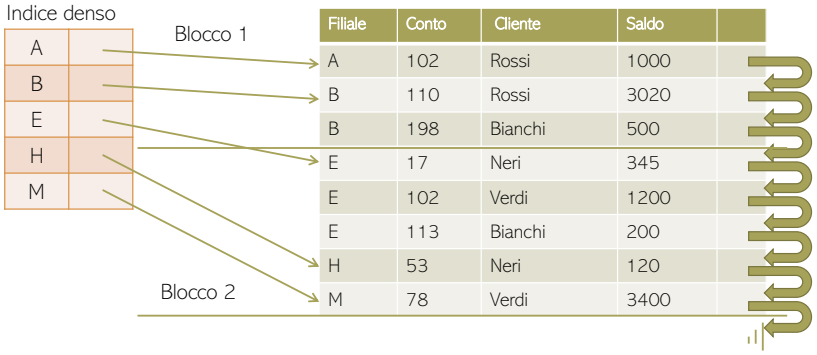
   Per dare la definizione di conflict-serializzabile, è necessario prima dare altre due definizioni. Due azioni eseguite da transazioni diverse, si dicono in **conflitto se operano sullo stesso oggetto e almeno una di esse è in scrittura**. Quindi, le possibili combinazioni sono: lettura-scrittura, scrittura-lettura e scrittura-scrittura.
   Uno schedule è definito **conflict-equivalente** ad un altro schedule se **entrambi presentano le stesse operazioni e ogni coppia di operazioni in conflitto è nello stesso ordine nei due schedule**.
   Adesso è possibile dare la definizione di conflict-serializzabile. Uno schedule è **conflict-serializzabile se esiste uno schedule seriale a esso conflict-equivalente**. L'insieme degli schedule conflict-serializzabili si chiama CSR.

Concetto	Descrizione
Definizione di conflitto	**Due azioni** di **due transazioni** diverse sono in **conflitto** se operano sullo **stesso oggetto** e almeno una di esse è **una scrittura**.
Definizione di conflict-equivalente	**Due schedule** sono **conflict-equivalenti** se: • Hanno le **stesse operazioni** • Ogni coppia di **operazioni in conflitto** è nello **stesso ordine** nei due schedule
Definizione di conflict-serializzabile	Uno schedule è conflict-serializzabile se esiste uno **schedule seriale a esso conflict-equivalente**.

Tabella 2: Riepilogo dei concetti della domanda su CSR.

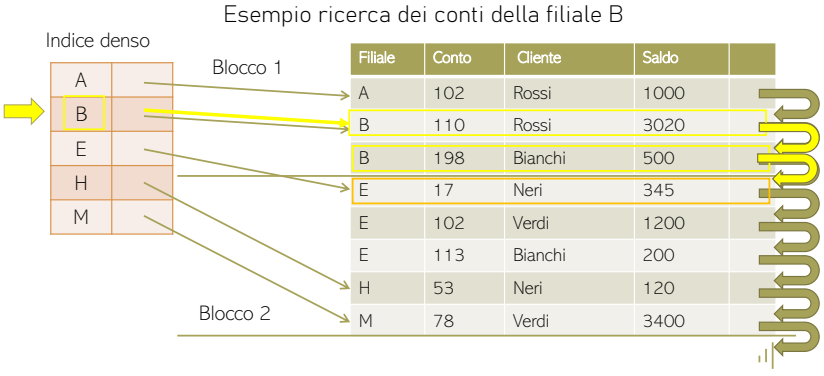
3. ***(2 punti)*** ***Lo studente illustri la struttura di accesso ai dati denominata indice primario denso: caratteristiche della struttura, ricerca, inserimento e cancellazione di entry dall'indice.***

Un indice primario utilizza una chiave di ricerca che coincide con la chiave di ordinamento del file sequenziale. Esistono due varianti dell'indice primario, una di queste è l'**indice denso**: **per ogni occorrenza della chiave presente nel file esiste un corrispondente record nell'indice**.



Operazione di **ricerca**: la **ricerca avviene tramite una scansione sequenziale dell'indice per trovare il record**. Una volta trovato, viene **effettuato l'accesso al file attraverso il puntatore**. Il costo è pari a:

$$\text{Costo} = 1 \text{ accesso indice } + 1 \text{ accesso blocco dati}$$



Operazione di **inserimento**: l'inserimento nell'indice avviene solo se la tupla inserita nel file ha un valore di chiave K (chiave di ricerca) che non è già presente.

Operazione di **cancellazione**: la cancellazione nell'indice avviene solo se la tupla cancellata nel file è l'ultima tupla con valore di chiave K (chiave di ricerca).

Concetto	Definizione
Definizione	Un **indice primario** utilizza una **chiave di ricerca che coincide con la chiave di ordinamento** del file sequenziale. Una **variante** dell'indice primario è l'indice primario denso: **per ogni occorrenza della chiave presente nel file, esiste un corrispondente record nell'indice**.
Op. ricerca	Eseguita una **scansione sequenziale dell'indice** per trovare il record ricercato. **Se viene trovato**, viene effettuato l'**accesso al file attraverso il puntatore**.
Op. inserimento	L'inserimento avviene solo se la **tupla inserita nel file ha un valore** di chiave K (**chiave di ricerca**) che **non** è già **presente**.
Op. cancellazione	La cancellazione avviene solo se la **tupla cancellata nel file è l'ultima tupla con valore di** chiave K (**chiave di ricerca**).

Tabella 3: Riepilogo dei concetti della domanda sull'indice denso.

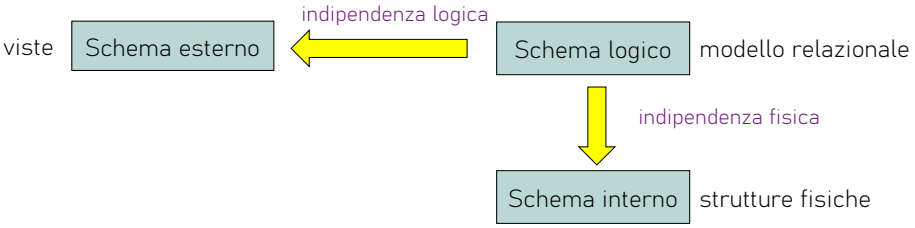
Domande di teoria tratte dalla terza prova intermedia dell'esame 07/06/2016.

1. ***(3 punti) Illustrare l'architettura di un DBMS descrivendo in particolare il modulo di gestione dei guasti (o gestore dell'affidabilità); si indichi inoltre, per ogni modulo dell'architettura, quali sono le proprietà delle transazioni che contribuisce a garantire.***

   L'architettura di un DBMS è strutturata su 3 livelli differenti:

   - Le viste, che appartengono allo schema esterno;
   - Il modello relazionale, che appartiene allo schema logico;
   - Le strutture fisiche, che appartengono allo schema interno.

   Lo schema logico comunica con lo schema esterno attraverso l'indipendenza logica e con lo schema interno attraverso l'indipendenza fisica.

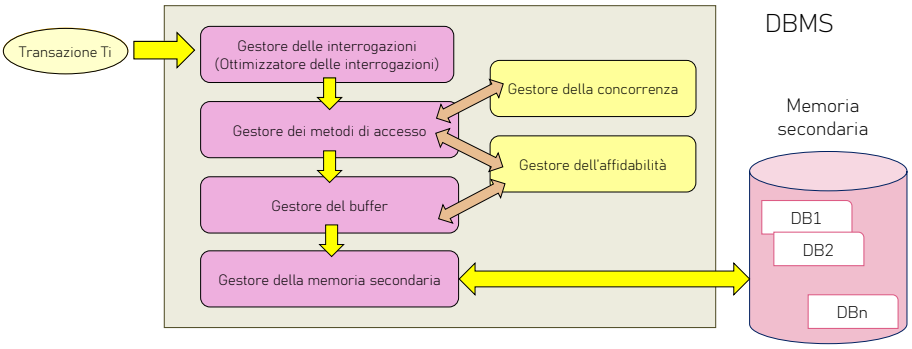


   All'interno dello schema logico ci sono molteplici moduli, i quali garantiscono anche alcune proprietà delle transazioni:

   - Gestore delle **interrogazioni**;
   - Gestore dell'**esecuzione concorrente**, il quale garantisce le proprietà di atomicità e isolamento;
   - Gestore dei **metodi d'accesso**, il quale garantisce la proprietà di consistenza;
   - Gestore dell'**affidabilità**, il quale garantisce atomicità e persistenza;
   - Gestore del **buffer**;
   - Gestore della **memoria secondaria**.

   Ricapitolando:

Proprietà	Garantita da
Atomicità	Gestore dell'**esecuzione concorrente** Gestore dell'**affidabilità**
Consistenza	Gestore dei **metodi d'accesso**
Isolamento	Gestore dell'**esecuzione concorrente**
Persistenza	Gestore dell'**affidabilità**



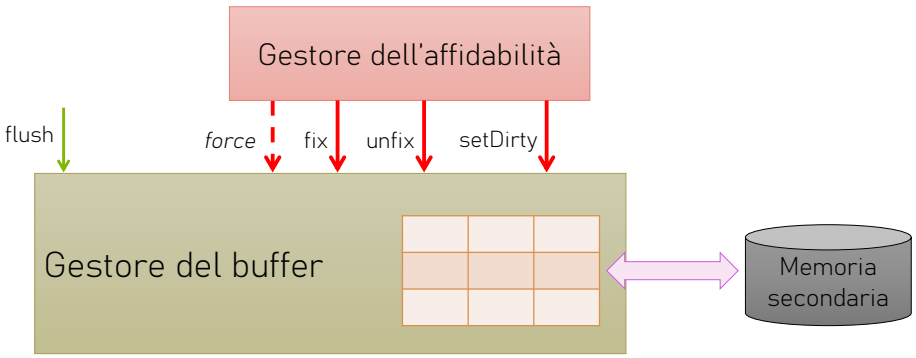
Il **modulo di gestione dei guasti**, o **gestore dell'affidabilità**, garantisce due **proprietà fondamentali**:

- **Atomicità**, garantire che le **transazioni non vengano lasciate incomplete**;
- **Persistenza**, garantire che gli **effetti di ciascuna transazione conclusa con un *commit* siano mantenuti in modo permanente**.

Le **proprietà** vengono **garantite grazie ai log**, ovvero un **archivio persistente su cui vengono registrate le varie azioni svolte dal DBMS**.

Il gestore dell'affidabilità utilizza delle primitive (force, fix, unfix, setDirty) per comunicare con il gestore del buffer. Questo perché esso invia **richieste di accessi a pagine in lettura/scrittura al *buffer manager***, e genera **ulteriori richieste di lettura/scrittura necessarie a garantire la robustezza e resistenza ai guasti**.

Infine, date le proprietà che deve garantire, questo **componente necessita di una memoria stabile, ovvero di una memoria resistente ai guasti**.



Concetto	Definizione
Architettura DBMS	L'architettura DBMS è strutturata su **3 livelli**: schema **esterno**, **logico** e **interno**. Lo schema **logico**, **il più importante**, ha al suo interno una serie di **moduli** (gestori) importanti: delle **interrogazioni**, dell'**esecuzione concorrente**, dei **metodi d'accesso**, dell'**affidabilità**, del **buffer**, della **memoria secondaria**.
Proprietà garantite	A sinistra le proprietà e a destra i moduli che le garantiscono: **Atomicità** → Gestore esecuzione concorrente, Gestore dell'affidabilità **Consistenza** → Gestore dei metodi d'accesso **Isolamento** → Gestore esecuzione concorrente **Persistenza** → Gestore dell'affidabilità
Approfondimento proprietà	L'**atomicità** garantisce che le transazioni non vengano lasciate incomplete. Invece, la **persistenza** garantisce che gli effetti di ciascuna transazione conclusa con un commit siano mantenuti in modo permanente. Queste **proprietà** sono **garantite dal modulo di gestione dei guasti** grazie al log, ovvero un **archivio persistente** su cui vengono **registrate** le varie **azioni svolte dal DBMS**.
Compito eseguito	Il modulo di gestione dei guasti invia **richieste di accessi a pagine in lettura/scrittura al buffer**. Inoltre, genera **ulteriori richieste di lettura/-scrittura necessarie a garantire la robustezza e resistenza ai guasti**.

Tabella 4: Riepilogo dei concetti della domanda sull'indice denso.

2. ***(3 punti)*** ***Si presenti in dettaglio la politica di concessione dei lock applicata dal gestore dell'esecuzione concorrente secondo la tecnica detta "Locking a due fasi".***

   Il ***locking*** è una tecnica in cui **tutte le operazioni di lettura/scrittura devono essere protette tramite l'esecuzione di opportune primitive** (r_lock, w_lock, unlock). Quindi, lo **scheduler riceve una sequenza di richieste di esecuzione di queste primitive da parte delle transazioni, e ne determina la correttezza con una semplice ispezione di una struttura dati**.

   Il **gestore della concorrenza mantiene per ogni risorsa**: il suo **stato** (identificato tramite le tre primitive), le transazioni in r_lock, ovvero tutte le transazioni che hanno un r_lock sulla risorsa x. Inoltre, viene salvata una **tabella dei conflitti** in cui:

   - Le **righe** identificano le **richieste**;
   - Le **colonne** identificano lo **stato della risorsa richiesta**;
   - In ogni **cella**:
     - Valore di **sinistra** identifica l'**esito della richiesta**;
     - Valore di **destra** identifica lo **stato** che verrà assunto dalla **risorsa dopo l'esecuzione della primitiva**.

   Grazie a questa tabella, il lock manager ha la possibilità di scegliere quando concedere o non concedere una risorsa in lettura/scrittura:

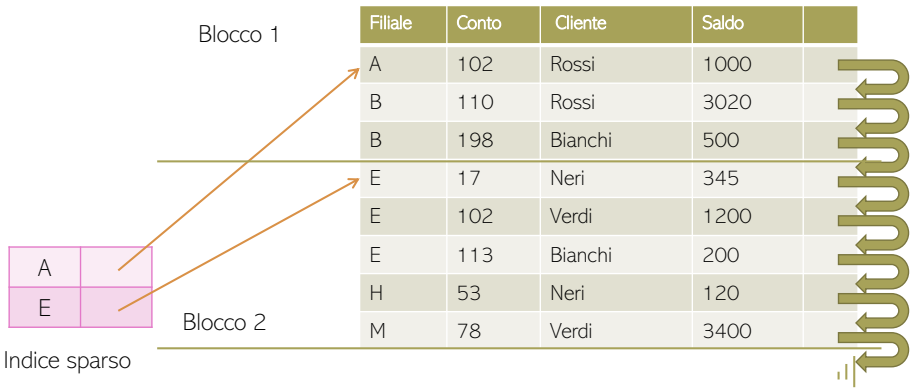
   - **Oggetto** bloccato in **lettura**, allora è possibile **concedere un'altra richiesta di lock in lettura** (*lock condiviso*);
   - Nel caso di unlock di una **risorsa bloccata** in modo condiviso (lettura), la **risorsa ritorna libera quando non ci sono altre transazioni in lettura che operano su di essa**.

Concetto	Definizione
Definizione locking	Il ***locking*** è una tecnica in cui **tutte le operazioni di lettura/scrittura devono essere protette tramite l'esecuzione di opportune primitive** (r_lock, w_lock, unlock).
Politica di concessione	Il **gestore della concorrenza mantiene per ogni risorsa**: il suo **stato** (identificato tramite le tre primitive), le transazioni in r_lock, ovvero **tutte le transazioni che hanno un r_lock sulla risorsa x**.
Tabella dei conflitti	Salvata una **tabella dei conflitti** in cui: • Le **righe** identificano le **richieste**; • Le **colonne** identificano lo **stato della risorsa richiesta**; • In ogni **cella**: – Valore di **sinistra** identifica l'**esito della richiesta**; – Valore di **destra** identifica lo **stato** che verrà assunto dalla **risorsa dopo l'esecuzione della primitiva**.
Concessione risorse	È possibile **scegliere quando concedere o non concedere una risorsa** in lettura/scrittura: • **Oggetto** bloccato in **lettura**, allora è possibile **concedere un'altra richiesta di lock in lettura** (*lock condiviso*); • Nel caso di unlock di una **risorsa bloccata** in modo condiviso (lettura), la **risorsa ritorna libera quando non ci sono altre transazioni in lettura che operano su di essa**.

Tabella 5: Riepilogo dei concetti della domanda sulla politica di concessione dei lock.

3. ***(2 punti)*** ***Lo studente illustri la struttura di accesso ai dati denominata indice primario sparso, 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.***

   Un indice primario utilizza una chiave di ricerca che coincide con la chiave di ordinamento del file sequenziale. Esistono due varianti dell'indice primario, una di queste è l'**indice sparso**: **solo per alcune occorrenze della chiave presenti nel file esiste un corrispondente record nell'indice, tipicamente una per blocco**.



   Operazione di ricerca con una chiave K:

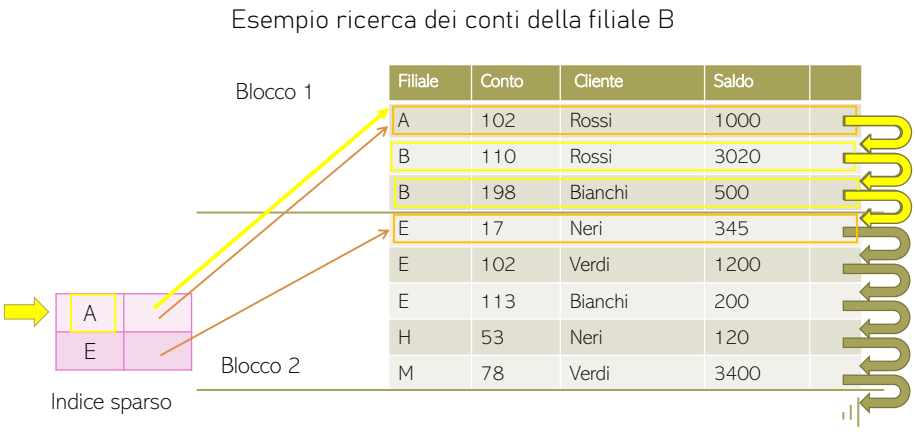
   (a) Eseguita una **scansione sequenziale dell'indice**. La scansione termina quando viene trovato un **record che ha come chiave il valore più grande**, ma che è comunque **minore o uguale alla chiave di ricerca** K;

   (b) Eseguito l'**accesso al file attraverso il puntatore all'interno del record**, ed eseguita la **scansione del file** (blocco corrente) **per trovare le tuple con chiave** K.

   N.B.: si cerca il valore di chiave più grande $\geq$ K, poiché data la struttura dell'indice sparso, la chiave K potrebbe non essere presente nell'indice.

$$\text{Costo totale} = 1 \text{ accesso indice } + 1 \text{ accesso blocco dati}$$

s



Concetto	Definizione
Definizione	Un **indice primario** utilizza una **chiave di ricerca che coincide con la chiave di ordinamento** del file sequenziale. Una **variante** dell'indice primario è l'indice primario sparso: **solo per alcune occorrenze delle chiavi presenti nel file esiste un corrispondente record nell'indice, tipicamente una per blocco**.
Op. ricerca	La ricerca si divide in due fasi: (a) Esecuzione di una **scansione sequenziale dell'indice**. La scansione termina quando viene trovato un **record che ha come chiave il valore più grande**, ma che è comunque **minore o uguale alla chiave di ricerca** K; (b) **Accesso al file attraverso il puntatore all'interno del record**, e **scansione del file** (blocco corrente) **per trovare le tuple con chiave** K.

Tabella 6: Riepilogo dei concetti della domanda sull'indice sparso.

4. ***(2 punti)*** ***Lo studente illustri le differenze tra i costrutti elemento e attributo del linguaggio XML, mostrando un esempio di documento XML dove vengono utilizzati entrambi.***

   Un **elemento** è tutto ciò che viene compreso tra il tag di inizio e il tag di fine e la keyword element li identifica in uno schema XML. Esistono varie modalità per dichiarare un elemento e inoltre possono essere eseguite dichiarazioni annidate.

   Un **attributo** è un valore opzionale che aggiunge informazioni ad un elemento. Solitamente viene utilizzato per rappresentare dei valori unici come gli ID.

   Date le due definizioni, le **maggiori differenze** sono:

   - La **dichiarazione di attributi deve avvenire sempre dopo la dichiarazione di elementi** (non viceversa).
   - Un **attributo** può contenere **solo un valore**, mentre gli **elementi possono avere figli**.

   Un esempio di codice XML:

```xml
<xsd:element name="Book">
  <xsd:complexType>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string">
      <xsd:complexType>
        <xsd:attribute  name="author_id" type="xsd:ID" use="
    required"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:complexType>
</xsd:element>
```

Concetto	Definizione
Breve def. elemento	Un **elemento** è tutto ciò che viene compreso tra il tag di inizio e il tag di fine. Viene identificato dalla keyword element nel XML schema.
Breve def. attributo	Un **attributo** è un valore opzionale che aggiunge informazioni ad un elemento. Utilizzato per rappresentare gli ID di solito.
Differenze	Due differenze sostanziali: (1) la **dichiarazione di attributi deve avvenire sempre dopo la dichiarazione di elementi** (non viceversa); (2) un **attributo** può contenere **solo un valore**, mentre gli **elementi possono avere figli**.
Esempio	Si veda il codice sopra.

Tabella 7: Riepilogo dei concetti della domanda sull'elemento e l'attributo in XML.

Domande di teoria tratte dalla terza prova intermedia dell'esame 21/04/2022.

1. ***(3 punti) Illustrare l'architettura di un DBMS descrivendo in particolare il modulo di gestione dei buffer; si indichi inoltre quali moduli garantiscono le proprietà di persistenza e consistenza delle transazioni.***

   Risposta a pagina 41.

2. ***(2 punti) Si presenti in dettaglio la definizione di conflict-equivalenza tra due schedule.***

   Prima della definizione di conflict-equivalenza, è necessario dire cosa è un conflitto.

   Date due azioni eseguite da transazioni diverse, si dice che un'azione è in **conflitto** con un'altra, se **operano sullo stesso oggetto** e almeno **una di esse è in scrittura**. I conflitti che si possono creare quindi sono: lettura-scrittura, scrittura-lettura, scrittura-scrittura.

   Uno schedule è definito **conflict-equivalente** ad un altro schedule, se **entrambi presentano le stesse operazioni** e ogni **coppia di operazioni in conflitto è nello stesso ordine** in entrambi gli schedule.

Concetto	Descrizione
Def. conflitto	**Due azioni** di **due transazioni** diverse sono in **conflitto** se operano sullo **stesso oggetto** e almeno una di esse è **una scrittura**.
Def. conflict-equivalenza	**Due schedule** sono **conflict-equivalenti** se: • Hanno le **stesse operazioni** • Ogni coppia di **operazioni in conflitto** è nello **stesso ordine** nei due schedule

Tabella 8: Riepilogo dei concetti della domanda sulla conflict-equivalenza.

3. ***(2 punti)*** ***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.***

   Le **strutture ad accesso calcolato** (hashing), si basano su una **funzione di hash che esegue un *mapping*** dei **valori chiave di ricerca** sugli **indirizzi di memorizzazione delle tuple** nelle pagine della memoria secondaria.

   Un **utilizzo efficiente** della funzione di hash prevede la **distribuzione dei valori delle chiavi** in maniera uniforme, casuale e all'interno dei *bucket*.

   Operazione di **ricerca**:

   (a) (nessun costo) **Calcolo la funzione di hash**

   (b) (1 per ogni pagina) **Accedo al bucket** ottenuto dalla funzione hash del punto precedente

   (c) (costo uguale agli accessi per pagina) **Accedo alle tuple attraverso i puntatori del bucket**

Concetto	Descrizione
Definizione	Le **strutture di accesso calcolato** si basano su una **funzione di hash** che ha l'obbiettivo di **eseguire un mapping** tra: Chiavi di ricerca $\longrightarrow$ Indirizzi di memorizzazione delle tuple
Struttura	La **distribuzione delle chiavi** deve essere eseguita in modo: • **Uniforme** • **Casuale** • All'**interno dei *bucket*** In questo modo vi è un utilizzo efficiente.
Op. ricerca	L'operazione di ricerca si divide in 3 punti: (a) **Calcolo della funzione hash** (b) **Accesso al *bucket*** con il risultato del punto 1 (c) **Accesso alle tuple** attraverso i puntatori del *bucket*

Tabella 9: Riepilogo dei concetti della domanda sulla struttura ad accesso calcolato (hashing).

4. ***(2 punti) Lo studente illustri l'algoritmo di ripresa a caldo.***

   L'**algoritmo** è il seguente:

   (a) Eseguito l'accesso all'ultimo log, ovvero quello al momento del guasto;

   (b) Si ripercorre l'intero log all'indietro finché non viene trovato l'ultimo checkpoint;

   (c) Creazione di due insiemi:
   - UNDO (rifare) inizializzato con le transazioni attive al checkpoint
   - REDO (disfare) inizializzato con l'insieme vuoto

   (d) Si ripercorre il file log in avanti, aggiungendo a:
   - UNDO le transazioni di cui è presente un record di begin
   - REDO gli identificativi delle transazioni di cui è presente il record di commit

   Giunti alla fine, UNDO avrà gli identificativi delle transazioni da disfare e REDO da rifare.

   (e) Si ripercorre, di nuovo, il log all'indietro disfacendo le azioni presenti nell'insieme UNDO. La salita termina quando viene trovata la prima azione della transazione più "vecchia" nei due insiemi (UNDO e REDO).

   (f) Si ripercorre, di nuovo, il log in avanti rifacendo le azioni presenti nell'insieme REDO.

Domande di teoria tratte dalla terza prova intermedia dell'esame 22/04/2022.

1. ***(3 punti)* Illustrare le proprietà delle transazioni; si indichi inoltre quali moduli di un DBMS garantiscono ciascuna di tali proprietà.**

   L'**atomicità** afferma che una **transazione è un'unità indivisibile di esecuzione**. Quindi, devono essere resi visibili tutti gli effetti di una transazione (tutto), altrimenti la transazione non deve avere nessun effetto sulla base di dati (o niente). Le implicazioni nel caso in cui la proprietà venisse applicata:

   - **Transazione interrotta prima del commit**, il sistema deve essere in grado di **ricostruire la situazione esistente prima dell'esecuzione della transazione eliminando il lavoro eseguito fino a quel momento**.
   - **Transazione interrotta dopo il commit**, il sistema deve essere in grado di **assicurare che la transazione lasci la base di dati nel suo stato finale**.

   L'atomicità è garantita dal gestore dell'esecuzione concorrente e dal gestore dell'affidabilità.

   La **consistenza** afferma che l'**esecuzione della transazione non deve violare i vincoli di integrità della base di dati**. In **caso di violazioni**, il sistema **annulla o corregge la transazione**. Il controllo può essere di due tipi:

   - **Controllo della violazione immediata**. I controlli vengono eseguiti durante l'esecuzione della transazione.
   - **Controllo della violazione differita**. I controlli vengono eseguiti al termine dell'esecuzione della transazione, cioè dopo il commit.

   La seconda causa un abort dell'intera transazione, mentre la prima no e risulta essere più efficiente. La consistenza è garantita dal gestore dei metodi d'accesso.

   L'**isolamento** afferma che l'**esecuzione di una transazione deve essere indipendente dalla contemporanea esecuzione di altre transazioni**. Le implicazioni in questo caso sono:

   - **Esecuzione concorrente** di un insieme di transazioni deve essere analogo al risultato che le stesse transazioni otterrebbero qualora ciascuna di esse fosse eseguita da sola.
   - **Esecuzione indipendente** evita che un eventuale rollback provochi un effetto dominio generando un rollback anche nelle altre.

   L'isolamento è garantita dal gestore dell'esecuzione concorrente.

   La **persistenza** afferma che l'**esecuzione di una transazione che ha eseguito il commit correttamente non venga più perso**. Quindi, questa proprietà garantisce gli effetti delle transazioni. La persistenza è garantita dal gestore dell'affidabilità.

Concetto	Descrizione
Atomicità	Una transazione è un'**unità indivisibile di esecuzione**. Quindi o vengono resi visibili tutti gli effetti di una transazione oppure non viene fatto niente. Le implicazioni sono: • **Transazione interrotta prima del commit**, il sistema deve essere in grado di ricostruire la situazione iniziale pre-transazione eliminando quanto fatto fino a quel momento • **Transazione interrotta dopo il commit**, il sistema deve assicurare che la transazione alsci la base di dati nel suo stato finale Atomicità garantita dal **gestore**: dell'**esecuzione concorrente** e dell'**affidabilità**.
Consistenza	L'**esecuzione della transazione non deve violare i vincoli di integrità della base di dati**. Nel caso di una violazione, il sistema **annulla/corregge** la transazione. Due tipi di controllo: • **Controllo della violazione immediata**. Controlli eseguiti **durante l'esecuzione** della transazione • **Controllo della violazione differita**. Controlli eseguiti **al termine dell'esecuzione** della transazione (post-commit). Consistenza garantita dal **gestore dei metodi d'accesso**.
Isolamento	L'**esecuzione di una transazione deve essere indipendente dalla contemporanea esecuzione di altre transazioni**. Le implicazioni sono: • **Esecuzione concorrente** di un insieme di transazioni deve essere analogo al risultato prodotto nel caso in cui le transazioni fossero eseguite indipendentemente. • **Esecuzione indipendente** evita che eventuali rollback provochino effetti domino. Isolamento garantita dal **gestore dell'esecuzione concorrente**.
Persistenza	L'**esecuzione di una transazione che ha eseguito il commit correttamente non deve essere più perso**. Persistenza garantita dal **gestore dell'affidabilità**.

Tabella 10: Riepilogo dei concetti della domanda sulle proprietà delle transazioni.

2. ***(2 punti) Si presenti in dettaglio la definizione di view-equivalenza tra due schedule.***

   Prima di dare la definizione di view-equivalenza, si forniscono altre due definizioni necessarie.

   Un'operazione di **lettura** $r_i(x)$ viene definita "**legge da**" quando una **scrittura** $w_j(x)$, **precede tale lettura** $r_i(x)$. Inoltre, non ci deve essere **nessun'altra scrittura sulla stessa risorsa da parte di un'altra transazione diversa** $w_k(x)$, tra le due operazioni citate.

   Un'operazione di **scrittura** $w_i(x)$ viene definita "**scrittura finale**" se è l'**ultima scrittura dell'oggetto** x **che appare nello schedule**.

   Due schedule vengono detti **view-equivalenti** se possiedono la **stessa relazione "legge da"** e le **stesse "scritture finali"**.

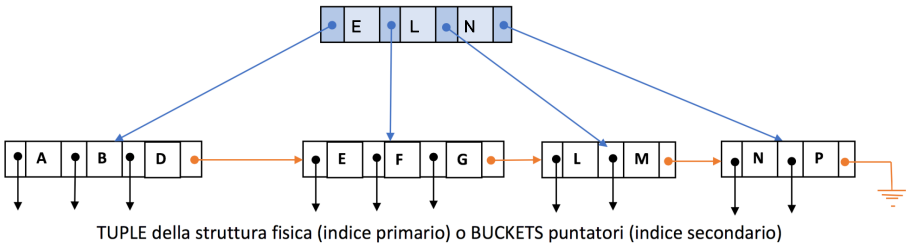
Concetto	Descrizione
Def. legge da	Quando, data una risorsa, una **scrittura di un'altra transizione precede una lettura di un'altra transazione sulla stessa risorsa**. Inoltre, **non ci deve essere una scrittura da parte di un'altra transizione**, su tale risorsa, tra le due operazioni citate.
Def. scrittura finale	Quando è l'**ultima scrittura dell'oggetto** x **che appare nello schedule**.
Def. view-equivalenza	Se **due schedule hanno la stessa relazione legge da e le stesse scritture finali**.

Tabella 11: Riepilogo dei concetti della domanda sulla view-equivalenza.

3. ***(2 punti)*** ***Lo studente illustri la struttura di accesso ai dati denominata B+-tree, 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.***

   Un albero B+-tree è una struttura ad albero formata da una serie di **nodi** che **identificano le pagine della memoria secondaria**. I nodi sono **collegati** tramite **legami**, i quali rappresentano i **puntatori ad ogni pagina**. Un nodo può avere molte foglie, cioè figli, e per questo motivo gli albero spesso hanno molti nodi ma pochi livelli.

   Un albero è **bilanciato** se la lunghezza dei percorsi che collegano la radice ai nodi foglia è costante.



Esempio di albero B+-tree.

Operazione di **ricerca**:

(a) Si **cerca** nel nodo radice il **più piccolo valore di chiave maggiore del valore di chiave di ricerca** K:
   - Se esiste, allora si segue il puntatore relativo;
   - Se non esiste, viene seguito l'ultimo puntatore presente nel nodo.

(b) Una volta **raggiunto il nodo foglia**, viene **cercato** il valore K (**chiave di ricerca**) nel nodo:
   - Se viene trovato il valore, si prosegue la strada verso le tuple;
   - Se non esiste, si ritorna al passo 1.

Concetto	Definizione
Struttura	Un albero B+-tree è una struttura ad albero composta da nodi e legami: • **Nodi**, identificano le **pagine della memoria secondaria**; • **Legami**, collegano i nodi e rappresentano i **puntatori ad ogni pagina**.
Op. ricerca	L'operazione di ricerca si divide in **due fasi**: (a) **Cercare** nel nodo radice il **più piccolo valore di chiave maggiore del valore di chiave di ricerca**. Se trovato, si **segue** il relativo **puntatore**, altrimenti viene **seguito l'ultimo puntatore presente nel nodo**; (b) Raggiunto il **nodo foglia**, viene **cercato il valore della chiave di ricerca**. Se trovato, si segue la **strada verso la tupla**, altrimenti si **torna al punto 1**.

Tabella 12: Riepilogo dei concetti della domanda sugli alberi B+-tree.

4. ***(2 punti) Illustrare il meccanismo di 2PL stretto.***

   Il locking a due fasi stretto (2PL stretto) è un principio che afferma: **i lock di una transazione possono essere rilasciati solo dopo aver correttamente effettuato le operazioni di commit/abort**.

   La conseguenza di questo metodo è che i lock vengono rilasciati solo al termine della transazione, ovvero una volta raggiunto lo stato finale.

   Il **vantaggio principale** riguarda l'**impossibilità** di verificarsi del fenomeno di **letture sporche**. Questo **perché viene impedito l'accesso, da parte di altre transazioni, ai dati scritti da transazioni che ancora non hanno effettuato il commit**.

Concetto	Descrizione
Definizione	Il 2PL stretto afferma che i **lock di una transazione possono essere rilasciati solo dopo aver correttamente effettuato le operazioni di commit/abort**.
Conseguenza	I lock vengono rilasciati solo al termine della transazione, ovvero al raggiungimento dello stato finale.
Vantaggio	Il fenomeno di **"lettura sporca" non può verificarsi poiché viene impedito l'accesso, da parte di altre transazioni, ai dati scritti da transazioni che ancora non hanno effettuato il commit**.

Tabella 13: Riepilogo dei concetti della domanda sul 2PL stretto.

Domande di teoria tratte dalla terza prova intermedia dell'esame 10/06/2022.

1. ***(2 punti) Illustrare le proprietà delle transazioni ed indicare da quali moduli del DBMS vengono garantite.***

   Risposta a pagina 58.

2. ***(3 punti) Si presenti la definizione di View-serializzabilità e la relazione tra l'insieme degli schedule VSR e l'insieme degli schedule CSR. Presentare la dimostrazione formale di tale relazione.***

   Prima di dare la dimostrazione di view-serializzabile, è necessario dare tre definizioni:

   (a) Un'operazione di lettura $r_i(x)$ si definisce "**legge da**" quando una scrittura $w_j(x)$ di un'altra transazione sulla stessa risorsa precede la lettura $r_i(x)$ sulla stessa risorsa. Inoltre, non ci deve essere alcuna scrittura $w_k(x)$ appartenente ad un'altra transazione tra le due operazioni ($r_i(x)$ e $w_j(x)$).

   (b) Un'operazione di scrittura $w_i(x)$ viene definita "**scrittura finale**" se è l'ultima scrittura dell'oggetto x che appare nello schedule.

   (c) **Due schedule** vengono definiti **view-equivalenti** se posseggono la **stessa relazione "legge da" e le stesse "scritture finali"**.

   Uno schedule viene detto view-serializzabile se esiste uno schedule seriale view-equivalente ad esso. L'insieme degli schedule seriali view-equivalenti ad esso si chiama VSR.

   Inoltre, esiste una **relazione tra l'insieme VSR e l'insieme dei conflict-serializzabili (CSR)**:

   $$CSR \subset VSR$$

   Ovvero, se uno **schedule è CSR, allora è anche VSR allo stesso tempo, ma non viceversa!**

Dimostrazione $CSR \Longrightarrow VSR$. Si supponga che lo schedule S_1 abbia uno schedule seriale conflict-equivalente a S_2. Allora si dimostra che lo schedule S_1 ha come schedule seriale view-equivalente S_2.

Dato che S_2 è uno schedule seriale conflict-equivalente a S_1, allora:

- L'insieme delle scritture finali di S_1 è uguale all'insieme delle scritture finali di S_2:

  $$\text{Scritture finali } S_1 = \text{Scritture finali } S_2$$

  Ipotizzando per assurdo il contrario, allora esisterebbero due scritture sulla stessa risorsa in ordine diverso. Tuttavia, dato che le due scritture sarebbero in conflitto, i due schedule non sarebbero conflict-equivalenti.

- L'insieme dei "legge da" di S_1 è uguale all'insieme dei "legge da" di S_2:

  $$\text{Legge da } S_1 = \text{Legge da } S_2$$

  Ipotizzando per assurdo il contrario, allora esisterebbero due scritture in ordine diverso o coppie lettura-scrittura in ordine diverso. Tuttavia, come sopra, dato che le due scritture sarebbero in conflitto, i due schedule non sarebbero conflict-equivalenti.

QED

Concetto	Definizione
Def. legge da	Quando, data una risorsa, una **scrittura di un'altra transizione precede una lettura di un'altra transazione sulla stessa risorsa**. Inoltre, **non ci deve essere una scrittura da parte di un'altra transizione**, su tale risorsa, tra le due operazioni citate.
Def. scrittura finale	Quando è l'**ultima scrittura dell'oggetto** x **che appare nello schedule**.
Def. view-equivalenza	Se **due schedule hanno la stessa relazione legge da e le stesse scritture finali**.
Dimostrazione $CSR \Rightarrow VSR$	Vedere sopra.

Tabella 14: Riepilogo dei concetti della domanda sulla view-serializzabilità e la relazione tra VSR e CSR.

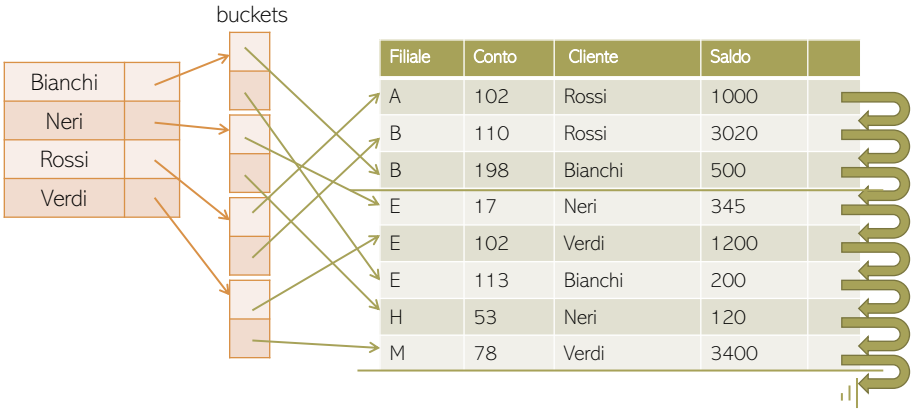
3. ***(2 punti)*** ***Lo studente illustri la struttura di accesso ai dati denominata indice secondario, si descrivano in particolare i seguenti punti: (i) le caratteristiche della struttura di accesso, (ii) l'algoritmo di ricerca di una tupla con chiave K usando l'indice.***

   L'**indice secondario** utilizza una **chiave di ricerca** che **non coincide con la chiave di ordinamento del file sequenziale**.

   Ogni record possiede:

   - Un **valore** della **chiave di ricerca**
   - **Puntatore al bucket di puntatori che individuano nel file sequenziale tutte le tuple con valore di chiave uguale al valore della chiave di ricerca**

   Inoltre, gli indici secondari **sono sempre densi**.



   Operazione di **ricerca**:

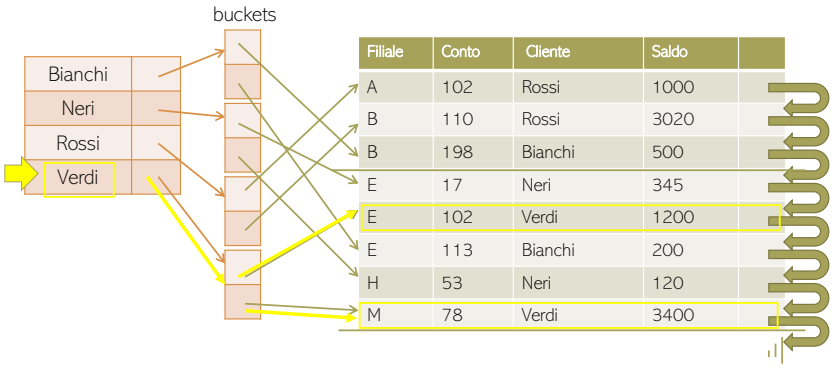
   (a) **Scansione sequenziale dell'indice** per trovare il record

   (b) **Accesso al bucket di puntatori** attraverso il puntatore presente dentro il record trovato

   (c) **Accesso al file attraverso i puntatori del bucket** al quale è stato fatto l'accesso al punto precedente.

   Costo = 1 accesso indice $+ 1$ accesso al bucket $+ n$ accessi pagine dati

Esempio di ricerca dei conti di Verdi



Concetto	Descrizione
Definizione	Indice secondario usa una **chiave di ricerca che non coincide con la chiave di ordinamento del file sequenziale**.
Valori nel record	Ogni record ha due valori: (a) **Valore chiave di ricerca** (b) **Puntatore al bucket di puntatori**, i quali individuano nel file sequenziale tutte le tuple con valore di chiave uguale al valore della chiave di ricerca.
Op. ricerca	La ricerca si divide in tre passi: (a) **Scansione sequenziale dell'indice** finché non viene trovato il record (b) **Accesso al bucket dei puntatori** grazie al puntatore trovato nel record (c) **Accesso al file sequenziale** grazie ai puntatori del bucket

4. ***(2 punti) Lo studente illustri l'algoritmo di ripresa a caldo.***

Risposta a pagina 57.

4 Indice delle domande

4.1 Terzo parziale

4.1.1 Domande teoriche

- Illustrare l'architettura di un DBMS descrivendo in particolare il modulo di **gestione dei buffer**; si indichi inoltre, per ogni modulo dell'architettura, quali sono le proprietà delle transazioni che contribuisce a garantire.

  Risposta: pagina 41

- Illustrare l'architettura di un DBMS descrivendo in particolare il modulo di **gestione dei guasti** (o gestore dell'affidabilità); si indichi inoltre, per ogni modulo dell'architettura, quali sono le proprietà delle transazioni che contribuisce a garantire.

  Risposta: pagina 47

- Illustrare le **proprietà delle transazioni**; si indichi inoltre quali moduli di un DBMS garantiscono ciascuna di tali proprietà.

  Risposta: pagina 58

- Illustrare il meccanismo di **2PL stretto**.

  Risposta: pagina 63

- Si presenti in dettaglio la definizione di **Conflict-Serializzabilità (CSR)**.

  Risposta: pagina 44

- Si presenti in dettaglio la **politica di concessione dei lock** applicata dal gestore dell'esecuzione concorrente secondo la tecnica detta "Locking a due fasi".

  Risposta: pagina 50

- Si presenti in dettaglio la **definizione di conflict-equivalenza** tra due schedule.

  Risposta: pagina 55

- Si presenti in dettaglio la **definizione di view-equivalenza** tra due schedule.

  Risposta: pagina 60

- Si presenti la definizione di **View-Serializzabilità (VSR)** e la **relazione** tra l'insieme degli schedule **VSR** e l'insieme degli schedule **CSR**. Presentare la dimostrazione formale di tale relazione.

  Risposta: pagina 64

- Lo studente illustri la struttura di accesso ai dati denominata **indice primario denso**: caratteristiche della struttura, ricerca, inserimento e cancellazione di entry dall'indice.

  Risposta: pagina 45

- Lo studente illustri la struttura di accesso ai dati denominata **indice primario sparso**, 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.

  Risposta: pagina 52

- 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.

  Risposta: pagina 56

- Lo studente illustri la struttura di accesso ai dati denominata **B+-tree**, 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.

  Risposta: pagina 61

- Lo studente illustri la struttura di accesso ai dati denominata **indice secondario**, 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.

  Risposta: pagina 66

- Lo studente illustri le differenze tra i costrutti **elemento** e **attributo** del linguaggio XML, mostrando un esempio di documento XML dove vengono utilizzati entrambi.

  Risposta: pagina 54

- Lo studente illustri l'algoritmo di **ripresa a caldo**.

  Risposta: pagina 57