



# ORGANIZZAZIONE FISICA

## FILE HEAP

É UN NON ORGANIZZAZIONE

I RECORD sono allocati in base all'ordine di inserimento

SE NON AMMETTIAMO DUPLICATI L'INSERIMENTO È COSTOSO  
E QUESTO PORTA A DELLE PRESTAZIONI PEGGIORI.  
ALTRIMENTI, SI INSERISCE IL BLOCCO ALL'ULTIMO POSTO

### RICERCA

DOBBIANO CERCARE IN TUTTO IL FILE

SE IL RECORD SI TROVA IN POSIZIONE  $m$  AVREMO  $m$  ACCESSI IN MEMORIA

BLOCCHI PER MEMORIZZARE  $N$  RECORD  $\rightarrow \frac{N}{\text{REC. PER BLOCCO}}$

SE LA CHIAVE NON ABBRTE DUPLICATI

$R$  = RECORD PER BLOCCO

$N$  = NUMERO DI RECORD

$$a = \frac{N}{R}$$



NUMERO BLOCCHE



PER ACCEDERE AL BLOCCO  $m$  SERVONO  $m$  ACCESSI

OBTENIAMO IL COSTO MEDIO SOMMO I COSTI DI ACCESSO E DIVIDO LA SOMMA PER IL NUMERO DI RECORD

$$\frac{R \cdot 1 + R \cdot 2 + \dots + R \cdot m}{N} : R \cdot (1 + \dots + m) : \frac{N}{m}$$

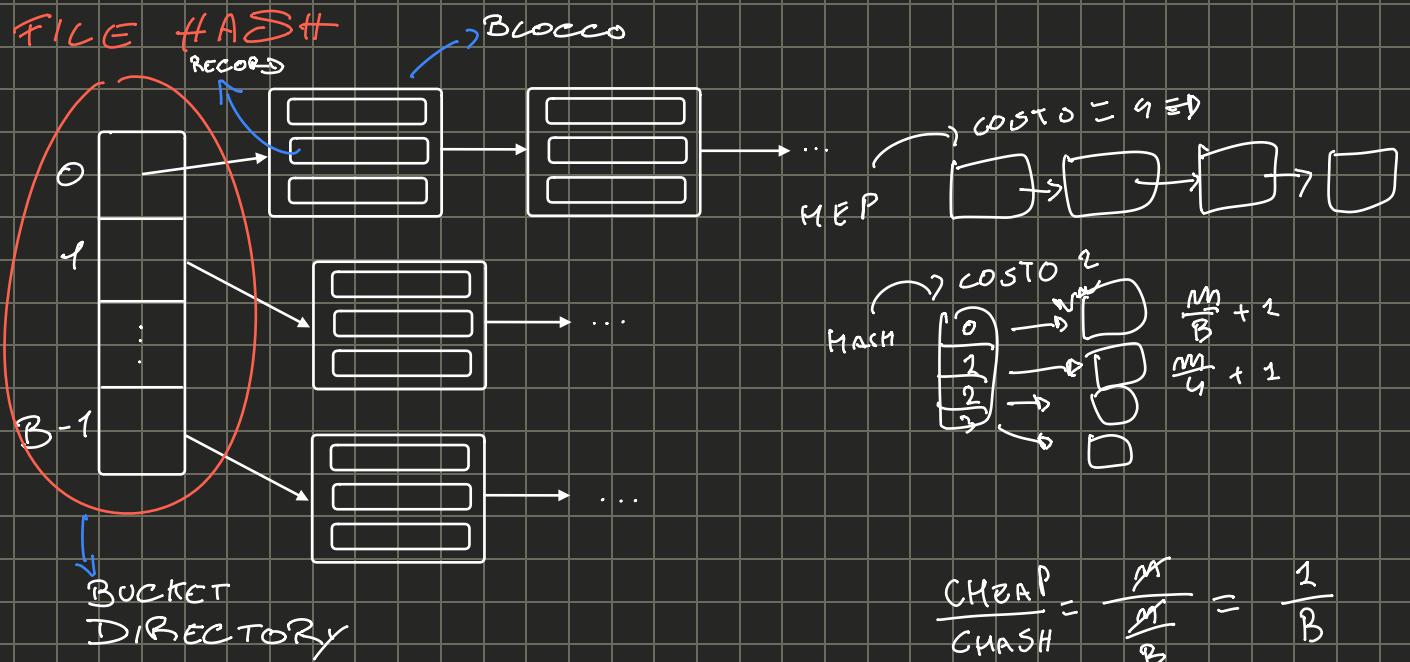
$$\text{SAPPiamo che } \frac{N}{R} = m \rightarrow \frac{1}{m} = \frac{R}{N}$$

$$\text{SOSTITUENDO OTTENIAMO CHE } \frac{1 + \dots + m}{m} : \frac{\sum_{k=1}^m k}{m}$$

RICORDANDO LA SOMMATORIA DI GAUSS

$$\frac{\sum_{k=1}^m k}{m} = \frac{m(m+1)}{2m} = \frac{m+1}{2} \approx \frac{m}{2}$$

COSTO MEDIO



È DIVISO IN BUCKET

UN BUCKET È COSTITUITO DA DA 10 PIZZI BLOCCHI CONNESI DA UN PUNTATORE

IN CUI C'È LA BUCKET DIRECTORY AL CUI INTERNO CI SONO I PUNTATORI AI BUCKET

## FUNZIONE HASH

DATO UN VALORE V PER FA CHE UNA FUNZIONE HASH  
IL NUMERO DEL BUCKET IN CUI SI TROVA È DATO DALLA FUNZIONE HASH

IN GENERE LA FUNZIONE "TRASFORMA" UNA CHIAVE IN UN NUMERO, LO DIVIDE PER B (NUMERO DEI BUCKET) E FORNISCE IL RESTO COME NUMERO DEL BUCKET IN CUI VERRÀ INSERITO IL RECORD

## OPERAZIONI

UNA QUALESiasi OPERAZIONE IN UN FILE HASH RICHIESTA:

- UNA VALUTAZIONE DELLA FUNZIONE HASH PER  $v, h(v)$  PER INDIVIDUARE IL BUCKET. MA IL COSTO O IN TERMINI DI ACCESSI IN MEMORIA
- ESECUZIONE DELL'OPERAZIONE SUL BUCKET CHE È ORGANIZZATO COME UN HEAP

L'INSERIMENTO VENE' EFFETTUATO SULL'ULTERIORE  
BUCKET.

LA FUNZIONE DI HASH DISTRIBUISCE IN TUTTO UNIFORME  
I RECORD NEI BUCKET.

QUINTO, OGNI BUCKET E' COSTRUITO DA  $\frac{m}{B}$  Blocco

NUMERO DI BLOCCO  
IN TUTTO IL FILE HASH  
↓  
NUMERO TOTALE  
DI BUCKET

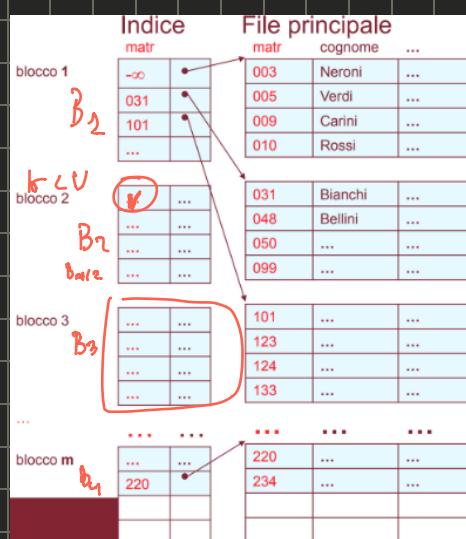
IL COSTO MEDIO E'  $\frac{m}{2B}$

# FILE CON INDICE

QUANDO LE CHIAVI PERMETTONO UN ORDINAMENTO SIGNIFICATIVO PER L'APPLICAZIONE CI CONVIENE UTILIZZARE UN'ORGANIZZAZIONE FISICA CHE NE TENGONO CONTO IN MODO DA AVERE PROVENTAGGI

## ISAM

	matr	cognome	...
blocco 1	003	Neroni	...
	005	Verdi	...
	009	Carini	...
	010	Rossi	...
blocco 2	031	Bianchi	...
	048	Bellini	...
	050	...	...
	099	...	...
blocco 3	101	...	...
	123	...	...
	124	...	...
	133	...	...
...	...	...	...
blocco n	220	...	...
	234	...	...



OSS: -> SI USA  
SOLO COME  
NOTAZIONE PER  
IL PRIMO BLOCCO

## RICERCA SUL FILE INDICE

PRENDIAMO COME RIFERIMENTO L'ESEMPIO QUI SOPRA

IL BLOCCO 030 DEVE TROVARSI NEL BLOCCO CHE CONTIENE 031 COME VALORE PIÙ PICCOLO DATO CHE NEL PRECEDENTE CHE CONTIENE 003 COME VALORE PIÙ PICCOLO. NEI SUCCESSIVI SONO 101 E

## RICERCA BINARIA

K = CHIAVE DEL FILE PRINCIPALE

SI FA UN ACCESSO AL BLOCCO  $\frac{m}{2} + 1$  E SI CONFRONTA IL VALORE V DEL PRIMO RECORD CON K

- SE  $k = v$  ABBIANO FATTO
- SE  $k < v$  SI RIPETE IL PROCEDIMENTO DA  $B_1$  A  $B_{\frac{m}{2}}$
- SE  $k > v$  SI RIPETE IL PROCEDIMENTO DA  $B_{\frac{m}{2}}$  A  $B_m$

NEL CASO PEGGIORE DOPO  $\log_2(m)$  ACCESSI IN MEMORIA PER LA RICERCA NELL'INDICE + 1 ACCESSO AL BLOCCO TRONATO

# RICERCA PER INTERPOLAZIONE

UTILE QUANDO SI CONOSCE LA DISTRIBUZIONE DELLE CHIAVI

CORSA DI STIMARE LA POSIZIONE DEL VALORE DA CERCARE IN BASE ALLA DISTRIBUZIONE DEI DATI.

FUNZIONE  $F(V_1, V_2, V_3)$

DOVE  $V_1 \rightarrow$  VALORE DA CERCARE

$V_2 \rightarrow$  INIZIO INTERVALLO

$V_3 \rightarrow$  FINE INTERVALLO

CALCOLA LA POSIZIONE DI  $V_1$  NELL'INTERVALLO  $[V_2, V_3]$

RESTITUISCE UN INDICE  $i$  CHE INDICA IN QUALE BLOCCO  $B_i$  POSSEREBBE TROVARSI  $V_1$

## PROCEDURA

DEFINIAMO I BLOCCHI FILE INDEX CON  $B_1, B_2, \dots, B_n \in K$  CORSE CHIAVE DEL RECORD FILE PRINCIPALE DA TROVARE

$$V_1 = k$$

$V_2 =$  CHIAVE DEL PRIMO RECORD CONTENUTO IN  $B_1$   
 $V_3 =$  CHIAVE DEL PRIMO RECORD CONTENUTO IN  $B_n$

## RISULTATO

- SE  $k <$  VALORE CHIAVE DI  $B_i$ , RIPETIAMO IL PROCEDIMENTO SU I BLOCCHI PRECEDENTI  $(B_1, \dots, B_{i-1})$

- SE  $k >$  VALORE CHIAVE  $B_i$ , RIPETIAMO IL PROCEDIMENTO SU BLOCCHI SUCCESSIVI  $(B_i, B_{i+1}, \dots, B_n)$

FINALMENTE QUANDO NON SI RESTRINGE LA RICERCA A UNICO BLOCCO. A QUESTO PUNTO SI SCANSIONA IL BLOCCO PER TROVARE UN VALORE NEI RECORD CHE RICOPRE  $k$

## COSTO

HA UN COSTO IN TERMINI DI ACCESSI A MEMORIA DI  $1 + \log_2(\log_2(u))$ , DOVE  $u$  È IL NUMERO DI BLOCCHI NEL FILE INDICE.

## INSEGNAMENTO

PER FARE UN INSERIMENTO DOBBIANO PAGARE IL COSTO DELLA RICERCA + 1 ACCESSO AL BLOCCO PER SCRIVERE.

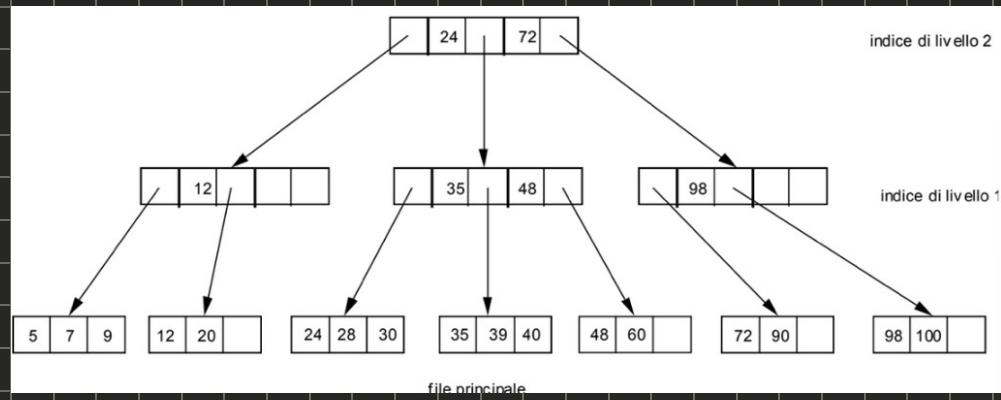
SE NEL BLOCCO NON C'È SPAZIO SEGUONO DIVERSE STRATEGIE E ABBIANO BISOGNO DI PIÙ ACCESSI:

- SE C'È SPAZIO NEL PREC. O SUCC., IL NUOVO RECORD, IL NUOVO RECORD DIVENTA IL PRIMO RECORD DEL BLOCCO SUCCESSIVO E VENGONO MODIFICATI ANCHE IL VALORE NEL FILE INDICE
- SE NON C'È SPAZIO NE NEL PREC. NE NEL SUCC. DOBBIANO RICHIEDERE UN NUOVO BLOCCO CHE VENGHE COLLEGATO ALL'ULTIMO TRA DI LORO

# B-TREE

È UNA GENERALIZZAZIONE DEL FILE INDICE DOVE ACCEDIAMO ATTRAVERSO UNA GERARCHIA DI FILE INDICE.

L'INDICE A LIVELLO PIÙ ALTO VENGUE DETTO RADICE ED È COSTITUITO DA UN SINGOLO BLOCCO, POSSIANO QUINDI CARICARLO IN RAM DURANTE L'UTILIZZO DEL FILE.



OGNI BLOCCO DI UN FILE È COSTITUITO DA RECORD CONTENENTI UNA COPIA (V. b) DOVE V È IL VALORE DELLA CHIAVE DEL PRIMO RECORD DELLA PRIMA PORZIONE DI FILE PRINCIPALE ACCESSIBILE TRAMITE IL PUNTATORE b.

b È IL PUNTATORE CHE PUÒ PUNTARE A UN BLOCCO DEL FILE INDICE A LIVELLO IMMEDIATAMENTE PIÙ BASSO O A UN BLOCCO DEL FILE PRINCIPALE

## RICERCA

PER CERCARE UN RECORD CON UN DATO VALORE, SI ACCEDE GLI INDICI DI LIVELLO PIÙ ALTO E SI SCENDE NEI LIVELLI PIÙ BASSI RESTRINGENDO LA PORZIONE DEL FILE PRINCIPALE IN CUI POTREBBE TROVARSI IL RECORD NELL BLOCCO

QUINDI, SI PARTE DALLA RADICE E SI ESAMINA BLOCCO PER BLOCCO.

SE IL BLOCCO APPARTIENE AL FILE PRINCIPALE, ALLORA QUELLO È IL BLOCCO IN CUI POTREBBE TROVARSI IL RECORD.

SE INVECE È UN BLOCCO DEL FILE INDICE, SI CERCA UN VALORE DELLA CHIAVE CHE RICOPRE CHE STIAMO CERCANDO E Poi SI SEGUO IL PUNTATORE ASSOCIAUTO A QUEL VALORE PROSEGUENDO COSÌ IN UN ALTRO LIVELLO

PER LA RICERCA SONO NECESSARI  $\lceil \log_b n \rceil + 1$  ACCESSI.

la è l'altezza dell'albero

OSS: 

- PIÙ I BLOCCHI SONO PIANI, PIÙ SARÀ PICCOLO  $h$  E QUINDI ALMENO COSTERÀ LA RICERCA.
- PER QUESTO, SI RICHIEDE CHE I BLOCCHI SIANO PIENI ALMENO PER METÀ.
- SE I BLOCCHI SONO COMPLETAMENTE PIENI, UN INSERIMENTO PUÒ RICHIEDERE UNA MODIFICA DELL'INDICE A OGNI LIVELLO E IN ALCUNI CASI, FAR CRESCERE L'ALTEZZA DI UN LIVELLO

Qual è il massimo valore che può assumere  $h$  nel caso peggiore?

## D EFINIAMO

- $N \rightarrow$  NUMERO DI RECORD NEL FILE PRINCIPALE
- $e-1 \rightarrow$  NUMERO DI RECORD DEL FILE PRINCIPALE CHE POSSONO ESSERE MEMORIZZATI IN UN BLOCCO
- $e^d - 1 \rightarrow$  NUMERO DI RECORD DEL FILE INDICE CHE POSSONO ESSERE MEMORIZZATI IN UN BLOCCO

LI RENDIAMO DISPARI PER RENDERE PIÙ SEMPLICI I CALCOLI

DATO CHE VOGLIAHO CHE I BLOCCHI SIANO PIENI ALMENO PER METÀ:

- OGNI BLOCCO DEL FILE PRINCIPALE DEVE CONTENERE ALMENO  $e$  RECORD
- OGNI BLOCCO DEL FILE INDICE DEVE CONTENERE ALMENO  $d$  RECORD

$k$  È L'ALTEZZA MASSIMA DELL'ALBERO E SI HA QUANDO I BLOCCHI SONO PIENI AL MINIMO E QUINDI QUANDO I BLOCCHI HANNO ESATTAMENTE IL NUMERO DI ELEMENTI SCRITTI ( $e$  ELEMENTI PER IL FILE PRINCIPALE E  $d$  PER L'INDICE)

- IL FILE PRINCIPALE HA ESATTAMENTE  $\frac{N}{e}$  BLOCCHI
- AL LIVELLO 1 IL FILE INDICE HA ESATTAMENTE  $\frac{N}{e^d}$  RECORD CHE POSSONO ESSERE MEMORIZZATI IN  $\frac{N}{e^d}$  BLOCCHI
- AL LIVELLO 2 IL FILE INDICE HA ESATTAMENTE  $\frac{N}{e^{d^2}}$  RECORD CHE POSSONO ESSERE MEMORIZZATI IN  $\frac{N}{e^{d^2}}$  BLOCCHI

## GENERALIZZANDO

AUREMO CHE

- AL LIVELLO  $i$  IL FILE INDICE HA  $\frac{N}{e^{d^i}}$  RECORD CHE POSSONO ESSERE MEMORIZZATI IN  $\frac{N}{e^{d^i}}$  BLOCCHI

AL LIVELLO  $K$ , IL FILE INDICE HA ESATTAMENTE 1 BLOCCO

$$\text{QUINDI } \frac{N}{e^{d^K}} \leq 1.$$

POSSIAMO DIRE CHE  $\lceil \frac{N}{e^{d^K}} \rceil = 1$  DATO CHE NELLE DIVISIONI PRENDIAMO LA PARTE SUPERIORE.

RISOLVO L'EGUAGLIAZIONE

$$\frac{N}{e^{d^K}} = 1 \rightarrow \frac{N}{e^{d^K}} \cdot d^K = 1 \cdot d^K \rightarrow d^K = \frac{N}{e}$$

$$K = \log_d \left( \frac{N}{e} \right)$$

QUINDI IL VALORE MASSIMO CHE  $K$  PUÒ ASSUNGERE NEL CASO PEGGIORE È  $\log_d \left( \frac{N}{e} \right)$  → NUMERO BLOCCHI FILE PRINC.

NUM DI RECORD CONTENUTI IN UN BLOCCO NELL FILE PRINCIPALE  
FILE INDICE

## INSERIMENTO

IN QUESTO CASO LA RICERCA COSTA  $h+2$  PERCHÉ ABBIANO  $h+1$  ACCESSI PER LA RICERCA + 1 ACCESSO PER SCRIVERE IL BLOCCO.

QUESTO PERÒ SUCCIDE SOLO SE NEL BLOCCO C'È SPAZIO SUFFICIENTE PER INSERIRE IL RECORD

SE NON C'È SPAZIO NEL BLOCCO ABBIANO  $h+1$  ACCESSI PER LA RICERCA + 2 ACCESSI

$$S \leq 2^{h+1}$$

NEL CASO PEGGIORE PER OGNI LIVELLO DOBBIANO "SDOPPIARE" UN BLOCCO E QUINDI EFFETTUARE DUE ACCESSI PIÙ 1 ALLA FINE PER LA NUOVA RADICE

## CANCELLAZIONE

ABBIATO SEMPRE IN ACCESSI A CUI DOBBIANO AGGIUNGERE  
UN ACCESSO PER RISCRIVERE IL BLOCCO MODIFICATO.  
QUESTO SOLO SE IL BLOCCO RIMANE PIENO PER ALmeno METÀ  
DOPO LA CANCELLAZIONE ALTRIMENTI SONO NECESSARI ULTERIORI  
ACCESSI.

# GESTIONE DELLA CONCORRENZA

IN UN COMPUTER I PROCESSI SONO ESEGUITI IN MODO **INTERLEAVED**, COSÌ VENGONO ALTERNATI TRA DI LORO MA LA CPU PUÒ ESEGUIRE SOLO UNO ALLA VOLTA.

LA CPU PUÒ QUINDI ESEGUIRE UN PROGRAMMA, SOSPENDERLO PER ESEGUIRE ISTRUZIONI DI ALTRI PROGRAMMI E Poi TORNARE SUL PRIMO. QUESTO PERMETTE UN UTILIZZO PIÙ **EFFICIENTE** DELLA CPU.

IN UN DBMS (DATA BASE MANAGEMENT SYSTEM) LA PRINCIPALE RISORSA A CUI ACCEDiamo IN MODO CONCORSANTE (PIÙ UTENTI O APPLICAZIONI POSSONO ACCEDERE E MODIFICARE IL DATA BASE CONTEMPORANEAMENTE) È, APPUNTO, LA BASE DATI. PERO, QUESTO TIPO DI ACCESSO PUÒ CAUSARE PROBLEMI QUANDO EFFETUIAMO DELLE SCRITURE E QUINDI VÀ **CONTROCCOLATO**.

## TRANSAZIONE

È L'ESECUZIONE DI UNA PARTE DI PROGRAMMA È RAPPRESENTATA UN'UNITÀ LOGICA DI ACCESSO O MODIFICA ALLA BASE DI DATI. POSSIAMO, DUNQUE, VEDERLA COME UN'OPERAZIONE SULLA BASE DI DATI.

## PROPRIETÀ TRANSAZIONI

- **ATOMICITÀ**: QUANDO ESEGUIAMO UNA TRANSAZIONE, QUESTA DEVE ESSERE ESEGUITA PER INTERO. SE VENISSE INTERRUOTTA PER QUALCHE MOTIVO, ALLORA DOVRANNO ESSERE ANNULLATE TUTTE LE AZIONI SVOLTE SU DI ESSA.
- **CONSISTENZA**: QUANDO ESEGUIAMO UNA TRANSAZIONE, LE MODIFICHE CHE APPORTA AL DATA BASE NON DEVONO VIOLARE I VINCOLI DI INTEGRITÀ. QUINDI, NON CI DEVONO ESSERE CONTRADDIZIONI TRA I DATI ARCHIVIATI NEL DB.
- **ISOLAMENTO**: OGNI TRANSAZIONE DEVE ESSERE ESEGUITA IN MODO ISOLATO E INDIPENDENTE DALLE ALTRE. L'EVENTUALE FALLIMENTO DI UNA NON DEVE INTERFERIRE CON LE ALTRE.
- **DURABILITÀ (O PERSISTENZA)**: I RISULTATI DI UNA TRANSAZIONE SCRITTI SU UNA BASE NON DEVONO ESSERE PERSI. PER ASSICURARSI CHE QUESTO NON ACCADA, VENGONO SCRITTI DEI **LOG** SU TUTTE LE OPERAZIONI ESEGUITE SUL DB.

## SCHEDULE

È UN ORDINAMENTO DI UN INSIEME  $T$  DI TRANSAZIONI CHE CONSERVA L'ORDINE CHE LE OPERAZIONI HANNO ALL'INTERNO DELLE SINGOLE TRANSAZIONI.

SE L'OPERAZIONE  $o_1$  PRECEDI L'OPERAZIONE  $o_2$  IN UNA TRANSAZIONE, ALLORA SARÀ COSÌ IN OGNI SCHEDULE DOVE COMPARIRÀ QUELLA TRANSAZIONE.

## SCHEDULE SERIALE

È UNO SCHEDULE CHE SI OTIENE PERMUTANDO LE TRANSAZIONI IN  $T$ . QUESTO CORRISPONDE A UN'ESECUZIONE SEQUENZIALE DELLE TRANSAZIONI. INFATTI, NON VENGONO INTERROTTE E VENGONO ESEGUITE UNA PER VOLTA PER INTERO.

## PROBLEMI DELL'ESECUZIONE CONCORRENTE

CONSIDERIAMO LE DUE TRANSAZIONI  $T_1$  E  $T_2$

$T_1$	$T_2$
$\text{read}(X)$	$\text{read}(X)$
$X := X - N$	$X := X + M$
$\text{write}(X)$	$\text{write}(X)$
$\text{read}(Y)$	
$Y := Y + N$	
$\text{write}(Y)$	

ORA CONSIDERIAMO IL SEGUENTE SCHEDULE

$T_1$	$T_2$
$\text{read}(X) \xrightarrow{X}$	
$X := X - N$ $X = X_0 - N$	$\text{LEGGI IL VALORE AGGIORNATO DI } X \text{ DOPPIO LEGGE } X$
$\text{SCRIVE } X = X_0 - N$	$\text{read}(X)$
$\text{write}(X)$	$X := X + M$ $X = X_0 + M$
$\text{read}(Y)$	
	$\text{write}(X)$
	$\text{SCRIVE } X = X_0 + M$
$Y := Y + N$	
$\text{write}(Y)$	

IN QUESTO CASO ABBIANO UN **LOST UPDATE**, COÉ UN AGGIORNAMENTO PERDUTO.

SE IL VALORE DI  $X$  È  $X_0$ , AL TERMINE DELLESECUCIONE INVECCE DI ESSERE  $(X_0 - N) + M$ , SARÀ  $X_0 + M$ .

QUESTO SUCCIDE PERCHE  $T_2$  LEGGE IL VALORE DI  $X$  PRIMA CHE VENGA AGGIORNATO IN  $T_1$

# CONSIDERIAMO ORA QUESTO SCHEDULE

$T_1$	$T_2$
$read(X)$	
$X := X - N$	
$write(X)$	
	$read(X)$
	$X := X + M$
$read(Y)$	
$T_1 \text{ fallisce}$	
	$write(X)$

IN QUESTO CASO SI PARLA DI DATO SPORCO.

I DATI SONO AGGIORNATI CORRETTAMENTE, MA LA TRANSAZIONE  $T_1$  FALLISCE.

## ALTRÒ ESEMPIO

$T_1$	$T_3$
	$somma := 0$
$read(X)$	
$X := X - N$	
$write(X)$	
	$read(X)$
	$somma := somma + X$
	$\text{read}(Y)$
	$somma := somma + Y$
$read(Y)$	
$Y := Y + N$	
$write(Y)$	

QUI L'ERRORE PRENDE IL NOME DI AGGREGATO NON CORRETTO.

SI HA QUANDO UNA PARTE DEI DATI CHE ABBIANO UTILIZZATO NON È CORRETA.

AL TERMINE DELL'ESECUZIONE DELLO SCHEDULE È  $X_0 - N + Y_0$  INVECE DI  $X_0 - Y_0$ .

QUELLO CHE VOGLIAMO ERA  $SOMMA = X_0 - N + Y_0 + N$ , MA  $Y$  VIDE AGGIORNATO DOPO

## IN GENERALE

DEFINIAMO UNO SCHEDULE ERRORE QUANDO I VALORI PRODOTTI NON SONO QUELLI CHE SI AVREBBERO SE LE TRANSAZIONI FOSSERO ESEGUITE IN modo SEQUENZIALE.

## SERIALIZZABILITÀ

TUTTI GLI SCHEDULE SERIALI SONO CORRETTI  
UNO SCHEDULE NON SERIALE È CORRETTO SE È SERIALIZZABILE,  
CIOÈ SE È EQUIVALENTE A UNO SCHEDULE SERIALE

ATTENZIONE → EQUIVALENTE SIGNIFICA CHE DUE SCHEDULE PRODUcono  
RISULTATI UGUALI

MA NON BASTA!!!

$T_1$	$T_2$
read( $X$ )	
	read( $X$ )
$X := X + 5$	
	$X := X * 1.5$
	write( $X$ )
write( $X$ )	

$T_1$	$T_2$
read( $X$ )	
	read( $X$ )
$X := X + 5$	
	$X := X * 1.5$
$X := X + 5$	
write( $X$ )	
	write( $X$ )

QUESTI SCHEDULE SONO EQUIVALENti.  
SOLO SE IL VALORE INIZIALE DI  $X$  È 10  
NEGLI ALTR. CASI NO

QUINDI

EQUIVALENTE SIGNIFICA CHE DUE SCHEDULE PRODUcono  
RISULTATI UGUALI E DUE VALORI SONO EQUIVALENti SOLO SE  
PRODOTTI DALLA STESSA SEQUENZA DI OPERAZIONI

TESTARE LA SERIALIZZABILITÀ

È MOLTO DIFFICILE PERCHÉ:

- È DIFFICILE STABILIRE QUANDO UNO SCHEDULE COMINCIA O FINISCE
- È IMPOSSIBILE DETERMINARE IN ANTICIPO IN CHE ORDINE VENGONO ESEGUITE LE OPERAZIONI DATO CHE DIPENDE DAL CARICO DEL SISTEMA.
- SE ESEGUISCO UNO SCHEDULE E Poi TESTANDO LA SERIALIZZABILITÀ NOTIAGO CHE NON È SERIALIZZABILE, I SUO EFFETTI DEVONO ESSERE ANNULLATI.

QUINDI, INVECE DI TESTARE, USIAMO DEI METODI CHE GARANTISCONO.

- O SI IMPOGGONO DEI PROTOCOLLI ALLE TRANSAZIONI
- O SI USANO I TIMESTAMP DELLE TRANSAZIONI (IDENTIFICATORI GENERATI DAL SISTEMA E USATI PER ORDINARE LE TRANSAZIONI E GARANTIRE SERIALIZZABILITÀ)



## ITEM

ENTRAMBI I METODI SOPRA ELENCATI FANNO USO DEGLI ITEM, CIOÈ UNITÀ DEL DB DI CUI CONTROLLIAMO L'ACCESSO

LE DIMENSIONI DEGLI ITEM VARIANO IN BASE ALL'USO CHE VIENE FATTO DEL DB IN MODO TALE CHE IN MEDIA UNA TRANSAZIONE ACCEDA A POCHI ITEM

GRANULARITÀ → È LA DIMENSIONE DELL'ITEM USATA DAL SISTEMA

- UNA GRANDE GRANULARITÀ PERMETTE UNA GESTIONE EFFICIENTE DELLA CONCORRENZA
- UNA PICCOLA GRANULARITÀ CONSENTE L'ESECUZIONE CONCORSIVA DI MOLTE TRANSAZIONI, MA SOVRACCARICA IL SISTEMA

## MECANISMO DI LOCK

SI REALIZZA TRAMITE UNA VARIABILE ASSOCIATA A OGNI ITEM CHE NE DESCRIVE LO STATO RISPETTO ALLE OPERAZIONI CHE POSSONO ESSERE ESEGUITE SU DI ESSO.

### LOCK BINARIO

LA VARIABILE ASSUME 2 VALORI

- LOCKED: L'ITEM È IN USO DA UNA TRANSAZIONE E NON PUÒ ESSERE UTILIZZATO DA ALTRE
- UNLOCKED: È POSSIBILE USARE L'ITEM PER SVOLGERE OPERAZIONI SU DI ESSO

SE UNA TRANSAZIONE VUOLE USARE UN ITEM, CONTROLLA CHE SIA IN STATO UNLOCKED. IN TAL CASO, LO PORTA IN STATO LOCKED ATTRAVERSO UN'OPERAZIONE DI LOCKING. QUANDO AVRÀ FINITO DI USARLO, LO RILASCIÀ CON UN'OPERAZIONE DI UNLOCKING, COSÌ POTRÀ ESSERE USATO DA ALTRE TRANSAZIONI

## SCHEDULE LEGALE

UNO SCHEDULE SI DICE LEGALE SE UNA TRANSAZIONE, OGNI VOLTA CHE DEVO LEGGERE O SCRIVERE UN ITEM EFFETTUO UN LOCKING E Poi RILASCIAT OGNI LOCK FATTO.

ABBIAMO VISTO CHE IL LOCK BINARIO ASSUME 2 VALORI:

- **LOCK (x)** PER RICHIEDERE L'ACCESSO A UN ITEM X
- **UNLOCK (x)** PER RILASCIARE L'ITEM X

L'INSIEME DEGLI ITEM LETTI E SCRITTI DA UNA TRANSAZIONE COINCIDONO

VANTAGGI LOCK BINARIO → RISOLVE IL PROBLEMA DEL **LOST UPDATE**

## EQUIVALENZA

LA PROPRIETÀ DI EQUIVALENZA DEGLI SCHEDULE CAMBIA IN BASE AL PROTOCOLLO DI LOCKING USATO

INNANZITUTTO, ADOTTIAMO UN MODELLO PER LE TRANSAZIONI CHE CONSIDERA SOLO LE OPERAZIONI RILEVANTI (IN QUESTO CASO **LOCK** E **UNLOCK**)

$T_1$
$lock(X)$
$unlock(X)$
$lock(Y)$
$unlock(Y)$

- QUINDI UNA TRANSAZIONE DIVENTA UN'OPERAZIONE DI **LOCK** E **UNLOCK**.
  - OGNI LOCK IMPLICA UNA LETTURA DI X E OGNI UNLOCK UNA SCRITURA DI X

PER VERIFICARE UN'EQUIVALENZA ANDIAMO AD ASSOCIARE A OGNI **UNLOCK** UNA FUNZIONE CHE PRENDE IN INPUT TUTTI GLI ITEM LETTI DALLA TRANSAZIONE PRIMA DI QUELLO **UNLOCK**

$T_1$
$lock(X)$
$unlock(X) f_1(X)$
$lock(Y)$
$unlock(Y) f_2(X, Y)$

DICIAMO CHE 2 SCHEDULE SONO EQUIVALENTI QUANDO LE FORMULE CHE DANNO I VALORI FINALI PER CIASCUN ITEM SONO LE STESSE

# CONSIDERIAMO LE TRANSAZIONI

$T_1$	$T_2$
$lock(X)$	$lock(Y)$
$unlock(X) f_1(X)$	$unlock(Y) f_3(Y)$
$lock(Y)$	$lock(X)$
$unlock(Y) f_2(X, Y)$	$unlock(X) f_4(X, Y)$

È IL SEGUENTE SCHEDULE

	$T_1$	$T_2$
legge $X_0$	$lock(X)$	
scrive $f_1(X_0)$	$unlock(X)$	
legge $f_3(Y_0)$		$lock(Y)$
scrive $f_2(X_0, f_3(Y_0))$		$unlock(Y)$
SCRIVE $X_0$ PERCHÉ È L'ULTIMO VALORE DI X LETTO IN T2		
	$lock(X)$	
	$unlock(X)$	$lock(Y)$
		$unlock(Y)$
		$lock(X)$
		$unlock(X)$

IL VALORE FINALE DI X È  $f_4(f_1(X_0), Y_0)$

PER CAPIRE SE QUESTO SCHEDULE È SERIALE E ABILE CONFRONTANDO I VALORI FINALI CON GLI SCHEDULE SERIALI

	$T_1$	$T_2$
legge $X_0$	$lock(X) x_0$	
scrive $f_1(X_0)$	$unlock(X) f_1(x_0)$	
legge $Y_0$		$lock(Y) y_0$
scrive $f_2(X_0, Y_0)$		$unlock(Y) f_2(x_0, y_0)$
		$lock(Y) f_2(x_0, y_0)$
		$unlock(Y) f_3(f_2(x_0, y_0))$
		$lock(X) f_3(x_0, y_0)$
		$unlock(X)$
		$lock(X)$
		$unlock(X)$

IL VALORE FINALE DI X È  $f_4(f_1(x_0), f_3(f_2(x_0, y_0)))$

NON È EQUIVALENTE CON IL VALORE OTENUTO NELL'ALTRO SCHEDULE

QUINDI CONTROLLIAMO CON L'ALTRÒ SCHEDULE SERIALE

	$T_1$	$T_2$
legge $f_4(X_0, Y_0)$		$lock(Y) y_0$
scrive $f_1(f_4(X_0, Y_0))$		$unlock(Y) f_3(y_0)$
legge $f_3(Y_0)$		$lock(X) x_0$
scrive $f_2(f_4(X_0, Y_0), f_3(Y_0))$		$unlock(X) f_2(f_4(x_0, y_0), f_3(y_0))$
		$lock(Y) f_2(f_4(x_0, y_0), f_3(y_0))$
		$unlock(Y) f_3(f_2(f_4(x_0, y_0), f_3(y_0)))$

IL VALORE FINALE DI X È  $f_4(f_1(x_0, y_0))$

NON È EQUIVALENTE

DATO CHE  $f_a(f_1(x_0)f_3(f_2(x_0, y_0))) \in f_1(f_a(x_0, y_0))$  NON SONO EQUIVALENTI A  $f_a(f_1(x_0), y_0)$  ALLORA LO SCHEDULE NON È SERIALIZZABILE

SE IL VALORE DI X DEL PRIMO SCHEDULE FOSSE STATO EQUIVALENTE AD ALMENO UNO DEI DUE SCHEDULE SERIALI, AUREI DOVUTO CONTROLLARE IL VALORE DI Y. SE ANCHE Y FOSSE STATO EQUIVALENTE, ALLORA LO SCHEDULE INIZIALE SAREBBE STATO SERIALIZZABILE

QUINDI UNO SCHEDULE SERIALIZZABILE SE ESISTE UNO SCHEDULE SERIALE TALE CHE PER OGNI ITEM, L'ORDINE IN CUI LE VARIE TRANSAZIONI FANNO UN LOCK SU QUELLO ITEM CONCIDE CON QUELLO DELLO SCHEDULE SERIALE

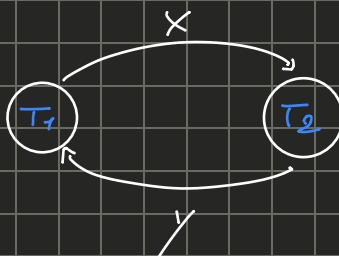
### ALGORITMO 1 - TESTARE SERIALIZZABILITÀ

DATO UNO SCHEDULE S CREA UNO GRAFO DIRETTO CHIAMATO GRAFO DI SERIALIZZABILITÀ DOVE:

- I NODI SONO LE TRANSAZIONI
- GLI ARCHI VANNO DA UNA TRANSAZIONE  $T_i$  AD UNA TRANSAZIONE  $T_j$  E HANNO ETICHETTA X SE IN S LA TRANSAZIONE  $T_i$  ESEGUE UN  $UNLOCK(X)$  E  $T_j$  ESEGUE IL SUCCESSIVO  $LOCK(X)$

$T_1$	$T_2$
$lock(X)$	
$unlock(X)$	
	$lock(Y)$
	$unlock(Y)$
$lock(Y)$	
$unlock(Y)$	
	$lock(X)$
	$unlock(X)$

PER ESEMPIO QUESTO SCHEDULE AVRA' CORE GRAFO



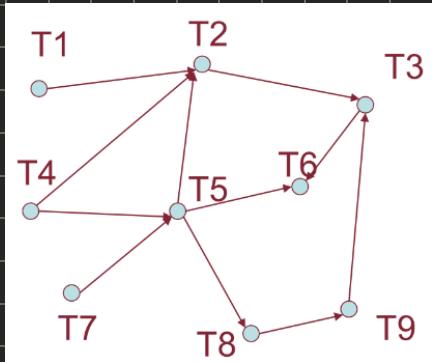
LO STEP SUCCESSIVO È CONTROLLARE SE IL GRAFO HA CICLI, IN TAL CASO NON È SERIALIZZABILE.

IN CASO CONTRARIO, SI USA IL SORT TOPOLOGICO PER OTENERE LO SCHEDULE SERIALE S' EQUIVALENTE A S

## SORT TOPOLOGICO

SI OTTIENE ELIMINANDO LE SORGENTI INSIEME AI SUOI ARCHI USCENTI

UN GRAFO PUÒ AVERE PIÙ SORT TOPOLOGICI



UN POSSIBILE SORT DI QUESTO GRAFO È

T<sub>1</sub> T<sub>4</sub> T<sub>2</sub> T<sub>5</sub> T<sub>2</sub> T<sub>8</sub> T<sub>9</sub> T<sub>3</sub> T<sub>6</sub>

## PROTOCOLLO DI LOCKING A 2 FASI

UNA TRANSAZIONE SI DICE A 2 FASI SE

- PRIMA EFFETTUO TUTTE LE LOCK
- Poi tutte le UNLOCK

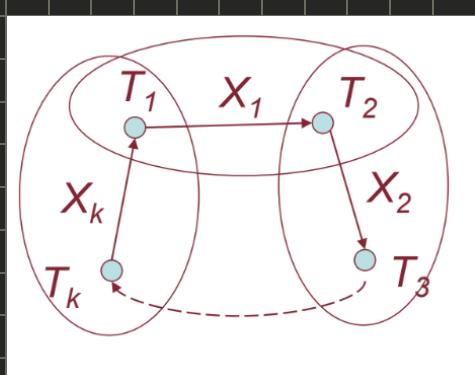
Dopo un UNLOCK NON È PIÙ POSSIBILE EFFETTUARE LOCK

## TEOREMA LOCK A 2 FASI

SIÀ  $T$  UN INSIEME DI TRANSAZIONI, SE OGNI TRANSAZIONE IN  $T$  È A 2 FASI ALLORA OGNI SCHEDULE IN  $T$  È SERIALIZZABILE

## DISOSTRAZIONE PER ASSURDO

ABBIAKNO OGNI TRANSAZIONE IN  $S$  A 2 FASI, MA NEL GRAFO ABBIAKNO UN CICLO



NEL NOSTRO SCHEDULE AVREMO

- $T_1 \text{ UNLOCK}(X_1)$
- ...
- $T_2 \text{ LOCK}(x_1)$
- ...
- $T_2 \text{ UNLOCK}(x_2)$
- ...
- $T_3 \text{ LOCK}(x_2)$
- ...
- $T_k \text{ UNLOCK}(x_k)$
- $T_r \text{ LOCK}(x_k)$

DÀ NOTARE CHE PER CREARE IL CICLO ABBRACCIO BISOGNA CHE TI ESEGUA UN LOCK SU X<sub>i</sub>, MA QUESTO NON È POSSIBILE PERCHÉ TI È A 2 FASI E QUINDI DOPO LA PRIMA UNLOCK NON È PIÙ POSSIBILE FAR E LOCK IN T<sub>Y</sub>

## DEADLOCK E LIVELOCK

### DEADLOCK

SI VERIFICA QUANDO OGNI TRANSAZIONE IN UN INSIEME T È IN ATTESA DI OTTENERE UN LOCK SU UN ITEM SOL QUALE UN'ALTRA TRANSAZIONE NELLO STESSO INSIEME MANTIENE UN LOCK.

QUINDI, LA TRANSAZIONE RIMANESCE BLOCCATA NON RILASCIANDO IL LOCK E BLOCCANDO ANCHE EVENTUALI TRANSAZIONI NELL'INSIEME

ESSENZIALMENTE SI HA QUESTO:

- UNA TRANSAZ. A STA ASPETTANDO L'UNLOCK DI B
- B STA ASPETTANDO L'UNLOCK DI C
- C STA ASPETTANDO L'UNLOCK DI A

QUINDI È TUTTO BLOCCATO

### CHE SI RISOLVE?

INNANZITUTTO, USIAMO IL GRAFO DI ATTESA PER VERIFICARE UNA SITUAZIONE DI STALLO

- NODI → TRANSAZIONI
- ARCO T<sub>i</sub> → T<sub>j</sub> SE LA TRANSAZ. T<sub>i</sub> È IN ATTESA DI UN LOCK SU UN ITEM SUL QUALE T<sub>j</sub> MANTIENE UN LOCK

SE NEL GRAFO CI È UN CICLO SI HA UNA SITUAZIONE DI STALLO

### ROLL-BACK

SERVE PER RISOLVERE IL DEADLOCK.

3 FASI

1. ABORTIRE LA TRANSAZIONE
2. ANNULLARE I SUOI EFFETTI SUL DB, RIPRISTINANDO I VALORI DEI DATI PRECEDENTI ALL'INIZIO DELLA TRANSAZIONE
3. TUTTI I LOCK MANTENUTI DALLA TRANSAZIONE VENGONO RILASCIATI

## LIVELOCK

SI VERIFICA QUANDO UNA TRANSAZIONE ASPETTA INDEFINITIVAMENTE CHE GLI VENGA CONCESSO UN LOCK SU UN CERTO ITEM

## COME SI RISOLVE?

- FIRST COME - FIRST SERVED

- OPPURE ESEGUGENDO TRANSAZIONI IN BASE ALLA LORO PRIORITÀ E AUTORIZZANDO LA GROPPIA IN BASE AL TEMPO IN CUI LA TRANSAZIONE È IN ATTESA

## ABORT DI UNA TRANSAZIONE

4 CASI:

1. LA TRANSAZIONE ESEGUE UN'OPERAZIONE NON CORRETTA (DIVISIONE PER 0)
2. LO SCHEDULER RICEVE UN DEADLOCK
3. LO SCHEDULER FA ABORTIRE LA TRANSAZIONE PER GARANTIRE SERIALIZZABILITÀ (TIESTAMP)
4. SI VERIFICA UN MALFUNZIONAMENTO HARDWARE O SOFTWARE

## PUNTO DI COMMIT

È IL PUNTO IN CUI LA TRANSAZIONE:

- HA OTTENUTO TUTTI I LOCK CHE GLI SONO NECESSARI
- HA EFFETTUATO TUTTI I CALCOLI NELL'ARIA DI LAVORO

## DATI SPORCHI

DATI SCRITTI DA UNA TRANSAZIONE SULLA BASE DI DATI PRIMA CHE ABBIA RAGGIUNTO IL PUNTO DI COMMIT

## ROLL-BACK A CASCATA

QUANDO UNA TRANSAZIONE È ABORTITA DEVONO ESSERE ANNULLATI ANCHE I SUOI EFFETTI SUL DB DA TUTTE LE TRANSAZIONI CHE HANNO LETTO DATI SPORCHI.

## PROTOCOLLI A DUE FASI STRETTO

UNA TRANSAZIONE SI DICE A 2 FASI STRETTO SE:

- NON SCRIVE SUL DB PRIMA DEL PRIMO PUNTO DI COMMIT
- NON RILASCI UN LOCK FINCHÉ NON È FINITO DI SCRIVERE TUTTI GLI ITEM SULLA BASE

## CLASSIFICAZIONE DEI PROTOCOLLI

- CONSERVATIVI : CERCANO DI EVITARE LA SITUAZIONE DI STACCO.  

- AGGRESSIVI : CERCANO DI ESEGUIRE LE TRANSAZIONI NEL MODO PIÙ VELOCE ANCHE CAUSANDO STACCI.  


### CONSERVATIVI

UNA TRANSAZIONE T RICHIEDE TUTTI I LOCK CHE SERVONO ALL'INIZIO E LI OTTIENE SE E SOLO SE TUTTI I LOCK SONO DISPONIBILI, ALTRIMENTI LA TRANSAZIONE VENGONO MESSE IN CODA.

IN QUESTO MODO EVITIAMO I DEADLOCK MA NON I LIVELOCK UTILIZZIAMO QUESTO TIPO DI PROTOCOLLO QUANDO Abbiamo un ALTO RISCHIO DI DEADLOCK.

PER EVITARE IL LIVELOCK T RICHIEDE TUTTI I LOCK E LI OTTIENE SE E SOLO SE:

- TUTTI I LOCK SONO DISPONIBILI
- NESSUNA TRANSAZIONE CHE PRECEDI T NELLA CODA È IN ATTESA DI UN LOCK RICHIESTO DA T

VANTAGGI : 

EVITA DEADLOCK E LIVELOCK

Svantaggi : 

TIPO ME

- L'ESECUZIONE DI UNA TRANSAZIONE PUÒ ESSERE RITARDATA
- UNA TRANSAZIONE È COSTRETTA A RICHIEDERE UN LOCK SU OGNI ITEM CHE POTREBBE SERVIRE ANCHE SE Poi NON LO USERA'

### AGGRESSIVI

UNA TRANSAZIONE DEVE RICHIEDERE UN LOCK SU UN ITEM IMMEDIATAMENTE PRIMA DI LEGGERLO O SCRIVERLO

IN QUESTO MODO, PERO', SI POSSONO CAUSARE DEADLOCK

IN BASE A COSA SCEGLIATO quale PROTOCOLLO USARE?

IN BASE ALLA PROBABILITÀ CHE 2 TRANSAZIONI RICHIEDANO UN LOCK SULLO STESSO ITEM

SE È:

- ALTA: SI UTILIZZANO I CONSERVATIVI PER EVITARE IL SOVRACCARICO DI SISTEMA PER GESTIRE I DEADLOCK
- BASSA: SI UTILIZZANO GLI AGGRESSIVI PER EVITARE IL SOVRACCARICO PER GESTIRE I LOCK

## TIME-STAMP

È UN VALORE SEQUENZIALE E CRESCENTE CHE IDENTIFICA UNA TRANSAZIONE.  
È ASSEGNAUTO ALLA TRANSAZIONE NEL MOMENTO IN CUI VIENE CREATATA  
E PUÒ ASSUNGERE DIVERSI VALORI COME UN CONTATORE  
O L'ORA DI CREAZIONE

PIÙ IL VALORE È ALTO, MENO È IL TEMPO IN CUI LA TRANSAZIONE SI TROVA NEL SISTEMA

SE  $T_1$  HA TIMESTAMP PIÙ BASSO RISPETTO A  $T_2$ , ALLORA  $T_1$  VERRÀ ESEGUITA PRIMA DI  $T_2$

## SERIALIZZABILITÀ

UNO SCHEDULE È SERIALIZZABILE SE È EQUIVALENTE ALLO SCHEDULE SERIALE DOVE LE TRANSAZIONI COMPIONO ORDINATE IN BASE AI LORO TIMESTAMP

$T_1$	$T_2$
read(X)	
$X := X + 10$	
write(X)	

$T_1$  HA  $TS = 100$   
 $T_2$  HA  $TS = 200$

QUINDI  $T_2$  VIENE ESEGUITA PRIMA DI  $T_1$

$T_1$	$T_2$
	read( $X$ )
read( $X$ )	
	$X := X + 5$
$X := X + 10$	
write( $X$ )	
	write( $X$ )

NON È SERIAZIONE PER CHE  
 $T_1$  LEGGE IL VALORE DI  $X$   
 PRIMA DELLA MODIFICA DI  $T_2$

$T_1$	$T_2$
read( $Y$ )	read( $Y$ )
$X := Y + 10$	$X := Y + 5$
write( $X$ )	write( $X$ )

$T_1$  HA TS = 140  
 $T_2$  HA TS = 100

QUINDI  $T_2$  VIENE ESEGUITA PRIMA DI  $T_1$

$T_1$	$T_2$
	read( $Y$ )
	$X := Y + 5$
	write( $X$ )
read( $Y$ )	
$X := Y + 10$	
write( $X$ )	

QUI IL VALORE FINALE DI  
 $X$  VENDE SCRITTO DA  $T_1$

$T_1$	$T_2$
	read( $Y$ )
read( $Y$ )	
	$X := Y + 5$
$X := Y + 10$	
write( $X$ )	
	write( $X$ )

QUI LO SCHEDEDOLE NON È SERIAZIONE.  
 SOLO SE NON VIENE ESEGUITA  
 LA SCRITTURA DI  $X$  DA PARTE DI  
 $T_2$

## READ TIMESTAMP E WRITE TIMESTAMP

A CIASCUN ITEM X VENGONO ASSOCIATI 2 TIMESTAMP

- READ TIMESTAMP DI X ( $\text{READ-TS}(x)$ ) ED È IL PIÙ GRANDE TRA TUTTI I TIMESTAMP CHE HANNO LETTO X CON SUCCESSO
- WRITE TIMESTAMP DI X ( $\text{WRITE-TS}(x)$ ) ED È IL PIÙ GRANDE TRA TUTTI I TIMESTAMP CHE HANNO SCRITTO X CON SUCCESSO

CHE SI USANO?

OGNI VOLTA CHE T CERCA DI FARE UN READ(x) O UN WRITE(x) CONFRONTA IL TIMESTAMP DELLA TRANSAZIONE CON IL READ E WRITER TIMESTAMP DELL'ITEM PER ASSICURARCI CHE NON CI SIANO VIOLAZIONI NELL'ORDINE STABILITO DEI TIMESTAMP

NELLO SPECIFICO:

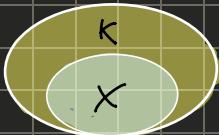
- T VUOLE EFFETTUARE UNA WRITE(x):
  - SE  $\text{READ-TS} > \text{TS}(\tau)$  ALLORA T VIENE ROLLED BACK PERCHÉ QUALCUNO DI PIÙ "Giovane" HA GIÀ LETTO IL DATO
  - SE  $\text{WRITE-TS}(x) > \text{TS}_T(\tau)$  L'OPERAZ. DI SCRITURA NON VERRÀ ESEGUITA
  - ALTRIMENTI, POSSIAMO ESEGUIRE IL WRITE E IMPOSTARE IL WRITE-TS DELL'ITEM = A QUELLO DELLA TRANSAZ.
- T VUOLE ESEGUIRE UNA READ(x):
  - SE  $\text{WRITE-TS}(x) > \text{TS}(\tau)$  ALLORA T VIENE ROLLED BACK, QUESTO SIGNIFICA CHE QUALCUNO ARRIVATO DOPO T HA GIÀ SCRITO IL DATO
  - SE  $\text{WRITE-TS}(x) > \text{TS}(\tau)$  ALLORA LA READ È ESEGUITA E SE  $\text{READ-TS} < \text{TS}(\tau)$  IMPOSTAMO IL READSTAMP DELL'ITEM UGUALE A QUELLO DELLA TRANSAZIONE.

# ALGEBRA RELAZIONALE (NON TUTTA)

## DIP PARZIALE

DATO R, DATO F  $\in X \rightarrow A \in F^+$  CON A  $\neq X$   
LA DIP È PARZIALE SE:

- A NON È PRIMO
- X È CONTENUTO PROPRIMENTE IN UNA CHIAVE



## DIP TRANSITIVA

DATO R, DATO F  $\in X \rightarrow A \in F^+$  CON A  $\neq X$   
LA DIP È TRANSITIVA SE:

- A NON È PRIMO
- PER OGNI CHIAVE K SI HA CHE X NON È CONTENUTO INTERAMENTE IN K E A SUA VOLTA X NON CONTIENE K. CIOÈ:
  - X NON È CONTENUTO IN K
  - $K - X \neq \emptyset$  (X NON CONTIENE ALLEVA CHIAVE)



## TEOREMA 3NF

DATO R, DATO F  
R È IN 3NF SE E SOLO SE NO DIP PARZ, NO DIP TRANSIT. IN R

## DIP

SE

SE NO DIP PARZ O TRANS, TUTTE LE DIP  $X \rightarrow A$  HANNO X SUPERCHIAVE O CHIAVE VERIFICANDO SEMPRE LA COND DI X SUPER.

SOLO SE: 3NF  $\rightarrow$  NO PARZ NO TRANS

DATO  $X \rightarrow A$ , O A È PRIMO O X È SUPERC.

- SE A È PRIMO, ALLORA VIENE A MANICA LA DEF DI TRANSIT E PARZ.
- SE X È SUPERC. ALLORA ABBIANO 2 SITUAZIONI:

- NON CI SONO DIP PARZIALI PERCHE' X DOVREBBE ESSERE CONTENUTO IN UNA CHIAVE
- NON CI SONO DIP TRANSIT. PERCHE' X NON DOVREBBE CONTENERE CHIAVI

QUINDI  $\exists$  N.F.  $\Leftrightarrow$  NO PARZ. NO TRANSIT



### ASSIOMI DI ARMSTRONG

DATO  $F, F^*$  E' L'INSIEME DI DIP FUNZ CHE SI OTTENG.  
NEI SEGUENTI ASSIOMI:

- RIFLESSIVITA': SE  $y \subseteq x \subseteq R$  ALLORA  $x \rightarrow y \in F^*$
- AUMENTO: SE  $x \rightarrow y \in F^*$  ALLORA  $xz \rightarrow yz \in F^*$  PER OGNI  $z \subseteq R$
- TRANSITIVITA': SE  $x \rightarrow y \in F^*$  E  $y \rightarrow z \in F^*$  ALLORA  $x \rightarrow z \in F^*$

### REGOLE DERIVATE DAGLI ASSIOMI

- UNIONE: SE  $x \rightarrow y \in F^*$  E  $x \rightarrow z \in F^*$  ALLORA  $x \rightarrow yz \in F^*$

#### DIM

- PER AUMENTO SI HA CHE  $x \rightarrow xy \in F^*$
- ANALOGAMENTE, SE  $x \rightarrow z \in F^*$ , ALLORA  $xy \rightarrow yz \in F^*$
- PER TRANSITIVITA', SE  $x \rightarrow xy$ ,  $xy \rightarrow yz$  ALLORA  $x \rightarrow yz \in F^*$

- DECOMPOSIZIONE: SE  $x \rightarrow y \in F^*$  E  $z \subseteq y$  ALLORA  $x \rightarrow z \in F^*$

#### DIM

- SE  $z \subseteq y$  ALLORA  $y \rightarrow z \in F^*$  PER RIFLESSIVITA'
- DATO CHE  $x \rightarrow y \in y \rightarrow z$ , ALLORA  $x \rightarrow z$  PER TRANSITIVITA'

- PSEUDOTRANSITIVITA': SE  $x \rightarrow y \in F^*$  E  $wy \rightarrow z$  ALLORA  $wx \rightarrow z \in F^*$

#### DIM

- SE  $x \rightarrow y \in F^*$ , ALLORA  $wx \rightarrow wy \in F^*$  PER AUMENTO
- DATO CHE  $wx \rightarrow wy \in wy \rightarrow z$ , ALLORA  $wx \rightarrow z$  PER TRANSITIVITA'

## DEF $X^+_F$

DATO  $R$ , DATO  $F \subseteq R$  UN SOTTOINSIEME  $X$  DI  $R$  CHIAMATO  $X$ , DICHIARO CHE LA CHIUSURA DI  $X$  RISPETTO A  $R$  ( $X^+_F$ ) È DEFINITA COME:

$$X^+_F = \{A : X \rightarrow A \in F^A\}$$

CIOÈ TUTTI GLI ATTRIBUTI DETERMINATI DA  $X$

## LEMMA 1

DATO  $R$ , DATO  $F$  SI HA CHE  $X \rightarrow y \in F^A$  SE E SOLO SE  $y \subseteq X^+$

DIM

$$\text{SIA } y = A_1 A_2 \dots A_m$$

## PARTE SE

DATO CHE  $X \rightarrow y \in F^A$ , PER DECOMPOSIZIONE AVREMO CHE PER OGNI  $i = 1, 2, \dots, m$   $X \rightarrow A_i \in F^A$  E QUESTO SIGNIFICA CHE  $A_i \in X^+$ , DOVE  $A_i$  SONO TUTTI GLI ATTRIBUTI IN  $y \in$  QUINDI  $y \subseteq X^+$

## PARTE SOLO SE $y \subseteq X^+ \rightarrow (X \rightarrow y)$

DATO CHE  $y \subseteq X^+ \forall i = 1, 2, \dots, m$  AVREMO CHE  $X \rightarrow A_i \in F^A$  PERTANTO  $X \rightarrow y \in F^A$

## TEOREMA $F^+ = F^A$

DATO  $R$ , DATO  $F$  SI HA CHE  $F^+ = F^A$

DIM

PER DIMOSTRARLO, DIMOSTRO CHE  $F^A \subseteq F^+$  E CHE  $F^+ \subseteq F^A$

$$F^A \subseteq F^+$$

DIMOSTRO PER IND SUL NUMERO DI APPLICAZIONI DEGLI ASSIOMI DI ARMSTRONG

## CASO BASE

$i=0$  QUINDI NON ABBIAMO APPLICATO ALCUN ASSIOMA QUINDI  $F^A$  CONTIENE  $F$  E  $F^+$  CONTIENE ELEMENTI DI  $F$

# PROTESSI INDUTTIVA

Ogni dip funz ottenuta da  $F$  applicando  $A$  a  $s$ .  
Un numero minore o uguale a  $i-j$  è in  $F^+$   
3 casi:

- $x \rightarrow y$  ottenuta per rifles., in tal caso  $y \subseteq x$   
Date 2 tuple  $t_1 \in t_2$  tali che  $t_1[x] = t_2[x]$  allora  
 $t_1[y] = t_2[y]$
- $x \rightarrow y$  ottenuta applicando l'aumento a una dipend.  
 $v \rightarrow w \in F^+$ , dove  $x = vz$  e  $y = wz$  per qualche  $z \subseteq R$   
Date 2 tuple  $t_1 \in t_2$  tali che  $t_1[x] = t_2[x]$   
Si avrà che  $t_1[v] : t_2[v] \subseteq t_1[z] : t_2[z]$ .  
Per pot. induttiva  $t_1[v] = t_2[v]$  porta a  $t_1[w] = t_2[w]$   
e insieme portano a  $t_1[z] = t_2[z]$ . Quindi  
 $t_1[y] : t_2[y]$
- $x \rightarrow y$  transitività  $x \rightarrow z \in z \rightarrow y \in F^+$   
Siano  $t_1 \in t_2$  tuple su  $r$  tali che  $t_1[x] = t_2[x]$   
per ip. indut  $x \rightarrow y \in z \rightarrow y$  sono in  $F^+$   
Questo significa che  $t_1[x] = t_2[x] \rightarrow t_1[z] = t_2[z]$   
 $\in t_1[z] : t_2[z] \rightarrow t_2[y] = t_1[y]$   
per transitività  $x \rightarrow y \in F^+$

$$F^+ \subseteq F^A$$

Cos truisco due tuple

$X^+$		$R - X^+$	
1	1	1	1
1	1	0	0

Le due tuple  
sono uguali  
su  $X^+$  ma non su  
 $R - X^+$

1° parte

Dato  $r$ , dato  $V \rightarrow W$ , supp. per ass che tale dip  
non sia sodd. da  $r$   
In questo caso le tuple hanno valore uguale su  $V$  ma  
non su  $W$ , quindi  $V \subseteq X^+ \in W \cap (R - X^+) \neq \emptyset$   
Dato che  $V \subseteq X^+$  per lemma  $X \rightarrow V \in V$  visto che  $V \rightarrow W$   
 $X \rightarrow W$  per transitività è sempre per il lemma  $W \subseteq X^+$   
contraddice  $W \cap (R - X^+) \neq \emptyset$

## 2° PARTE

Abbiamo dim che n<sup>o</sup> 1 istanza legge è quando  
soddisfa ogni dip in  $F^+$ , quindi anche  $x \rightarrow y$   
dato che  $x \subseteq X^+$ , le due tuple coincidono su  $x \in$   
dato che soddisfa  $x \rightarrow y$  devono coincidere anche su  
 $y$  e questo implica che  $y \subseteq X^+$  e per lemma  $x \rightarrow y \in F^+$

Algo chiusura  $X$

INPUT:  $R, F$ , sottins.  $X$  di  $R$

OUT: chiusura di  $X$  rispetto a  $F$

BEGIN

$$Z = X$$

$$S = \{ A \mid \text{LA DIP E INF} \}$$

$$\begin{array}{l} \text{LA STA} \\ \text{A DESTRA} \\ \text{E } x \rightarrow y \end{array}$$

$$\begin{array}{l} \text{LA PARTE} \\ \text{A SINISTRA} \\ \text{E SOT. DI } X \\ \text{E } x \rightarrow y \end{array}$$

$$Y \subseteq Z \}$$

while  $S \neq Z$

DO

BEGIN

$$Z = Z \cup S$$

$$S = \{ A \mid Y \rightarrow V \in F \wedge A \in V \wedge Y \subseteq Z \}$$

END

Dim correttza

induzione

$Z^0$  è il valore iniziale di  $Z$  ( $Z^0 = X$ )

$Z^i$  è  $S^i$  con  $i \geq 1$ , valore di  $Z$  dopo  $i$ -esima iteraz.

del while.

Diciamo che  $Z^i \subseteq Z^{i+1}$

OBIETTIVO È DIMOSTRARE CHE  $\exists i$  t.c.  $A \in Z^i$  SE E SOLO SE  $A \in X^+$ :

PARTE SE

$$Z^i \subseteq X^+ \quad \checkmark_i$$

$i > 0$  a b3, also per p. ind. che  $Z^{i-1} \subseteq X^+$ . sia  $A$  un attr.  
in  $Z^i - Z^{i-1}$ , significa che  $A$  è stato aggiunto  
all'ultima iteraz. è quindi, deve essere un a  
dip  $y \rightarrow V \in F$  t.c.  $y \subseteq Z^{i-1} \subseteq A \in V$ .  
dato che  $y \subseteq Z^{i-1}$  per p. ind. abbiamo che  $y \subseteq X^+$   
e per lemma  $x \rightarrow y \in F^+$ .

IN OLTRE, DATO CHE  $y \rightarrow v$  E  $x \rightarrow y$  ALLORA  $x \rightarrow v \in F^A$   
 PER TRANSITIVITÀ E QUINDI PER LEMMA 1  $v \subseteq x^+$ .  
 QUINDI, SE  $A \in 2^{\omega} - 2^{<\omega}$  SI HA CHE  $A \in x^+$  E QUINDI  
 $2^{\omega} \subseteq x^+$  PER IP. IND.

PARTE SOLO SE  $x^+ \subseteq 2^{\omega}$

SI A UN ATQ IN  $x^+$  È SIA j T.C.  $S^j = 2^j$  ( $2^j \in \omega$   
 VALORE DI 2 QUANDO ALGO TERMINA)  
 MOSTRANO CHE  $A \in 2^j$   
 POICHÉ  $A \in x^+$  SI HA  $x \rightarrow A \in F^A$  PER TEO  $F^A = F^A$   
 QUINDI,  $x \rightarrow A$  DEVE ESSERE SODD DA OAN,  
 ISTANZA LEGALE DI R.

COSTRUISCO TUPLE

$2^j$	$R - 2^j$
1 1 1 1 1 1	
1 1 1 0 0 0	

SE ESISTE UNA D.P.  $V \rightarrow W$  NON SODDISFAVA  
 DI R, SI AVREBBE CHE  $V \subseteq X^+$  E  $W \cap (R - 2^j) \neq \emptyset$  E  
 QUINDI  $2^j \neq S^j$  (CONTRAD). PER CHE SIGNIFICA CHE  
 NON SIAMO ARRIVATI ALL'ULTIMA ITERAZIONE.  
 DATO CHE R È ISIAN. LEG. DI R DEVE SODDISFARE  
 $X \rightarrow A$  CHE SI TROVA IN  $F^A$  E PER TEO  $F^A = F^A$   $A \in F^A \subseteq$   
 DATO CHE  $X = 2^0 \subseteq 2^j$   $A \in 2^j$

## DECOMPOSIZIONE

DATO R, UNA DECOM. DI R È UNA FAMIGLIA  $\rho = \{R_1, R_2, \dots, R_n\}$   
 DI SOTTOINS DI R CHE RICOPRE R, CIOÈ  $\bigcup_{i=1}^n R_i = R$

## EQUIVALENZA $F \equiv G$

DATI DUE SCHEMI  $F \in G$ .  
 $F \in G$  SONO EQUIVALENTI  $F \equiv G$  SE  $F^+ = G^+$

## LEMMA 2

DATI  $F \subseteq G$

SE  $F \subseteq G^+$  ALLORA  $F^+ \subseteq G^+$

DIM

SIA  $f$  UNA DIPENDENZA DI  $F^+ - F$ , PER TEOREMA  $F^+ = \overline{F}^+$ ,  $f$  È DERIVABILE DA  $\overline{F}$  CON ASS. ARMSTRONG E OGNI DIP IN  $F$  È DERIVABILE DA  $G$  CON ASS. ARM., ACCORDA  $f$  È DERIVABILE DA  $G$  ATTRAVERSO GLI ASS DI ARM

QUINDI DATO CHE  $F \subseteq G^+$  SIGNIFICA CHE DENTRO ALLA CHIUSURA DI  $G$  HO ANCHE  $F$ , MA GLI ASSIOMI CI APPLICO FINCHÉ POSSO E QUINDI DENTRO  $G^+$  C'È  $F^+$

P PRESERVA R

DATO R, DATO  $F \subseteq G$  DATO  $P$   
P PRESERVA R SE  $F = \bigcup_{i=1}^k \pi_{R_i}(F)$  DOVVE  
 $\pi_{R_i}(F) = \{x \rightarrow y \mid x \rightarrow y \in F^+ \wedge xy \subseteq R_i\}$

VERIFICA DEC F

DOBBIAMO VERIFICARE L'EQUIVALENZA DI  $F \subseteq G = \bigcup_{i=1}^k \pi_{R_i}(F)$   
PER DEC  $G \subseteq F^+$

ORA VERIFICHiamo CHE  $F \subseteq G^+$

ALGORITMO

IN:  $F \subseteq G$

OUT: VARIABILE SUCCESSO (SARÀ TRUE SE  $F \subseteq G^+$ )

BEGIN

SUCCESSO : TRUE

FOR EVERY  $x \rightarrow y \in F$

DO BEGIN

CACCIA  $x_G^+$

IF  $y \notin x_G^+$  THEN SUCCESSO : FALSE

END

END

ORA CI SERVE CALCOLARE  $X_G^+$ , MA NON CONOSCIAMO G

IN: R, f, f, SOTTOINS X DI R

OUT: CHIUSURA DI X RISPETTO A G =  $\bigcup_{i=1}^k \mathcal{N}_{R_i}(f)$  (IN 2)

BEGIN

Z = X

S = Ø

FOR j : 1 TO k

DO S = S  $\cup$  (Z  $\cap$  R<sub>j</sub>)<sup>+</sup>  $\cap$  R<sub>j</sub>

PRENDE  
GLI ATTR DI Z  
IN R<sub>j</sub> E NE  
CALCOLA LA  
CHIUSURA SU T

WHILE S ≠ Z

DO

BEGIN

Z = Z  $\cup$  S

FOR j : 1 TO k

DO S = S  $\cup$  (Z  $\cap$  R<sub>j</sub>)<sup>+</sup>  $\cap$  R<sub>j</sub>

PRENDE GLI  
ATTRIBUTI CHE  
APPARTENGONO  
A R<sub>j</sub>

END

END

CORRETEZZA DIM

INDICO CON Z<sup>0</sup> IL VALORE INIZIALE DI Z = X E CON Z<sup>i</sup> IL VALORE DI Z ALL'ESIMA ITERAZIONE (Z = Z  $\cup$  S). DA NOTARE CHE Z<sup>i</sup> ⊂ Z<sup>i+1</sup>. INDICHIAMO CON Z<sup>f</sup> IL VAL. DI Z QUANDO L'ACCO TERMINA.

DOBBIAMO MOSTRARE CHE  $X_G^+ \subseteq Z^f \Leftrightarrow Z^f \subseteq X_G^+$

DIMOSTRO SOLO  $Z^f \subseteq X_G^+$

CASO BASE

$Z^0 \subseteq X_G^+$  PERCHE'  $Z^0 = X$  (VERO PER Rif.)

IP. IND FINO A  $i-1$  ABBIAMO CHE  $Z^{i-1} \subseteq X_G^+$  ( $X \rightarrow Z^i \in G^+$ )

BANALMENTE  $2^{i-1} \subseteq 2^i \in \text{PER IP. IND } 2^{i-1} \subseteq X_G^+$

CONTROLLIAMO CHE A E  $2^{i-1} \subseteq 2^i$  (AGGIUNTA AL PASSO  $i$ )

QUESTO SIGNIFICA CHE  $\exists j \text{ T.C. } A \in (2^{i-1} \cap R_j)^+ \neq \cap R_j$   
E QUESTO VALE A DIRE CHE  $A \in (2^{i-1} \cap R_j)^+ \subseteq A \in R_j$



VALE A DIRE  
 $(2^{i-1} \cap R_j) \rightarrow A \in F^+$

SICCOME  $A \in R_j$ ,  $(2^{i-1} \cap R_j) \subseteq R_j \in$

$(2^{i-1} \cap R_j) \rightarrow A \in F^+$ , RICORDANDO LA DEF DI  
 $G = (\cup_{i=1}^k \pi_{R_i}(F))$  POSSIAMO DIRE CHE LA DIT E' IN G

1. PER IP. IND  $x \rightarrow 2^{i-1} \in G^+$

2. PER DEC.  $x \rightarrow (2^{i-1} \cap R_j) \in G^+$

3. PER TRANSIT.  $x \rightarrow A \in G^+$ , cioè  $A \in X_G^+$

QUINDI  $2^i \subseteq X_G^+$

JOIN SENZA PERDITA

DATO  $R_1$  PER OGNI  $r$  HA UN JOIN LEGALE DI  $R$  SENZA PERDITA  
 $r = \pi_{R_1}(r) \bowtie \dots \bowtie \pi_{R_k}(r)$  CHE

PER OGNI ISTANZA LEGALE DI  $r$   
 $m p(r) = \pi_{R_1}(r) \bowtie \dots \bowtie \pi_{R_k}(r) \models I$  HA:

-  $r \subseteq m p(r)$  FRANCHE ABBIANO  $\subseteq$  ABBIANO PERDITA  
CIOÈ IL JOIN CONTIENE PIÙ TUPLE  
DELLA STANTE, NEL MOMENTO IN CUI SONO = ABBIANO  
JOIN SENZA PERDITA.

-  $\pi_{R_i}(m p(r)) = \pi_{R_i}(r)$

-  $m p(m p(r)) = m p(r)$

# ALGO JOIN SENZA PERDITA (TABELLA)

IN:  $R, F, \rho$

OUT: DICE SE HA UN JOIN SENZA PERDITA

COSTRUISCO TAB:

- HA  $|R|$  COL E RIGHE
- ALL'INCROCIO DEGLI ELEMENTI RIGA E J-ESIMA COL MEZZO
  - $a_j$  SE  $A_j \in R_j$
  - $b_{ij}$  SE  $A_j \notin R_j$

REPEAT

FOR EVERY  $x \rightarrow F$

DO IF CI SONO  $t_1, t_2$  IN  $r$  T.C.  $t_1[x] = t_2[x]$   
 $\in t_1[y] \neq t_2[y]$

THEN FOR EVERY ATTRIBUTE  $A_j$  IN  $F$  DO IF  $t_1[A_j] = "a_j"$

THEN  $t_2[A_j] = t_1[A_j]$   
 ELSE  $t_2[A_j] = t_2[A_j]$

UNTIL  $r$  HA UNA RIGA CON TUTTE "a" OR NON E' CAMBI.

IF HA UNA RIGA CON TUTTE "a"  
 THEN JOIN SENZA PERDITA

ELSE  $\rho$  NON HA JOIN

END

## DIMOSTRAZIONE CORRETTEZZA

NON AVEVO VOGLIA DI SCRIVERLO

*Dimostrazione:* Dobbiamo dimostrare che ha un join senza perdita se e solo se quando l'algoritmo termina la tabella ha una tupla con tutte a.

- Dimostriamo soltanto che se la decomposizione ha un join senza perdita allora la tabella deve avere una riga con tutte a.

Supponiamo per assurdo che  $\rho$  abbia un join senza perdita e che con l'algoritmo terminato la tabella  $r$  non abbia una tupla con tutte a.

La tabella possiamo vederla come un'istanza legale di  $R$  dato che l'algoritmo termina quando non ci sono più violazioni delle dipendenze in  $F$ .

Poiché nessun simbolo a che compare nella tabella iniziale viene modificato dall'algoritmo, per ogni  $i = 1, \dots, k$  abbiamo che  $\pi_{R_i}(r)$  contiene una tupla con tutte a, quindi questo significa che  $m_\rho(r)$  contiene una tupla con tutte a dato che prima o poi quelle tuple iniziali si incontreranno nel join. Questo significa che  $m_\rho(r) \neq r$  almeno per questa istanza infatti abbiamo detto che la tabella non ha una riga con tutte a come ipotesi ma abbiamo trovato un'istanza legale che invece le ha.

Essenzialmente non sappiamo come funziona l'algoritmo perché serve la seconda parte, però, in  $m_\rho(r)$  ci sarà sempre una riga con tutte a dato che è il join fra i sottoschemi, infatti accade sempre che  $r \subseteq m_\rho(r)$ . Noi però stiamo supponendo che  $m_\rho(r) = r$  dato che abbiamo join senza perdita e che, per assurdo, in  $r$  non ci sia una riga con tutte a ma questo appunto implicherebbe che  $r \neq m_\rho(r)$ . Quindi non stiamo dicendo il "come" funziona l'algoritmo ma soltanto che nel caso di un join senza perdita questo viene rilevato. (Ci ho messo un po' per capirlo spero di ricapirlo leggendo)

## COPERTURA MINIMALE

ESISTE SEMPRE UNA DEC. CHE

- È IN BNF
- PRESERVA F
- HA JOIN SENZA PERDITA

SI CACCIOLO IN TEMPO POLINOMIALE CON ALGO

## DEF COP MIN

DATO F, LA COP MIN DI F È ON INSERIRE G  
DI DIP FUNZ EQUIV. A F T.C.

- OGNI DIP IN G HA LA PARTE A DX SINGLETON
- PER NESSUNA DIP  $X \rightarrow A \in G$   $\exists X' \subseteq X$  T.C.  
 $G = G - \{X \rightarrow A\} \cup \{X' \rightarrow A\}$  (ATTR. A SX RIDOND.)
- PER NESSUNA DIP  $X \rightarrow A \in G$  DEVE ACCADERE CHE  $G = G - \{X \rightarrow A\}$  (DIP NON RIDOND.)

LA COP. NON È UNICA

## ALGO DECOMPOSITION

IN: R, F, COP MIN

OUT: DEC<sub>IN P E IN BNF</sub> DI R T.C. OGNI SOTTOSCHEMA

BEGIN

$$P = \emptyset$$

$$S = \emptyset$$

FOR EVERY  $A \in R$  T.C. CHE È A NON È  
CONVOLTO IN NESSUNA DIP FUNZ IN F

DO  $S = S \cup A \rightarrow S$  NON CONV. VIENE ACCIUNTO

IF  $S \neq \emptyset$  THEN  $\rightarrow S$  ATTR. NON CONV.

BEGIN  
 $R = R - S \rightarrow$  RIHAUOVA DA R  
 $P = P \cup \{S\} \rightarrow$  LI ACCUNGE ACC DEC  
END

IF  $\exists$  DIP FUNZ IN R  
GCI ATTR IN F CHG CON VOLCE TUTTI

THEN  $\rho = \rho \cup \{R_f\} -$  AGGIUNGE R A  $\rho$   
 EDGE FOR EVERY  $x \rightarrow A \in F$  DO  $\rho = \rho \cup \{x \rightarrow A\}$   
 ACTAMENTI AGGIUNGE  
 $x \cup \{A\}$  A  $\rho$

## DIM CORRETEZZA

-  $\rho$  PRESERVA  $F$

SE  $\rho$  PRESERVA COP MIN ALLORA PRESERVA  $F$

SIA  $G = \bigcup_{i=1}^k R_i(F)$ .  $\nexists$  DIP  $x \rightarrow A \in F$  L'ALLO  
 CHE  $x \in A \in \rho$ , QUESTE DIP SARANNO IN G  
 QUINDI  $F \subseteq G \rightarrow F^+ \subseteq G^+$

INVECE  $G \subseteq F^+$  PER DEF QUINDI  $\rho$  PRES F

- OGNI SOTOSCHEMA E' IN 3NF

### 3 CASI:

#### • Caso 1: $S \in \rho$

Questo è il caso degli attributi "orfani" (non coinvolti in nessuna dipendenza).

#### 👉 Spiegazione:

- Se  $S$  è stato aggiunto alla decomposizione, significa che ogni attributo in  $S$  non appare mai in nessuna dipendenza.
- Quindi ogni attributo in  $S$  è parte della chiave di quel sottoschema (perché nessuna dipendenza lo determina).
- Di conseguenza, non esistono dipendenze da violare, e il sottoschema è automaticamente in 3NF.

#### • Caso 2: $R \in \rho$

Questo è il caso in cui c'è una dipendenza funzionale che coinvolge tutti gli attributi di  $R$ .

#### 👉 Spiegazione:

- Se  $R$  è nella decomposizione, allora esiste una dipendenza come  $R-A \rightarrow A$  (cioè quasi tutti gli attributi determinano uno solo).
- Dato che  $F$  è una copertura minima, questo tipo di dipendenza è essenziale e non ridondante.
- $R-A$  sarà quindi una chiave del sottoschema  $R$ , oppure un superchiave.
  - "Se è una chiave, allora  $R-A \rightarrow A$  è in 3NF perché la sinistra è una chiave."
  - "Se è solo una superchiave, va comunque bene perché 3NF lo permette."

Inoltre:

- Non possono esistere altre dipendenze con sinistra  $x \subseteq R-A$  e destra  $A$ , perché sarebbe ridondante (violerebbe la minimalità).
- Per ogni altra dipendenza nel sottoschema, si dimostra che:
  - O la destra è parte di una chiave, oppure
  - La sinistra è una superchiave.

In entrambi i casi: la dipendenza non viola la 3NF.

◆ Caso 3:  $X \rightarrow A \in \rho$

Questo è il caso in cui l'algoritmo crea un sottoschema per ogni dipendenza  $X \rightarrow A$  in  $F$ .

👉 Spiegazione:

- Poiché  $F$  è minimale, non esiste una dipendenza  $X^* \rightarrow A$  con  $X^* \subset X \rightarrow$  significa che  $X$  è **minimo** per determinare  $A$ .
- Questo implica che  $X$  è una chiave per  $X \rightarrow A$ , oppure è una superchiave.
- Quindi ogni dipendenza  $X \rightarrow A$  non viola la 3NF.

Per qualsiasi altra dipendenza  $Y \rightarrow B$  all'interno del sottoschema:

- Se  $Y$  determina  $B$ , si mostra che:
  - $Y$  è superchiave, oppure
  - $B$  è parte di una chiave.

In entrambi i casi → non viola la 3NF.

$$\begin{array}{c}
 R \models F \\
 Y \subseteq X \subseteq R \\
 X \rightarrow Y \in F^A \\
 \\ 
 \text{AUM} \quad X \rightarrow Y \in F^A \\
 XZ \rightarrow YZ \in F^A \quad \forall Z \subseteq R \\
 \\ 
 \text{TRANSIT.} \quad X \rightarrow Y \in F^A \\
 Y \rightarrow Z \in F^A \\
 X \rightarrow Z \in F^A
 \end{array}$$

