

DEFINIZIONE DF

DATO $R \subset T, F$ E $X, Y \in T$,

- $\forall r$ VERA DA DI R

- $\forall t_1, t_2 \in r$, $t_1[x] = t_2[x] \Rightarrow t_1[y] = t_2[y]$

$X \rightarrow Y$

SUPERCHIAVE

DATO $R \subset T, F$ E $X \in T$

$X \rightarrow T$

CHIAVE (SUPERCHIAVE MINIMALE)

DATO $R \subset T, F$ E $X \in T$

$\nexists W \subset X$ NOR VALORE $W \rightarrow T$

DIFFERENZA "IN" E "EXISTS"

IN = BINARIO. VERIFICA SE UN ELEMENTO DELLA QUERY PRINCIPALE È CONTENUTO NEL RISULTATO DELLA SUBQUERY.

EXISTS = UNARIO. VERIFICA SE ESISTE UNA RIGA CHE SODDISFA LA CONDIZIONE DELLA SUBQUERY.

DIFFERENZA "ANY", "ALL", "EXISTS"

CONFRONTO RISULTATO SUBQUERY CON RIGA PRINCIPALE

ANY = SE ALMENO UN VALORE RESTITUITO SODDISFA

ALL = SE TUTTI I VALORI SODDISFANO

EXISTS = VERO SE RESTITUISCE ALMENO UNA TUPLA

DIFFERENZA "WHERE" "HAVING"

WHERE = SI APPLICA ALLA RIGA

HAVING = SI APPLICA AI GRUPPI

NORMALIZZAZIONE

CHIAVE CANDIDATA

X È CHIAVE CANDIDATA DI $R_{CT,F}$ SSE:

- X È SUPERCHIAVE, $X^+ = T$

- $Y \subset X \Rightarrow Y^+ \neq T$, nessun Y È SUPERCHIAVE

ATTRIBUTO ESTRANGE

DATO $X \rightarrow Y$, $A \in X$ È ESTRANGE SSE:

- $X - \{A\} \rightarrow Y \in F^+$

ATTRIBUTO PARI

ATTRIBUTO CHE APPARTIENE ALMENO A UNA

CHIAVE CANDIDATA

CITIANG E SUPERCHIAVE CON DF

SUPERCHIAVE: $X \rightarrow T$

CHIAVE: SE $X \rightarrow T$, $\forall A_i \subset X, A_i \rightarrow T \notin F$

COPERTURA CANONICA

F È UNA COPERTURA CANONICA SSE:

- LA PARTE DESTRA DI OGNI DF IN F È ATTRIBUTO
- NO ATTRIBUTI ESTRANEI
- NO DF RIDONDANTI

DIPENDENZA RIDONDANTE

$X \rightarrow Y$ RIDONDANTE SE $(F - \{X \rightarrow Y\})^+ = F^+$

DIPENDENZA ATOMICA

$X \rightarrow A_1 A_2 A_3$ NO!

$X \rightarrow A_1 \quad X \rightarrow A_2 \quad X \rightarrow A_3$ SI!

ALGORITMO PER COPERTURA CANONICA

- 1) DF IN DIPENDENZE ATOMICHE
- 2) ELIMINARE ATTRIBUTI ESTRANEI
- 3) ELIMINARE DF RIDONDANTI

ASSIOMI DI ARMSTRONG

- 1) SE $Y \subseteq X \Rightarrow X \rightarrow Y$
- 2) SE $X \rightarrow Y$, $Z \subseteq T \Rightarrow XZ \rightarrow YZ$
- 3) SE $X \rightarrow Y$, $Y \rightarrow Z \Rightarrow X \rightarrow Z$

R
A
T



DECOMPOSIZIONE SENZA PERSITA

$R(X)$ DECOMPOSTO IN $R_1(X_1), R_2(X_2)$

È SENZA PERSITA SE:

- NATURAL JOIN PRODUCE $R(X)$
- SE $T_1 \cap T_2 \rightarrow T_1 \in F^+$ OR $T_2 \in F^+$

CONSERVA LE DIPENDENZE SE:

- L'UNIONE DELLE DIPENDENZE PROIETTATE È UNA COPERTURA DI F

BCNF CON DF

DATO $R(T, F)$, $X \rightarrow Y | X$ È SUPERCHIAVE DI R)

3FN

R È IN 3FN SE $\boxed{X \rightarrow Y}$ NON PARTE IN F È VENIRE CATA

ALMENO UNA DELLE CONDIZIONI:

ex. $F(A \rightarrow B, C \rightarrow D)$

- X CONTIENE UNA SUPERCHIAVE

- $\forall A_i \in Y, A_i \in k$ SUPERCHIAVE

DEVE
ISPETTARE

PURE

- 3FN E BCNF PRESERVANO I DATI

- BCNF POTREBBE NON PRESERVARE LE DF

- 3FN LE PRESERVA SEMPRE

ALGORITMO DI ANALISI (BCNF)

COMPLICATÀ: $O(n^n)$ ESPONENZIALE

① VERRÀ RICERCATO CHE R SIA IN BCNF

② SEPARA IN $R_1 = \{X^+\}$, $R_2 = \{Y \notin X^+ + X\}$

③ PROGETTA LE DIPENDENZE SU R_1 E R_2

④ TORNA AL PASSO ①

ALGORITMO DI SINTESI (3FN)

① CREO COPERTURA CANONICA G-DI-F

①.1 CREO DF ATOMICHE

①.2 RIMUOVO ATTRIBUTI ESTRANEI

①.3 RIMUOVO ATTRIBUTI RIDONDANTI

- ② DECOMPOGO \mathcal{G} IN INSIEMI CON STESSO X DI $X \rightarrow Y$
- ③ UNISCO LE DF DI OGNI f_i IN UNA UNICA ($X \rightarrow Y$, $X \rightarrow Z \Rightarrow X \rightarrow YZ$)
- ④ TRASFORMO OGNI f IN UN R CON X COME CHIAVE
- ⑤ ELIMINO SCHEMI RIDONDANTI
- ⑥ SE MESSUN R_i CONTIENE UNA CHIAVE DELL' ORIGINALE R CERCHANDOLA SU \mathcal{G} MIDOTTO IN COPERTURA CANONICA,
AVVIENNO R_{i+1} CHE CONTIENE LA CHIAVE

COMPLESSITÀ: $O(n)$ POLINOMIALE

VANTAGGI 3FN VS BCNF

- 3FN MENO RESTRITTIVA

X

- TOLLENA ALCUNE RIDONDANZE

- MENO QUALITÀ CERTIFICATA

- 3FN SEMPRE OTTEMIBILE

✓

DBMS

GESTIONE DEL BUFFER, AREA DELLE PAGINE

GESTORE MEM. PERMANENTE = RESPONSABILE DELLA GESTIONE DI FILE LOGICI MEMORIZZATI SUL DISCO. PRESENTA I FILE COME UN INSIEME DI FILE LOGICI COMPOSTI DA PAGINE FISICHE.

GESTORE DEL BUFFER = TRASFERISCE LE PAGINE DALLA MEMORIA TEMPORANEA A QUELLA PERMANENTE.

CARICA LE PAGINE RICHIESTE DAL DISCO IN MEMORIA PRINCIPALE E MANTIENE UNA CACHE.

COME FUNZIONA LA GESTIONE DELL'AREA DEL BUFFER?

IL BUFFER POOL È UNA CACHE CHE CONTIENE PAGINE LETTE RECENTEMENTE DAL DISCO.

Ogni PAGINA DEL BUFFER HA 2 STATI:

- Pin Count = N° TRANSAZIONI CHE STANNO USANDO LA PAGINA
- Dirty Bit = PAGINA MODIFICATA MA NON ANCORA

SALVATA SU DISCO SE TRUE. SE FALSE, PAGINA SINCRONIZZATA SUL DISCO

INDICE PRIMARIO VS SECONDARIO

- **PRIMARIO:** CHIAVE DI ORDIMENTO DEL FILE
COINCIDE CON L' INDICE.
- **SECONDARIO:** COSTRUITO SU UNA CHIAVE DIVERSA
DALLA CHIAVE DI ORDIMENTO DEL FILE.

EX:

ID_UTENTE	NOME
0	MARCO
1	MARCO
2	LUCA
IDX	PRIMARIO SECONARIO

PARAMETRI SORT-MERGE

PRENDE O_E E O_I ORDINATE SULL'ATTRIBUTO DI GIUNZIONE

NESTED LOOP VS SORT-MERGE EFFICIENZA

SORT-MERGE: $O(N \log(N))$

NESTED LOOP: $O(n^2)$

SORT MERGE COME FUNZIONA

- SORT INTERNO = si leggono le pagine una alla volta e si sostano (Quicksort). Prima ordinata = "RUN"
- MERGE = le RUN vengono messe per produrre una run ordinata in blocchi di Z run alla volta. Vengono utilizzati NB BUFFER che servono a:
 - 1° > contenere la prima run
 - 2° > contenere la seconda run
 - 3° > contenere la run mescolataIn caso di $Z = 2$.

COMPLESSITÀ = $O(N \log(N))$

QUANDO SORT MERGE > PAGE NESTED LOOP ?

QUANDO voglio il mio risultato ordinato, aumenti PAGE NESTED LOOP SFARITA MENO OPERAZIONI I/O

PAGE NESTED LOOP VS NESTED LOOP

- **Nested Loop** = È UN JOIN BASATO SU UN DOPPIO CICLO FOR
- **Page Nested Loop** = JOIN CHE ANZICHÉ CONFRONTARE SINGOLE TUPLE,
CONSIDERA INTERE PAGINE DI DATI, RIDUCENDO L' I/O NECESSARI PER
IL JOIN

PIANO DI ACCESSO FISICO

ALGORITMO PER ESEGUIRE UN'INTERROGAZIONE USANDO
GLI OPERATORI FISICI DISPONIBILI

OPERATORI FISICI?

SONO ITERATORI CHE UTILIZZANO LE RISORSE DELLA
MACCHINA.

OFFRONO LE INTERFAZIE OPEN, CLOSE, NEXT, isDONE PER ITERARE SULLE
TABELLE

FILTER?

RESTRIZIONE DI UNA TABELLA SENZA INDICI. $\sigma(\psi)$

GESTIONE DELL' AFFIDABILITÀ

DEFINIZIONE TRANSAZIONE

UNITÀ DI LAVORO FONDAMENTALE PER MODIFICARE IL CONTENUTO DI UNA BD. DELIMITATA DA BEGIN/END TRANSACTION E PUÒ ESSERE CONFERMATA (COMMIT) O ANNULLATA (ROLLBACK).

CON UNA TRANSAZIONE VOGLIAMO SCRIVERE SU DISCO?

Sì, solo con "COMMIT". LE TRANSAZIONI DEVONO ESSERE ATOMICHE.

ACID?

- A, ATOMICITÀ = UNA TRX È "TUTTO O NIENTE". LE OPERAZIONI DEVONO ESSERE ESEGUITE CON SUCCESSO PRIMA CHE LE MODIFICHE SIANO APPLICATE AL DATABASE. ANNULLATE DEL TUTTO IN CASO DI FALLIMENTO.

- C, CONSISTENZA = UNA TRX DEVE MANTENERE IL DB IN UNO STATO CONSISTENTE, RISPETTANDO I VINCOLI.

- I, ISOLAMENTO = OGNI TRX È ESEGUITA INDIPENDENTEMENTE DALLE ALTRE. LE TRX CONCORRENTI VENGONO ESEGUITE COME SE FOSSENRO SEQUENZIALI (E NON PARALLELE)

- D, (DURABILITY) DURABILITÀ = LE MODIFICHE DEL COMMIT DEVONO ESSERE PERMANENTI, ANCHE DOPO UN GUASTO.

REGOLE SCRITTURA LOG

- **WAL** = Write Ahead Log. Il BS di ogni record di log deve essere scritto prima che l'operazione venga effettuata nella BD.
- **CP** = Commit Precedence. La parte AS di ogni record di log deve essere scritta nel log prima del commit della TRX.

UNDO/REDO

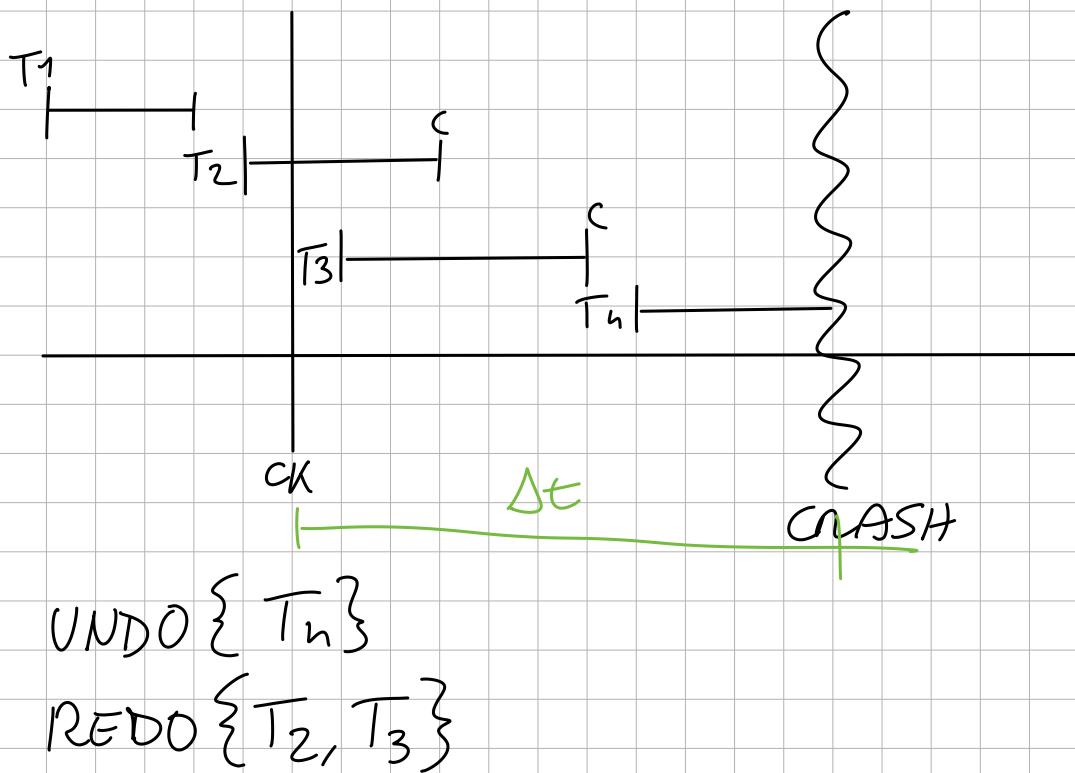
- **UNDO** = disfa un'azione su O. Copia il BS su O, annullando le modifiche
- **REDO** = rifà un'azione su O. Copia l'AS su O, ripristinando le modifiche

PERMUTAZIONI UNDO/REDO

- **UNDO-NON-REDO** = ANNULLA MA NON RIPRISTINA. Modifiche applicate solo dopo il "COMMIT"
- **Non-UNDO-REDO** = NON ANNULLA LE MODIFICHE. SONO APPLICATE IMMEDIATAMENTE
- **Non-UNDO-Non-REDO** = NON USA IL LOG. Modifiche applicate IMMEDIATAMENTE

COME Agisci con UNDO-REDO?

OTTENGO i 2 insiemi UNDO-REDO così:



SPOSTO IN REDO{} TUTTE LE T CHE ESEGUONO UN Commit prima del CRASH.

LASCIO E AGGIUNGO TUTTE LE T CHE SONO INIZIATE o INIZIANO NEL Δt CHE NON HANNO UN Commit o ABORTANO.

REDO{} \Rightarrow copio in O il AS

UNDO{} \Rightarrow copio in O il BS

GESTIONE CONCORRENZA

LOCK?

È UN BLOCCO PER LA LETTURA/SCRITTURA DEI DATI. UNA TRX RICHIENDE IL LOCK PRIMA DI ESEGUIRE UN' OPERAZIONE SU UN DATO E A TRX FINITA LO RILASCIÀ (ZPL). LETTURE CONCORR. SCRITTURE MUTEX.

Semiallazzabilità?

UN' INSIEME DI TRX PARALLELE PRODUCE LO STESSO RISULTATO DI UN' INSIEME DI TRX SEQUENZIALI.
GARANTISCE CONSISTENZA.

PROBLEMI CONCORRENZA?

DEADLOCK

STARVATION

PROTOCOLLO PESSIMISTICO?

PROTOCOLLO DI GESTIONE DELLE CONCORRENZE CHE AGISCE RITARDANDO L'ESECUZIONE DI TRX CHE POTREBBERO GENERARE CONFLITTO CON QUESTA CORRENTE.

PROTOCOLLO OTTIMISTICO ?

PERMETTONO L'ESECUZIONE SOVRAPPOSTA E
NON SINCRONIZZATA DI TRX ED EFFETTUANO
CONTROLLI SUI CONFLITTI SOLO DOPO IL COMMIT

PROTOCOLLO Strict - 2PL ?

- FASE DI CRESCITA : LA TRX CHIEDE TUTTI i LOCK IN QUESTA FASE
- FASE DI DECRESCA : RILASCIÀ TUTTI i LOCK AL MOMENTO DEL COMMIT O ROLLBACK
- Strict = LOCK LETTURA RILASCIATI PRIMA SE NON NECESSARI PER IL COMMIT. LOCK SCRITTURA MANTENUTI FINO AL COMMIT.

GRAFO DELLE RICHIESTE E GESTIONE CONCERNERZE
GRAFO DINETTO I CUI MODI SONO LE TRX ATTIVE :

$T_1 \rightarrow T_2$ T_1 ASPETTA UNA RELEASE DA T_2

SE SI CREA UN CICLO (DEADLOCK), VIENE FATTA ABORTIRE LA TRX PIÙ RECENTE, QUELLA CON MENO RISORSE O QUELLA CON COSTO DI ABORT PIÙ BASSO.

COME MISURARE UN DEADLOCK?

- TIME OUT: OGNI TRX HA UN TIMEOUT ENTRO IL QUALE DEVE CONCLUDERSI OPPURE ABORT
- DEADLOCK AVOIDANCE: PREVENIRE EVITANDO LOCK/UNLOCK SIMULTANEI DI TUTTE LE RISORSE O CON L'USO DI TIMESTAMP / CLASSI DI PRIORITÀ.
(RISCHIO STARVATION)
- DEADLOCK DETECTION: GRAFO DELLE RICHIESTE

TIMEOUT E TIMESTAMP

ALTERNATIVA A 2PL:

- OGNI TRX HA UN TIMESTAMP (INIZIO TRX)
- OGNI TRX NON PUÒ LEGGERE O SCRIVERE UN DATO SCRITTO DA UNA TRX CON TIMESTAMP MAGGIORE
- UNAVALA MA UN DATO SCRITTO DA TRX CON TIMESTAMP MAGGIORE