



Tecnologia di un database server

Università degli Studi del Sannio
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Corso di Basi di Dati
Anno Accademico 2005/2006
docente: ing. Corrado Aaron Visaggio

email: visaggio@unisannio.it

ricevimento: mercoledì 11.00-13.00.

Corrado Aaron Visaggio

1



Avvisi

- Seminario
- Recuperi
- Sourceforge.net
- <http://www.javaworld.com/>
- <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/> - "The Cathedral and the Bazaar" by Eric Steven Raymond
- www.sei.cmu.edu – Software Engineering Institute

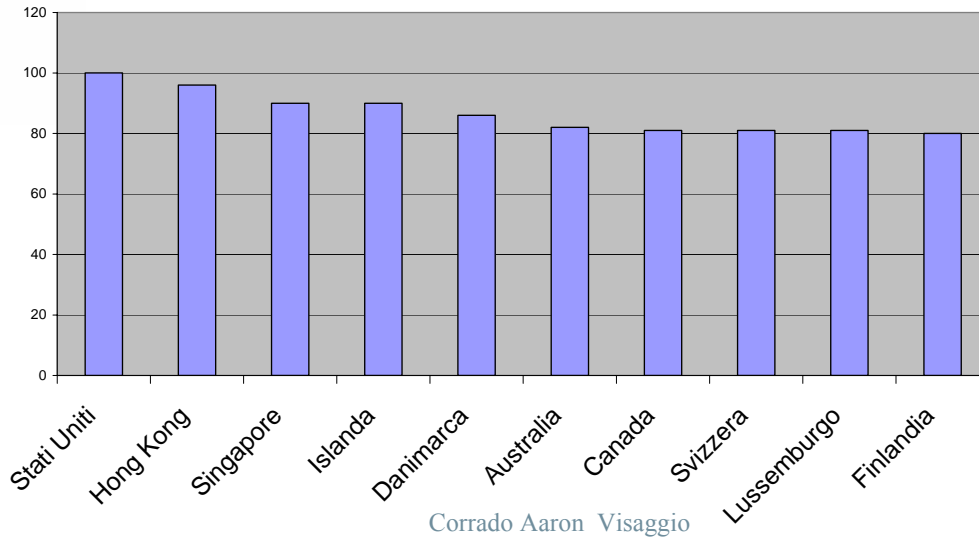
Corrado Aaron Visaggio

2

Classifica Internazionale Competitività

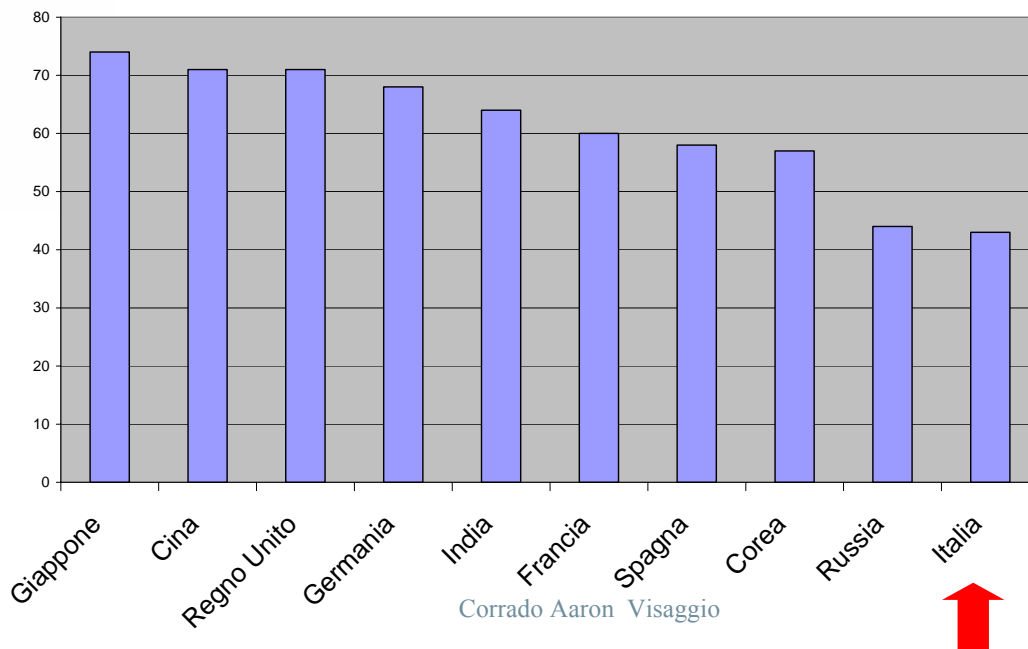
Fonte: World Competitiveness Yearbook

I Primi dieci



3

Al 56° posto l'Italia



4



Cosa aspetta gli informatici?

Secondo il rapporto Prometeia-Banca Intesa, presentato l'11/05/06:

- Nel 2004 flessione dell'1,3%



Trainano l'inversione di tendenza: high-tech: meccanica, elettrotecnica, servizi informatici (di cui l'Italia è forte importatore)



Componenti

Ottimizzatore: decide le strategie di accesso ai dati per rispondere alle interrogazioni. Svolge analisi lessicali, sintattiche, semantiche, trasformando in una forma analoga all'algebra relazionale.

Gestore dei metodi di accesso ai dati: noto come Relational Storage System (RSS) esegue gli accessi fisici ai dati, in accordo alla strategia definita dall'ottimizzatore.

Buffer Manager: gestisce il trasferimento delle pagine della base di dati dai dispositivi di memoria di massa alla memoria centrale.

Controllore dell'affidabilità: gestisce il ripristino del sistema durante i malfunzionamenti e guasti

Controllore della concorrenza: regola gli accessi concorrenti alla base di dati



Transazione

Una **transazione** identifica un'unità di lavoro elementare svolta da un'applicazione, cui si vogliono associare caratteristiche di **robustezza, correttezza ed isolamento** [sistemi transazionali].

Una transazione va a buon fine solo a seguito di un **commit** [**commit work**]; altrimenti, in caso di **abort** [**rollback work**] non produce alcun effetto tangibile sulla base di dati.



Una transazione è **ben formata** se:

- Comincia con begin transaction (bot)
- Termina con end transaction (eot)
- Nel suo corso viene eseguito **uno solo** dei due comandi (commit work o rollback work)
- A **seguito di questi due comandi** non avvengono operazioni di accesso o modifica alla base di dati

```
Begin transaction
x= x*2;
y=y+4;
if y>x commit work
else rollback work;
end transaction
```

Corrado Aaron Visaggio



Proprietà Acide delle transazioni

L' **Atomicità**. Una transazione è **un'unità indivisibile di esecuzioni**. Gli **effetti** della transazione o sono tutti visibili (commit) o nessuno di essi lo è (abort). Le operazioni che compongono la transazione possono essere disfatte (**undo**) o, in caso di commit, potrebbero essere ripetute (**redo**).

Il **rollback** può essere invocato dalla stessa transazione o dal sistema.

La **Consistenza**. Richiede che l'esecuzione della transazione **non violi i vincoli di integrità** definiti sulla base di dati.

La **verifica** del vincolo può essere **immediata**, all'interno della stessa transazione senza provocare un abort; oppure **differita**, alla conclusione della transazione dopo il commit: il sistema dopo il commit annulla gli effetti della transazione.

L'**Isolamento**. Richiede che l'esecuzione di una transazione sia **indipendente dalla esecuzione contemporanea di altre transazioni**.

La **Persistenza**. Richiede che **gli effetti** di una transazione **si mantengano** anche dopo il commit.



Proprietà e moduli del sistema

Atomicità e persistenza -> Controllo di affidabilità

Isolamento -> controllo di concorrenza

Consistenza -> compilatori di DDL



Attenzione:

- Una **transazione utente** può incapsulare diverse transazioni di sistema
- Una **transazione di sistema** può incapsulare diverse transazioni utente



Controllo di Concorrenza

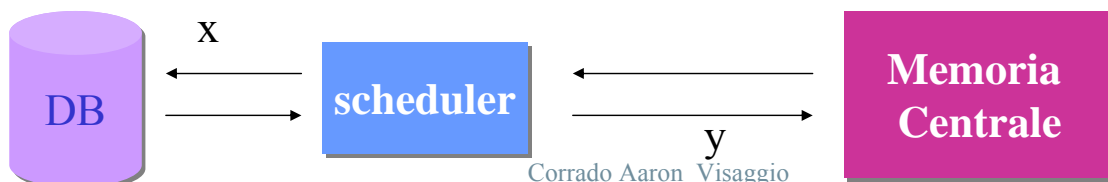
Una unità di misura spesso utilizzata per valutare le **prestazioni** di un DBMS è il **numero di transazioni al secondo** (tps). L'ordine di grandezza può variare da alcune centinaia ad alcune centinaia di migliaia. *L'esecuzione seriale è impensabile.*

Il **controllo di concorrenza** riguarda le **operazioni di ingresso/uscita sui blocchi** (pagine):

- **Read**: da memoria di massa a quella centrale.
- **Write**: da memoria centrale a quella di massa.

Lo **scheduler** autorizza la read/write di un blocco.

Si parlerà di $r(x)$ e $w(x)$, dove x è una pagina, anche se viene trattato come dato numerico.



Anomalie delle transazioni concorrenti...

Perdita di aggiornamento. Si considerino le seguenti transazioni:

- $t_1: r(x), x=x+1, w(x);$
- $t_2: r(x), x=x+1, w(x);$

Dato $x=2$, l'esecuzione consecutiva dovrebbe produrre il valore 4 al seguito dell'esecuzione delle transazioni.

Invece...

Perdo gli effetti di $t_1!$

X=2
X=2
X=3

X=2
X=3
X=3

X=3

Transazione t1	Transazione t2
bot	
r1(x) ●	
$x = x + 1$ ●	
	bot ●
	r2(x) ●
	$x = x + 1$ ●
	w2(x) ●
	Commit ●
w1(x) ●	
Commit ●	

...Anomalie delle transazioni concorrenti...

Letture Sporche. Si considerino le seguenti transazioni:

- $t_1: r(x), x=x+1, w(x)$ [con abort];
- $t_2: r(x), x=x+1, w(x);$

Dato $x=2$, l'esecuzione consecutiva dovrebbe produrre il valore 3, in quanto una transazione è annullata.

Invece...

Perdo gli effetti di $t_1!$

X=2
X=2
X=3
X=3

X=3
X=4
X=4

Transazione t1	Transazione t2
bot	
r1(x) ●	
$x = x + 1$ ●	
w1(x) ●	
	bot ●
	r2(x) ●
	$x = x + 1$ ●
	w2(x) ●
	Commit ●
Abort ●	

...Anomalie delle transazioni concorrenti...

Letture Inconsistenti. Si considerino le seguenti transazioni:

- t1: r(x), r(x);
- t2: r(x), x=x+1,w(x);

Dato x=2, t1 dovrebbe leggere sempre il valore 2 di x.

Invece...

t1 risente gli effetti di t2!

	Transazione t1	Transazione t2
X=2	bot	
X=2	r1(x) ●	
		bot ●
X=2		r2(x) ●
X=3		x = x + 1 ●
X=3		w2(x) ●
		Commit ●
X=3	r1(x) ●	
	Commit ●	

Aggiornamento fantasma. Si considerino le seguenti transazioni:

- t1: r(x), r(y), r(z), s= x+y+z;
- t2: r(y), y=y-100, r(z), z = z+100, w(y),w(z);
- Vincolo di integrità: x+y+z=1000.

Nessuna delle due transazioni viola il vincolo di integrità.

Invece...

Gli effetti della concorrenza alterano il vincolo!

	Transazione t1	Transazione t2
	bot	
X=400	r1(x) ●	
		bot
Y=300		r2(y) ●
Y=300	r1(y) ●	
Y=200		y=y-100 ●
Z=300		r2(z) ●
Z=400		z=z+100 ●
Y=200		w2(y) ●
Z=400		w2(z) ●
X=400		Commit ●
Y=200	R1(z) ●	
Z=400	S=x+y+z ●	
	Commit ●	

Z=400
S=400
+ 300
+ 400
= 1100



Teoria del controllo di concorrenza...

Una transazione include letture e scritture caratterizzate da uno **stesso indice**.

Una transazione t_1 è rappresentata in questo modo:

$r_1(x)r_1(y)w_1(x)w_2(y)$

Uno schedule S_1 rappresenta la sequenza di operazioni di ingresso/uscita concorrenti:

$r_1(x)r_2(z)w_1(x)w_2(z)$

Il **controllo di concorrenza** ha lo scopo di accettare alcuni schedule e rifiutarne altri, in modo che *non si verifichino anomalie* (scheduler).

Si considereranno gli schedule **commit-proiezione**, che includono solo transazioni che producono un commit.

- Questa assunzione è inaccettabile nella pratica (non includerebbe l'anomalia da lettura sporca)

Si definisce **seriale** uno schedule in cui tutte le operazioni appartenenti ad una stessa transazione appaiono in sequenza:

$r_0(x)r_0(y)w_0(x)r_1(y)r_1(x)w_1(y)r_2(x)r_2(y)r_2(z)w_2(z)$



... Teoria del controllo di concorrenza...

L'**esecuzione** della commit-proiezione di uno schedule S_i è **corretta** quando produce lo stesso risultato prodotto da un qualunque schedule seriale S_j delle stesse transazioni. In tal caso S_i si dice **serializzabile**.

View-equivalenza.

- Una operazione di lettura $r_i(x)$ **legge da** una scrittura $w_j(x)$ quando $w_j(x)$ precede $r_i(x)$ e non vi è alcuna $w_k(x)$ compresa tra le due operazioni.
- $w_i(x)$ è detta **finale** se è l'ultima lettura che appare nello schedule
- Due schedule sono **view-equivalenti** se possiedono la stessa **legge da** e la stessa **scrittura finale**.

View Equivalenza...

S3 : w0(x)r2(x)r1(x)w2(x)w2(z)

— Legge da

S4: w0(x)r1(x)r2(x)w2(x)w2(z)

— Scritture
Finali

S5:w0(x)r1(x)w1(x)r2(x)w1(z)

S6:w0(x)r1(x)w1(x)w1(z)r2(x)

S2 ed S4 sono view-
equivalenti

... View Equivalenza...

S3 : w0(x)r2(x)r1(x)w2(x)w2(z)

— Legge da

S4: w0(x)r1(x)r2(x)w2(x)w2(z)

— Scritture
Finali

→ S5:w0(x)r1(x)w1(x)r2(x)w1(z)

→ S6:w0(x)r1(x)w1(x)w1(z)r2(x)

S5 ed S6 sono view-
equivalenti

... View Equivalenza...

S3 : $w_0(x)r_2(x)r_1(x)w_2(x)w_2(z)$ — Legge da

S4: $w_0(x)r_1(x)r_2(x)w_2(x)w_2(z)$ — Scritture Finali

S5: $w_0(x)r_1(x)w_1(x)r_2(x)w_1(z)$

S6: $w_0(x)r_1(x)w_1(x)w_1(z)r_2(x)$

S4 ed S5 non sono view-equivalenti

... View Equivalenza...

Determinare la view-equivalenza è un problema con **complessità lineare**.

Determinare se uno schedule è view-equivalente ad un qualsiasi schedule seriale è un problema **NP-difficile**.

E' **conveniente** confrontare tra loro due schedule.

Se non viene fornito uno schedule di confronto, invece, bisogna confrontare lo schedule con tutti gli schedule seriali ottenuti, permutando tutte le possibili transazioni.

Si preferisce, quindi definire una **condizione di equivalenza più ristretta**, che non copra tutti i casi di equivalenza, ma possa essere usata nella pratica.

Conflict-equivalenza



Si dice che l'azione a_i è in conflitto con a_j , dove $i \neq j$, se entrambe operano sullo stesso oggetto e almeno una di esse è una write: conflitti di **lettura-scrittura** e **scrittura-scrittura**.

Si dice che uno schedule S_i è **conflict-equivalente** con uno schedule S_j se presentano **le stesse operazioni** ed ogni coppia di **operazioni in conflitto è nello stesso ordine**.

Uno schedule risulta **conflict-serializzabile**, se esiste uno **schedule seriale** ad esso **conflict-equivalente**.

La classe degli schedule CSR è strettamente inclusa in quella dei VSR. La conflict serializzabilità è condizione sufficiente ma non necessaria per la view-serializzabilità.

Esempio

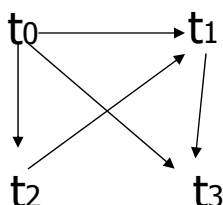
S_{10} : $w_0(x)r_1(x)w_0(z)$ $r_1(z)r_2(x)r_3(z)w_3(z)w_1(x)$



S_{11} : $w_0(x)w_0(z)r_2(x)r_1(x)r_1(z)w_1(x)r_3(z)w_3(z)$



Grafo dei conflitti: gli archi identificano i conflitti tra le azioni delle transizioni t_i .



Si dimostra che lo schedule è in CSR se e solo se il grafo è aciclico.

Sebbene questo problema abbia ancora linearità lineare, determinare i grafi di conflitti in basi di dati distribuite è troppo oneroso

Locking a 2 fasi...

Il locking si basa sull'idea che tutte le operazioni di scrittura e lettura debbano essere protette tramite l'esecuzione di tre primitive: r_lock, w_lock, unlock.

Vincoli per una transazione ben formata rispetto al lock:

- Ogni operazione di lettura deve essere preceduta da un r-lock e seguita da un unlock. Il lock è **condiviso**
- Ogni operazione di scrittura deve essere preceduta da un w_lock e seguita da un unlock. Il lock è **esclusivo**.

Se la richiesta di lock non viene concessa, la transazione è posta in **stato di attesa**. L'attesa termina quando la risorsa viene sbloccata e diviene disponibile. I lock concessi sono tracciati in **tabelle di lock**.

La **tabella dei conflitti** illustra la politica seguita dal lock manager nel concedere le risorse.

... Locking a 2 fasi

Per essere certi che le transazioni seguano uno schedule serializzabile è necessario porre una restrizione:

Locking a 2 fasi:

Una transazione, dopo aver rilasciato un lock non può acquisirne altri.

Si distinguono due fasi:

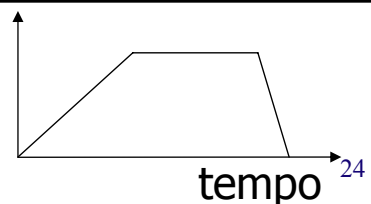
Acquisizione della risorsa

Rilascio della risorsa

Ogni schedule che rispetti il 2PL è serializzabile secondo la conflict-equivalenza.

Richiesta	Stato Risorsa		
	Libero	R_locked	W_locked
R_lock	Ok/ r_locked	Ok/ r_locked	No / w_locked
W_lock	Ok/ w_locked	No/r_locke d	No/w_lock ed
unlock	error	Ok/ dipende	Ok/libero

risorse





timestamp

Rimuoviamo la condizione di commit-proiezione...

2PL stretto: i lock di una transazione possono essere rilasciati dopo aver correttamente effettuato le operazioni di commit/abort.

Il **controllo di concorrenza basato sui time-stamp** avviene in questo modo:

- Ad ogni transazione si associa un **timestamp** che rappresenta l'inizio della transazione.
- Uno schedule si accetta solo se esso riflette l'**ordinamento seriale** delle transazioni.
- Ad ogni oggetto vengono associati due indicatori **WTM(x)** [ultima scrittura] e **RTM(x)** [ts più grande che ha letto x]

Read(x,ts): se $ts < WTM(x)$, la transazione è uccisa. Se la transazione è accettata **RTM(x)** aggiornato al massimo tra ts e **RTM(x)**.

Write(x,ts): se $ts < WTM(x)$ o $ts < RTM(x)$, la transazione è uccisa. Se la transazione è accettata, **WTM(x)** è posto pari a ts.



Blocco critico

Si prendano due transazioni, $t_1: r(x) w(y)$ e $t_2: r(y) w(x)$

$r_lock1(x)$, $r_lock2(y)$, $read1(x)$, $read2(y)$, $w_lock1(y)$, $w_lock2(x)$.

Tre tecniche:

- **Timeout:** la risposta al lock deve avvenire entro un certo tempo, dopo il quale esso è concesso oppure no.
- **Prevenzioni:**
 - Richiedono tutte le risorse in anticipo
 - Instaurare relazioni di precedenza tra timestamp
- **Rilevamento dei blocchi critici:** si controlla il contenuto delle tabelle dei lock.

Organizzazione del Log...

Il log è un file di cui è responsabile il **controllore della affidabilità** scritto in **memoria stabile**.

I **record** (di transazione e di sistema) del log vengono **scritti sequenzialmente** nel blocco corrente, fino ad esaurimento del blocco.

I record di transazione registrano le attività svolte da ciascuna transazione nell'ordine di esecuzione.

I record di log sono:

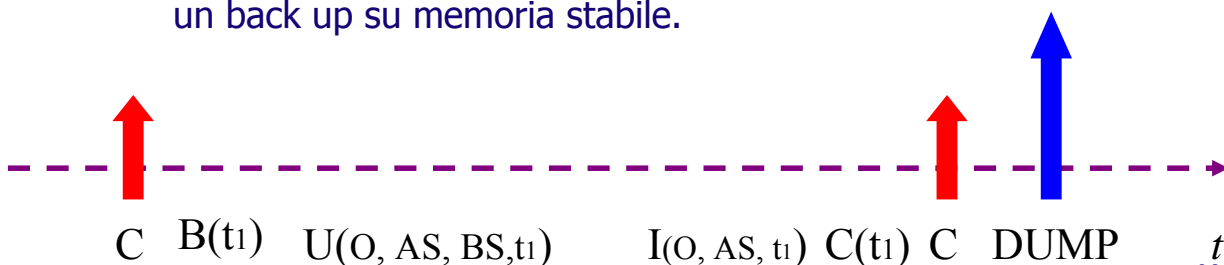
- Record di **begin, commit ed abort** con l'indicazione della transazione t .
- Record di **update** con l'indicazione della transazione t l'**oggetto O** che è oggetto dell'aggiornamento e i due valori **Before State (BS)** ed **After State (AS)**, di O
- Record di **insert e delete**, uguali a quelli di update ma senza BS (insert) e AS (delete)
- Primitiva di **undo**: per disfare un'azione sull'oggetto O è sufficiente ricopiarvi il suo BS; nel caso di insert bisogna eliminarlo.
- Primitiva di **redo**: per ripetere un'azione sull'oggetto O è sufficiente ricopiarvi il valore AS; nel caso di delete bisogna eliminarlo.

...Organizzazione del Log

I record di sistema indicano operazioni di **dump** e di **checkpoint**.

Il **checkpoint** registra quali **transazioni sono attive**; tale operazione è coordinata con il **buffer manager**: vengono **registrati gli id (force)** delle transazioni in corso e si realizza il **flush** di tutte le pagine delle transazioni che hanno già effettuato il commit o l'abort. Le transazioni attive vengono scritte nel log.

Un **dump** è una **copia completa** della base di dati effettuata in modo **mutuamente esclusivo** con le transazioni attive. Si produce un back up su memoria stabile.





Gestione delle transazioni

Il **controllore dell'affidabilità** deve garantire che siano verificate due regole:

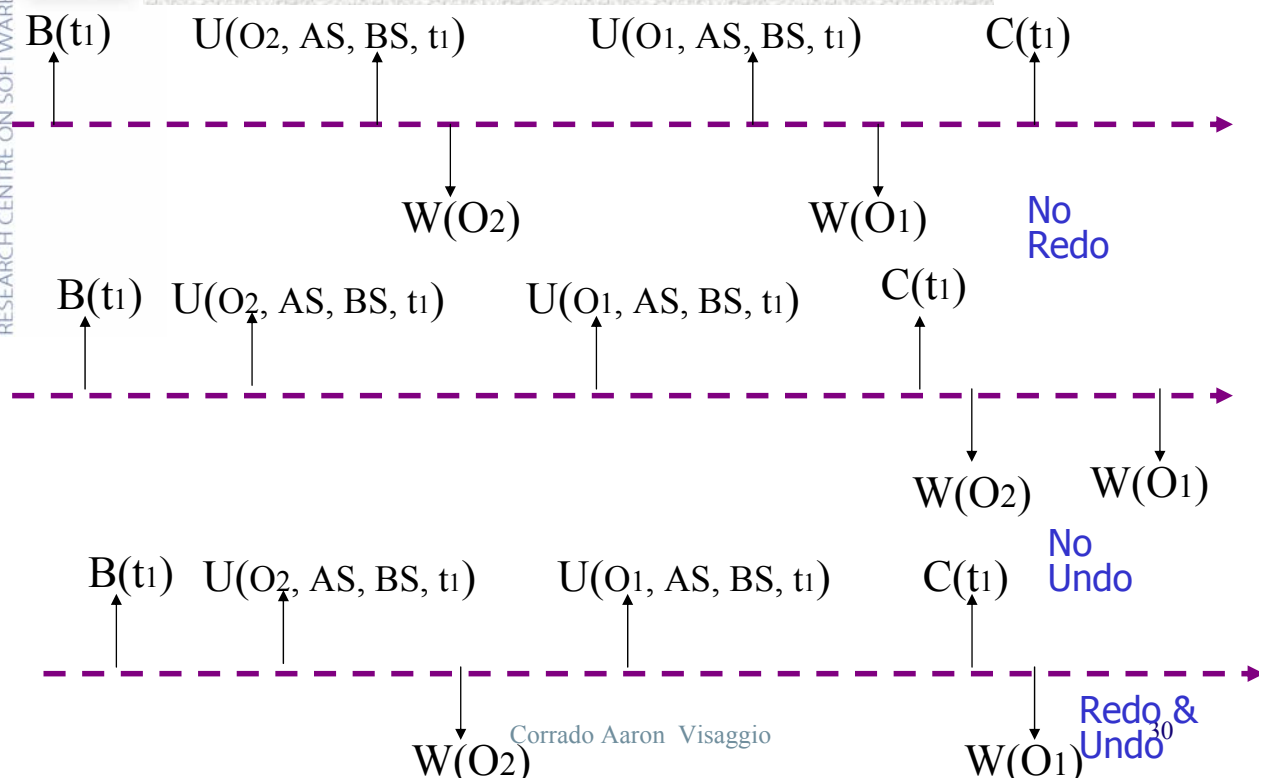
- **WAL**: write ahead Log, la parte **before state** di un record venga scritta prima di effettuare le rispettiva operazione sulla base di dati
- **Commit-precedenza**: la parte **after state** venga scritta nel log prima di effettuare il commit.

La transazione sceglie l'esito del commit in modo sincrono (force) alla scrittura del record di commit sul log. Prima del commit, un eventuale guasto comporta l'Undo; un eventuale guasto dopo la commit comporta un Redo.

Il record di abort può essere scritto in modo asincrono nel buffer.



Scrittura congiunta di log e base di dati



Gestione dei guasti

I **guasti di sistema** sono indotti da **malfunzionamenti del software** oppure da malfunzionamenti di rete o **dispositivi esterni al software**. Minano la memoria centrale, ma non quella di massa.

I **guasti di dispositivo** riguardano i **dispositivi di gestione della memoria di massa**. In tal caso si perde il database ma non il log, scritto su memoria stabile.

Il modello ideale è di **fail-stop**: un guasto causa l'arresto del sistema ed il ripristino del sistema operativo:

- Ripresa a freddo, nel caso di guasto di dispositivo
- Ripresa a caldo, nel caso di guasto di sistema.

Per le transazioni attive al momento del guasto:

- Redo per quelle che hanno effettuato il commit
- Undo per quelle che non hanno effettuato il commit

In alcuni casi si aggiunge un **record di end** per identificare il termine delle operazioni di flush, ed evitare le operazioni di undo e redo superflue.

Strutture fisiche di accesso...

Il gestore degli accessi trasforma il piano di accesso (ottimizzatore) in opportune sequenze di accesso alle pagine del db.

I **metodi di accesso** conoscono:

- Le **procedure di accesso** ai blocchi del db;
- La **disposizione delle tuple** nelle pagine.

I metodi di accesso possono condividere le strutture dati: ogni pagina contiene **informazioni utili** (i dati veri e propri) e l'**informazione di controllo**. Ogni pagina ha :

- un **block header** ed un **block trailer** per il file system.
- un **page header** ed un **page trailer** per il metodo di accesso: id dell'oggetto, contenuto, puntatori, info sulla memoria utilizzata
- **Dizionario di pagina**: puntatori a ciascun dato utile e parte utile
- **Bit di parità**, per verificare la validità dell'informazione contenuta.

...Strutture fisiche di accesso

Alcuni gestori non consentono di dividere una tupla su più pagine. In alcuni casi le tuple possono avere **dimensioni fisse** (**memory waste**) oppure **variabili** (**overhead di informazione** per il dizionario di pagina).

In alcuni casi tuple di tavole diverse possono trovarsi nella stessa pagina.

Le primitive offerte dal gestore delle pagine sono:

- Inserimento ed aggiornamento
- Cancellazione
- Accesso ad una tupla particolare
- Accesso ad un campo di una particolare tupla.

Strutture sequenziali...

Le tuple sono inserite nei blocchi in modo sequenziale, di vario tipo:

- Organizzazione **entry-sequenced**:
 - la sequenza delle tuple è indotta dal loro ordine di immissione.
 - Ottima per svolgere **operazioni di lettura/scrittura sequenziale**.
 - Si accede tramite scansione seq. Il caricamento e l'inserimento sono efficienti perché avvengono in modo sequenziale.
 - La modifica e la cancellazione sono i **punti di debolezza**: ricerca della tupla e spazio vuoto.
- Organizzazione ad **array**:
 - le tuple sono disposte come in un array e la **loro posizione** dipende dal **valore dell'indice**
 - Possibile solo con tuple di **dimensione fissa**
 - Ciascun blocco è posta nella i -ima posizione, ove i è l'indice.
 - Gli inserimenti devono essere effettuati negli slot lasciati liberi dalle cancellazioni
 - Read-ind, delete-ind, insert-at, insert-near, insert-at-end



... Strutture sequenziali

- Organizzazione **sequenziale ordinata**:
 - la sequenza delle tuple dipende dal valore assunto in ciascuna tupla da un **campo di ordinamento**, detto chiave (key).
 - Ha **costi di gestione** molto pesanti, per cui **decaduta**.
 - L'ordinamento delle tuple riflette l'**ordinamento lessicografico** dei valori della chiave.
 - Sono avvantaggiate le **ricerche casuali** (ricerca dicotomica)
 - Ogni inserimento comporta un riordinamento delle tuple già presenti in memoria.
 - Si possono prevedere **slot di memoria** proprio per i futuri inserimenti e disseminarli nella struttura.
 - Si può integrare il file sequenziale con uno di **overflow**. I blocchi di overflow partono da un blocco del file ordinario.



Strutture con accesso calcolato...

Una struttura con accesso calcolato consente un **accesso associativo ai dati**. Preferibile perché le **tuple non devono comparire in ordine sequenziale** in memoria di massa.

Ogni file ha un numero B di blocchi; il gestore dispone di un algoritmo di calcolo che associa ad ogni chiave un blocco.

In **assenza di collisione** il rendimento è ottimale: 1 sola op di L/S.

Hash(Fileid, key):blockid:

- Operazione di **folding**, trasforma le chiavi in valori interi positivi come sequenze di bit.
- Operazione di **hashing**, trasforma le sequenze di bit in interi compresi tra 0 e B .

... Strutture con accesso calcolato

Questa tecnica funziona bene se è previsto un **sovradimensionamento**, ovvero se: T rappresenta il numero di tuple per il file ed F il numero di tuple per ogni pagina $\Rightarrow B$ è il numero primo immediatamente superiore a $T/(0.8 \times F)$.

Il problema principale di queste strutture è la **probabilità di collisione**:

- Il problema della collisione sussiste quando per lo stesso blocco si eccede il valore F .
- Quando si verifica un numero eccessivo di collisioni e la capacità della pagina è esaurita, la soluzione proposta da questa organizzazione dei dati consiste nel creare **catene di overflow**, ma si rallenta il tempo di ricerca.

Strutture ad albero

Le **strutture ad albero B tree o B+-tree** consentono **accessi associativi** sulla base di **uno o più valori delle chiavi**, svincolando la tupla dal vincolo di una specifica collocazione fisica.

I legami tra i nodi (radice-intermedio-foglia) sono realizzati a mezzo di **puntatori**.

Perché i **tempi di accesso siano costanti**, gli **alberi** devono essere **bilanciati** (cammini radice-foglia uguali).

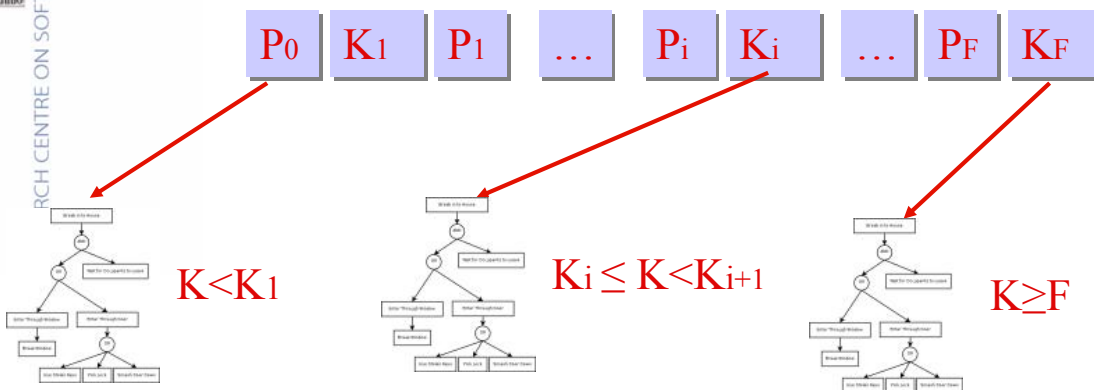
Ogni nodo ha, in genere un numero ampio di discendenti, per cui le pagine sono occupate per lo più da nodi foglia.

Tecnica di ricerca...

Ogni chiave K_i $1 \leq i \leq F$ è seguita da un puntatore P_i ; K_1 è preceduta da P_0 . Ciascun puntatore utilizza un sottoalbero caratterizzato in questo modo:

- P_0 indirizza al sottoalbero che contiene l'informazione relativa alle chiavi strettamente inferiori a K_1
 - P_F indirizza al sottoalbero che contiene l'informazione relativa alle chiavi uguali o superiori a K_F .
 - Ciascun puntatore intermedio, indirizza un sotto-albero che contiene tutta l'informazione relativa alle chiavi K_j comprese nell'intervallo $K_i \leq K_j \leq K_{i+1}$
- ➔ Ciascun nodo contiene F chiavi e $F+1$ puntatori; $F+1$ è detto **fan out** dell'albero. F dipende dall'ampiezza della pagina e dalla **dimensione occupata dai valori di chiave**, e di puntatori della parte utile dell'albero.
- ➔ F deve essere il più grande possibile per limitare il numero di livelli.

... Tecnica di ricerca...



Il meccanismo di ricerca consiste nel seguire i puntatori: Si cerca la chiave che ha valore V :

- Se $V < K_1$ si segue il puntatore P_0 ;
- Se $V \geq K_F$ si segue il puntatore P_F ;
- Altrimenti si segue il puntatore P_j tale che $K_j \leq V < K_{j+1}$

... Tecnica di ricerca...

La ricerca prosegue in questo modo fino ai **nodi foglia** che possono essere organizzati nel seguente modo:

- Essi possono contenere **l'intera tupla**. La struttura dati che si ottiene in questo modo è chiamata **key sequenced** [la **posizione** di una tupla è vincolata dal valore del campo chiave]
- Ciascun nodo foglia contiene **puntatori ai blocchi della base di dati** che contengono tuple con il valore di chiave specificato. La **struttura dati** che si ottiene in questo modo è detta **indiretta**. Il posizionamento delle tuple nel file può essere uno qualsiasi.
- Talvolta nell'indice non sono compresi tutti i valori della chiave [**indice sparso**]. E' di solito costruito su una **struttura sequenziale ordinata**: si localizza, con l'indice un valore di chiave prossimo al valore ricercato e, quindi, si svolge una ricerca sequenziale.

... Tecnica di ricerca...

La struttura **key sequenced** è preferita per realizzare **l'indice primario** (primary key). Chiave-> definizione della tavola; Indice-> implementazione della tavola.

La **struttura indiretta** è preferita per realizzare gli **indici secondari** che possono essere unique o multiple [ad ogni valore della chiave corrispondono più tuple, ciascuna indirizzata con un diverso puntatore].

Inserimenti e cancellazioni di tuple provocano anche aggiornamenti degli indici.

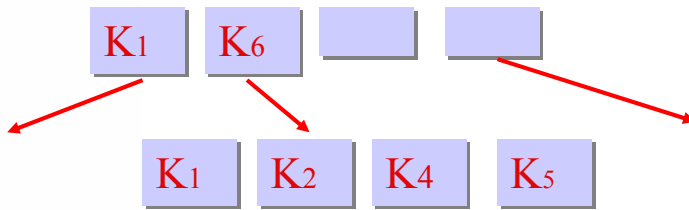
Un **inserimento** non provoca problemi quando vi sono slot liberi.

- Se non vi è spazio disponibile bisogna realizzare uno **split**, allocando due foglie al posto di una. Lo split potrebbe propagarsi verso l'alto aumentando la profondità dell'albero.

Una **cancellazione** marca uno slot allocato come invalido.

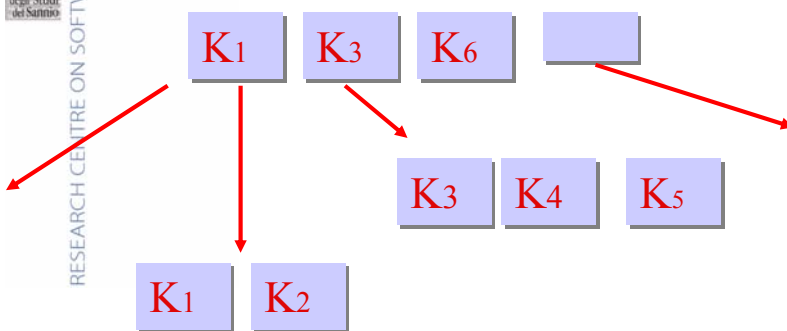
- Si procede ad un **merge**, che ricolloca tutti i puntatori, e si potrebbe propagare fino alla radice.
- Bisogna recuperare il successivo valore dalla base di dati ed inserirlo al posto di quello eliminato.

... Tecnica di ricerca...



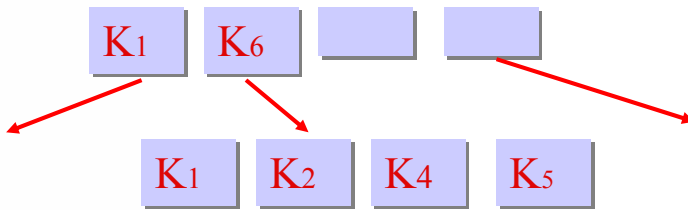
Inserisco K3

... Tecnica di ricerca...



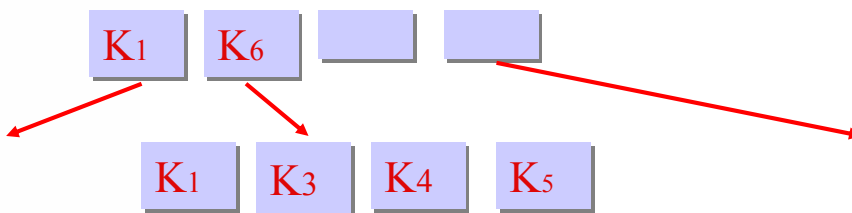
split

... Tecnica di ricerca...



elimino K2

... Tecnica di ricerca...



merge

Alberi B+

Negli alberi B+ i nodi foglia sono collegati da una catena che li connette in base all'ordine imposto dalla chiave.

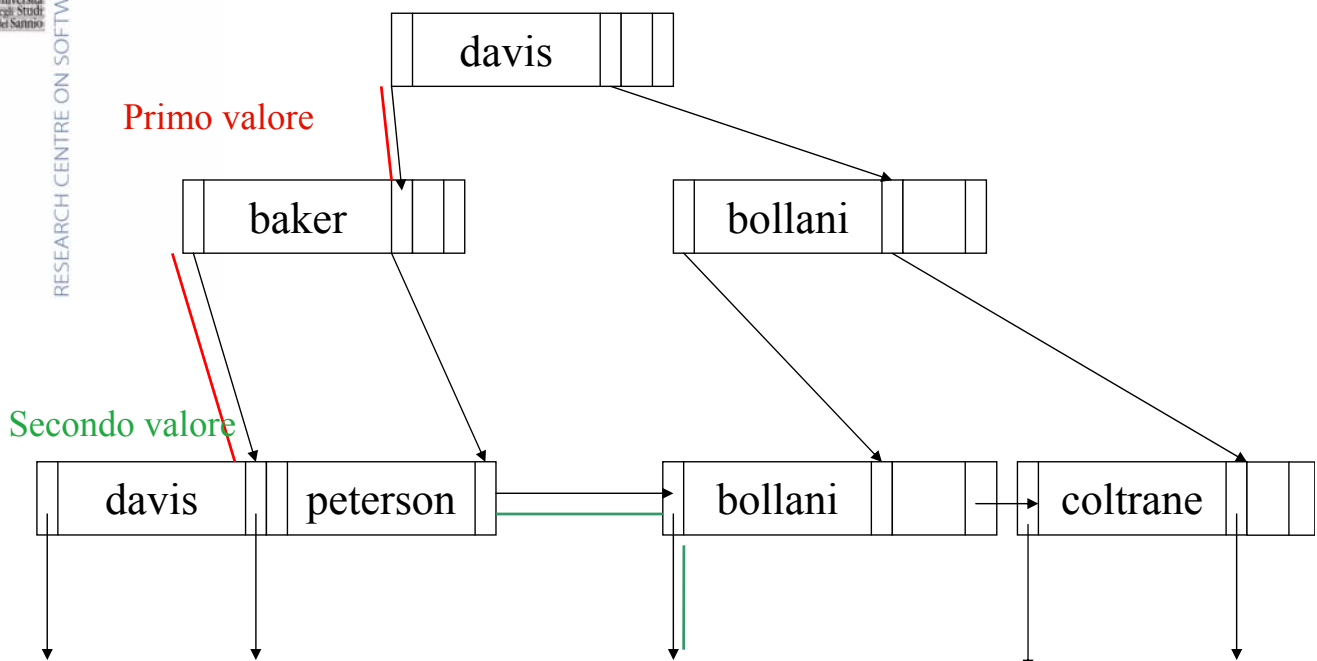
Tale catena consente di svolgere in modo efficiente interrogazioni il cui predicato di selezione definisce un intervallo di valori.

Si accede al primo valore dell'intervallo e si scandiscono tutti i nodi foglia fino ai valori chiave maggiori del secondo elemento dell'intervallo.

- Nel caso key sequenced la risposta si compone di tutte le tuple così trovate.
- Nel caso indiretto bisognerà accedere a tutte le tuple.

Questo metodo è utilizzato nella maggior parte dei DBMS.

Un esempio



Ottimizzazione

Si possono utilizzare **due puntatori** nei nodi intermedi per **ogni valore della chiave K_i** :

- Uno punta direttamente al blocco che contiene la tupla corrispondente a K_i ;
 - L'altro puntatore potrebbe puntare ai valore di chiave compresi tra K_i e K_{i+1}
- Questa tecnica fa **risparmiare spazio nelle pagine dell'indice** e consente di terminare la ricerca senza dover percorrere tutti i livelli.
- Le pagine contenenti i **primi livelli** possono divenire molto richieste; per fortuna l'accesso avviene, in questo caso, in sola lettura.

Ottimizzazione delle interrogazioni

L'interrogazione viene analizzata per rimuovere eventuali errori lessicali, sintattici o semantici. L'interrogazione viene quindi tradotta in una **forma algebrica**. Si compone delle seguenti fasi:

- Si realizzano le trasformazioni algebriche più convenienti al caso in questione
- Si svolge un'ottimizzazione che dipende dai metodi di accesso e dal modello dei costi assunto.
- Si genera il codice che utilizza i metodi di accesso al codice.

L'ottimizzatore agisce a tempo di compilazione: **compile and store**, e **compile and go**.



Profili delle interrogazioni

Il dizionario dei dati cattura alcune informazioni relative alle caratteristiche proprie delle relazioni:

- Cardinalità della tabella R: $CARD(R)$
- Dimensione in byte di una tupla T: $SIZE(T)$
- Dimensione in byte di ciascun attributo A_j di T: $SIZE(A_j, T)$
- Numero di valori distinti di ciascun attributo: $VAL(A_j, T)$
- valori minimo e massimo di ciascun attributo: $MIN/MAX (A_j, T)$



Profili relativi alle selezioni

Il profilo di una tabella T' prodotta da una selezione:

$$CARD(T') = (1/VAL(A_i, T)) \times CARD(T)$$

$$SIZE(T') = SIZE(T)$$

$$VAL(A_i, T') = 1;$$

$$VAL(A_j, T') = col(CARD(T), VAL(A_j, T), CARD(T')), \quad j \neq i$$

$$MAX(A_i, T') = MIN(A_i, T')$$

$MAX(A_j, T')$ e $MIN(A_j, T')$ rimangono immutati per $j \neq i$

Rappresentazione interna delle interrogazioni...

La rappresentazione data dall'ottimizzatore tiene conto della struttura fisica delle tabelle e dei suoi indici.

- I nodi foglia sono adeguati alle strutture fisiche delle tabelle
- I nodi intermedi sono trasformati in operazioni di accesso.

Un'operazione di scansione:

- Proiezione su una lista di attributi
- Selezione su un predicato semplice.
- Ordinamento in base al valore degli attributi
- Inserimenti, cancellazioni e modifiche delle tuple durante gli accessi
- Si mantiene sempre un puntatore alla tupla corrente.

...Rappresentazione interna delle interrogazioni...

Il problema dell'ordinamento attualmente non è algoritmico, ma di spazio nel buffer.

Gli indici sono predisposti dall'amministratore per favorire predicati semplici o di intervallo.

→ In una **congiunzione di predicati** l'indice che viene usato per primo è quello che esegue una maggiore selezione.

In **caso di disgiunzione** si utilizzano indici e scansione.

Il join è ritenuto pericoloso per la possibilità di esplosione delle tuple risultato.

Tre tecniche per la visualizzazione del join:

- **Nested Loop**: per ogni valore dell'attributo di join della tabella esterna, si alloca un indice per ogni tupla coinvolta nel join, e, se necessario, si esegue una scansione.

...Rappresentazione interna delle interrogazioni...

- **Merge Scan**: entrambe le tabelle sono ordinate in accordo all'attributo di join. Le due tabelle vengono scandite in parallelo e laddove possibile, si creano le tuple risultato.
- **Hash-based**: tramite delle funzioni hash si partizionano entrambe le tavole. Le tuple risultato del join saranno calcolate effettuando B join nelle rispettive partizioni.

Un ottimizzatore deve scegliere:

- Le operazioni di accesso ai dati
- L'ordine delle operazioni
- Quale alternativa associare alle operazioni
- Stabilire la strategia di ordinamento.

Gli ottimizzatori dispongono di **formule di costo approssimate**, in base alle quali producono un **albero delle alternative**;

Ogni foglia dell'albero corrisponde ad una specifica strategia di esecuzione.

...Rappresentazione interna delle interrogazioni

Tipicamente il costo si calcola come:

$$C_{tot} = C_{I/O} \times n_{I/O} + C_{cpu} \times n_{cpu}$$

Talvolta è necessario **creare strutture ad hoc**. In questo caso bisogna considerare il **costo addizionale** legato a tali strutture.

Gli ottimizzatori si accontentano di ottenere soluzioni buone.

Vi sono strategie che eliminano sottoalberi parziali se il loro costo è superiore a quello della strategia globale.