

# Basi di dati distribuite

Una **base di dati distribuita**

- i dati sono distribuiti su server (DBMS) diversi
- le transazioni coinvolgono server diversi
- i DBMS possono essere **omogenei** o **eterogenei**
- può essere locale o geografica

L'esecuzione di una **transazione distribuita**

- la base di dati può presentare diversi livelli di **trasparenza**
  - un **estremo**: l'utente conosce un solo server a cui invia richieste SQL come se i dati si trovassero su quel server
  - l'**altro estremo**: l'utente deve conoscere l'esatta locazione dei dati e i linguaggi usati dai vari server

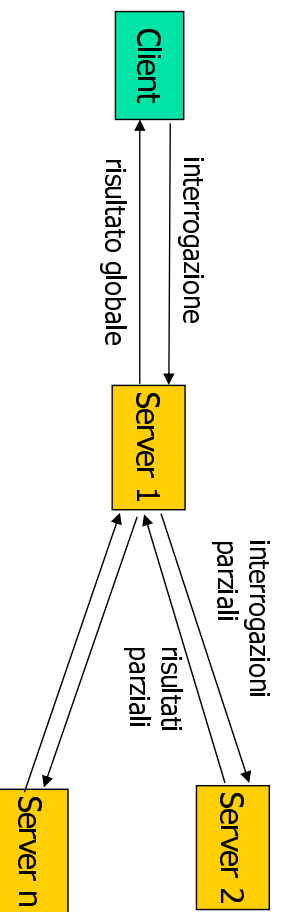
Scarselli Franco

Sistemi per basi di dati 2006-2007

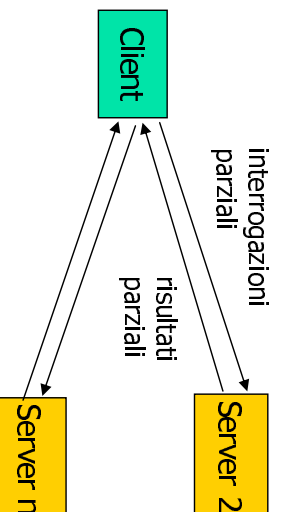
215

## Interazione client – DBMS

Schema di  
interazione 1



Schema di  
interazione 2



Scarselli Franco

Sistemi per basi di dati 2006-2007

216

# Livelli di trasparenza: un esempio



FORNITORE(codice,Nome,Citta)

Nella sede di Milano: FORNITORE1 =  $\sigma_{Citta="milano"}(FORNITORE)$

Nella sede di Roma: FORNITORE2 =  $\sigma_{Citta="roma"}(FORNITORE)$

Esempio di  
frammentazione  
orizzontale

Trasparenza completa

```
SELECT nome INTO :n
FROM fornitore
WHERE codice=:c
```

Trasparenza di linguaggio

```
SELECT nome INTO :n
FROM fornitore1@milano.ditta.it
WHERE codice=:c
if :empty then
SELECT nome INTO :n
FROM fornitore2@roma.ditta.it
WHERE codice=:c
```

Nessuna trasparenza

```
SELECT nome INTO :n
FROM fornitore1
WHERE codice=:c
if :empty then
SELECT nome INTO :n
FROM fornitore2
WHERE codice=:c
```

Scarselli Franco

Sistemi per basi di dati 2006-2007

1. Apri una connessione con Milano
2. Invia un'interrogazione nel linguaggio del server di Milano
3. Ricevi la risposta
4. Apri una connessione con Roma
5. Invia un'interrogazione nel linguaggio del server di Roma
6. Ricevi la risposta

## Classificazione delle transazioni



### Richieste remote

- sono transazioni di sola lettura indirizzate ad un solo DBMS remoto

### Transazioni remote

- sono transazioni costituite da un numero qualsiasi di comandi SQL indirizzate ad un solo DBMS remoto

### Richieste distribuite

- sono transazioni rivolte a piu' DBMS, ma ogni comando SQL fa riferimento ad un solo DBMS

### Transazioni distribuite

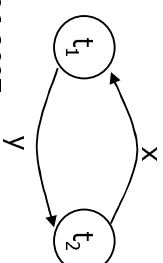
- sono transazioni rivolte a piu' DBMS, dove ogni comando SQL puo' far riferimento a dati presenti su DBMS diversi

# Controllo di concorrenza in ambito distribuito

## Teoria

- una **transazione distribuita** è rappresentata da un insieme di **sottotransazioni**, ognuna relativa a un nodo
- es:
  - $t_1: r_{11}(X) w_{11}(X) r_{12}(Y) w_{12}(Y)$
  - $t_2: r_{22}(Y) w_{22}(Y) r_{21}(X) w_{21}(X)$   
(il primo indice indica la transazione, il secondo indica il nodo)
- La **serializzabilità locale** degli schedule **non garantisce quella globale**
- es: i seguenti sono due schedule relativi a  $t_1$  e  $t_2$  che sono **localmente seriali**, ma **non globalmente serializzabili**

- $S_1: r_{11}(X) w_{11}(X) r_{21}(X) w_{21}(X)$
- $S_2: r_{22}(Y) w_{22}(Y) r_{12}(Y) w_{12}(Y)$



Scarselli Franco

Sistemi per basi di dati 2006-2007

219

# Controllo di concorrenza in ambito distribuito II

## La **serializzabilità globale**

- Richiede l'esistenza un unico schedule globale S e schedule locali  $S_i$ 
  - S deve essere seriale
  - la proiezione di S sui nodi deve produrre gli  $S_i$

Lo schedule S è **serializzabile**

- secondo il **locking a due fasi**
  - se ogni nodo applica il metodo 2PL stretto per le operazioni di competenza
  - l'azione di commit viene fatta in maniera atomica in un istante in cui tutti i nodi detengono tutte le risorse necessarie
- secondo il metodo del **timestamp**
  - le sottotransazioni acquisiscono un unico timestamp
  - i nodi applicano localmente il controllo di concorrenza basato sul timestamp

Scarselli Franco

Sistemi per basi di dati 2006-2007

220

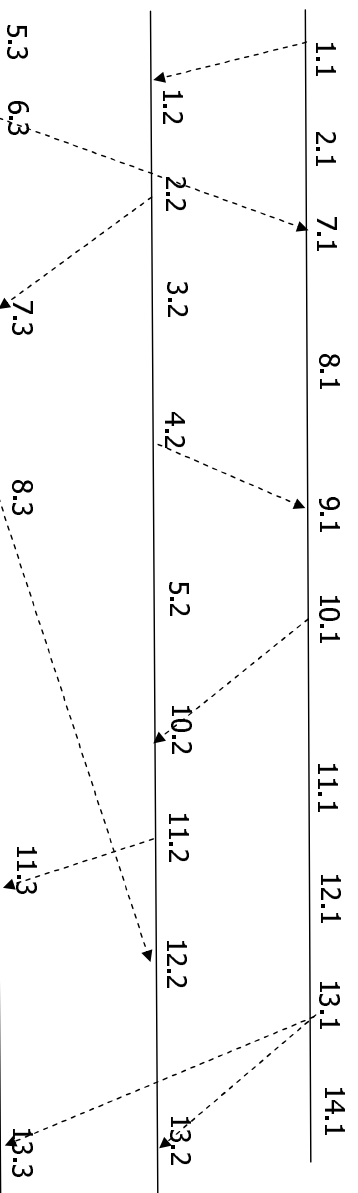
# Metodo di Lamport per assegnare i timestamp

Per avere i timestamp **unici in ambito distribuito**

- Il timestamp è composto di due gruppi di cifre
  - le meno significative identificano **il nodo**
  - le piu' significative identificano **gli eventi che accadono sul nodo**
- l'indice relativo agli eventi
  - viene incrementato ad ogni evento locale
  - viene sincronizzato ad ogni scambio di messaggi
- I timestamp **rispettano l'ordinamento degli eventi sui nodi**

# Metodo di Lamport per assegnare i timestamp II

Un esempio con tre nodi





# Commit distribuito

## Atomicita' in ambito distribuito

- tutti i nodi partecipanti in una transazione devono **fare la stessa cosa**: commit o rollback
- occorre garantire che eventuali guasti non lascino i nodi in uno stato **non coerente**
- Il protocollo piu' usato è il **commit a due fasi**

## Tipi di guasti

- caduta di un nodo
- perdita di messaggi
- caduta della rete ed eventuale partizionamento

Scarselli Franco

Sistemi per basi di dati 2006-2007

223



# Commit a due fasi

Il **commit a due fasi** si svolge attraverso uno scambio di messaggi fra

- **resource manager**  
i server che gestiscono di dati
- **transaction manager**  
il coordinatore che ha il compito di controllare l'andamento della transazione

## Consiste di due fasi

1. il transaction manager chiede ai resource manager se vogliono fare un abort o un commit e raccoglie le risposte
2. il transaction manager decide se fare abort o commit e ne informa i resource manager

Scarselli Franco

Sistemi per basi di dati 2006-2007

224

## Commit a due fasi II

Il protocollo prevede la scrittura *sincrona* di nuovi record sul log

### Il transaction manager

- **prepare**
  - contiene la lista dei processi dei resource manager coinvolti
  - indica l'inizio del protocollo
- **global commit (global abort)**
  - esprime la decisione di fare un commit (un abort)
- **complete**
  - indica la fine del protocollo

### Il resource manager

- **ready**
  - indica la disponibilità a partecipare al protocollo e a fare il commit della transazione
- **don't commit**
  - indica la disponibilità a partecipare al protocollo e l'impossibilità di fare il commit

Scarselli Franco

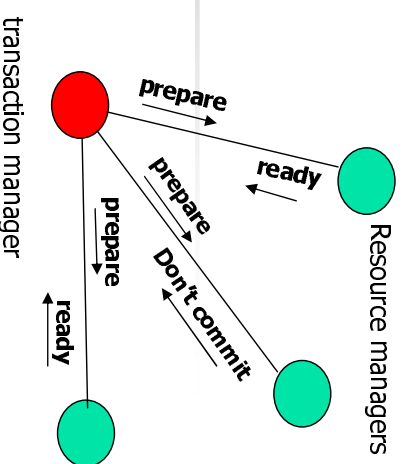
Sistemi per basi di dati 2006-2007

225

## Commit a due fasi III

### Fase 1 (si decide cosa fare)

- Il transaction manager
  - scrive **prepare** sul log
  - **invia un messaggio** ai resource manager
  - si mette in attesa della risposta impostando un timeout



- I resource manager
  - non appena hanno finito di eseguire la loro parte di transazione scrivono **ready** sul log
  - quando arriva il messaggio del transaction manager, **rispondono**
    - se sono in stato affidabile, i resource manager rispondono con **ready**
    - se non sono in stato affidabile, i resource manager rispondono con **don't commit**

### Il transaction manager

- **colleziona** le risposte
  - se tutte le risposte sono **ready** scrive **global commit** sul log
  - se almeno una risposta è **don't commit** scrive **global abort** sul log

Scarselli Franco

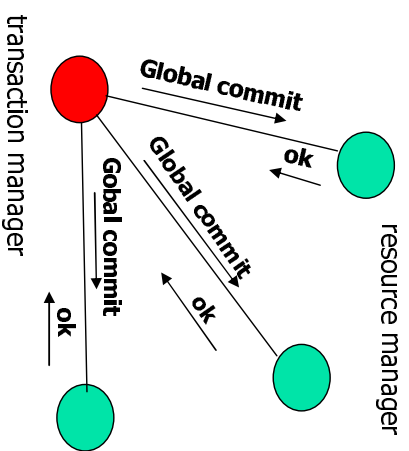
Sistemi per basi di dati 2006-2007

226

# Commit a due fasi IV

## Fase 2 (si implementa la decisione)

- Il transaction manager
  - comunica la decisione a tutti i resource manager
- I resource manager
  - scrivono sul log **commit** o **abort**
  - inviano un acknowledge al transaction manager
- Il transaction manager
  - quando ha ricevuto tutti gli acknowledge scrive **complete** sul log



Scarselli Franco

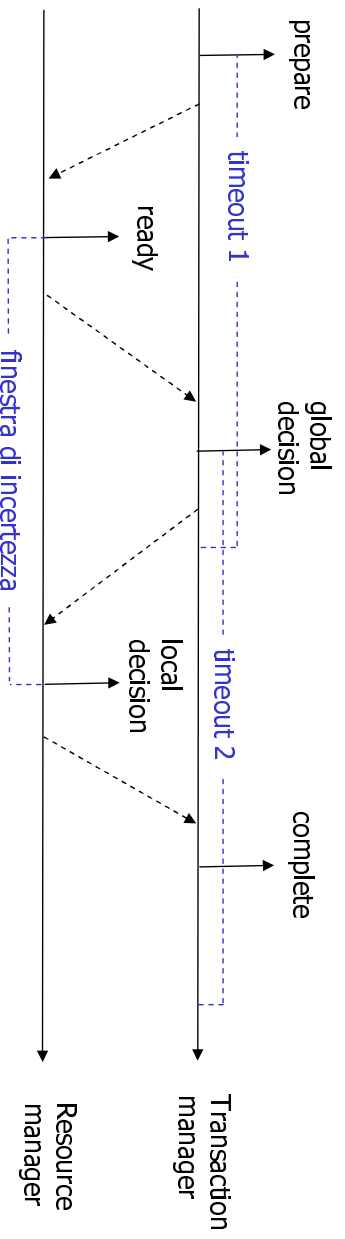
Sistemi per basi di dati 2006-2007

227

# Osservazioni

## Osservazioni

- la mancanza di comunicazione fra transaction e resource manager causa
  - durante fase 1, l'abort della transazione
  - durante fase 2, la ripetizione dell'invio del messaggio
- il protocollo è progettato per ridurre al minimo la **finestra di incertezza**
- durante la finestra di incertezza alcune risorse rimangono bloccate



Scarselli Franco

Sistemi per basi di dati 2006-2007

228



# Ripristino

In caso di perdita di messaggi

- **prepare e ready**
  - scatta il timeout e si procede con un global abort
- **messaggio di decisione e acknowledge**
  - scatta il timeout e si ripete la seconda fase

In caso di **caduta del resource manager**, si ripristina il server e si controlla il log:

- se l'ultima scrittura di una transazione è **ready**
  - si attende che il transaction manager comunichi l'esito della transazione
  - oppure si chiede esplicitamente l'esito
- **altrimenti**
  - si procede con una normale ripartenza a caldo

Scarselli Franco

Sistemi per basi di dati 2006-2007

229



# Ripristino II

In caso di **caduta del transaction manager**, si ripristina il server e si controlla il log:

- se l'ultima scrittura di una transazione è **prepare**
  - si decide per un **global abort** e si procede alla seconda fase (un resource manager deve sempre rispondere a piu` messaggi uguali)
  - oppure si tenta di ripetere la prima fase
- se l'ultima scrittura di una transazione è una **global decision**
  - il transaction manager **ripete la seconda fase**
- se l'ultima scrittura di una transazione è **complete**
  - non si fa niente

Scarselli Franco

Sistemi per basi di dati 2006-2007

230





## Ripristino III

Cosa fare se il transaction manager **non puo' essere ripristinato** ?

- se **elegge** un nuovo transaction manager
  - es. il resource manager che ha l' IP piu' grande
  - attenzione alle **partizioni della rete**!!
- Il nuovo transaction manager contatta tutti i resource manager per decidere cosa fare
  - se **qualche nodo** ha un **abort** o un **commit**:  
la decisione era gia' stata presa, si ripete la seconda fase
  - se **nessuno** ha un **abort** o un **commit**, **almeno uno non ha ready**:  
si decide per un global abort
  - se **tutti** hanno un **ready**, **ma nessuno un abort o un commit**:  
gestito manualmente da un operatore!

Scarselli Franco

Sistemi per basi di dati 2006-2007

231



## Estensioni del commit a due fasi

### Abort presunto

- **permette di evitare la scrittura sincrona dei record prepare, global abort e complete**
- caduta del transaction manager
  - mancanza di **prepare**: il transaction manager ripete la prima fase
  - mancanza di **global abort**: il transaction manager risponde per default con global abort
  - mancanza di **complete**, ma c'è il commit: il transaction manager ripete la seconda fase

### Sola lettura

- se un resource manager effettua **solo letture**
  - invia al transaction manager un messaggio **read only**
  - il transaction manager **lo ignora** nel seguito del protocollo
- **I DBMS commerciali**
  - adottano il **commit a due fasi con abort presunto e sola lettura**

Scarselli Franco

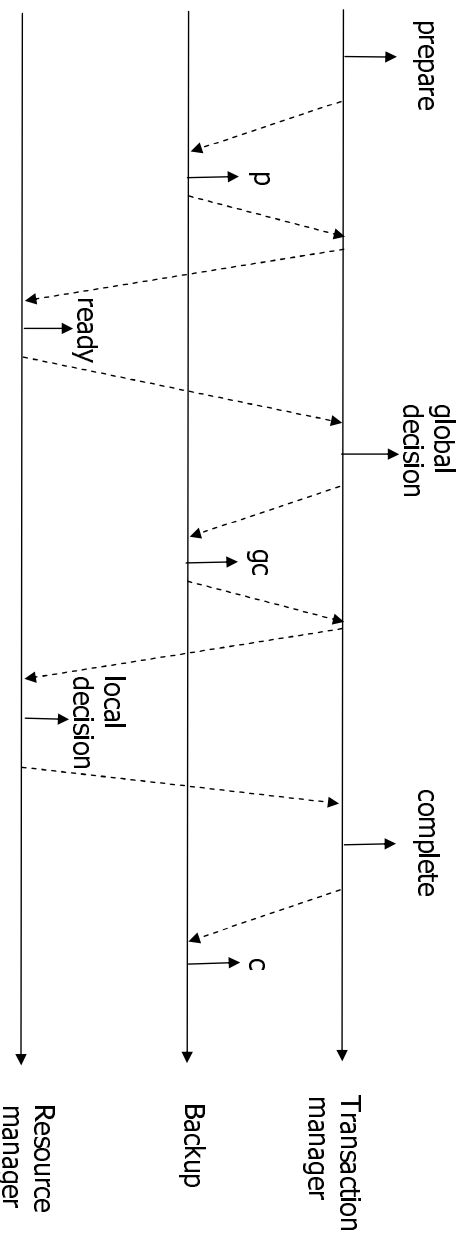
Sistemi per basi di dati 2006-2007

232

# Commit a quattro fasi

## Commit a quattro fasi

- include un server di **backup** che ha il compito di sostituire il transaction manager in caso di caduta
- implementato da Tandem che produce architetture fault tolerant



Scarselli Franco

Sistemi per basi di dati 2006-2007

233

# Interoperabilit  del commit a due fasi

## X-Open DTP (Distributed Transaction Processing)

- permette di realizzare il commit a due fasi con DBMS eterogenei
- consta di due interfacce principali, ognuna delle quali mette a disposizione una serie di procedure
  - l'interfaccia fra client e transaction manager (**TM-interface**)  
tm\_init, tm\_exit, tm\_open, tm\_begin, tm\_commit
  - l'interfaccia fra resource manager e transaction manager (**XA-interface**)  
xa\_open, xa\_close, xa\_start, xa\_end, xa\_precom, xa\_commit, xa\_abort, xa\_recover, xa\_forget
- i resource manager sono totalmente passivi, guidati dal transaction manager
- il client dialoga con il transaction manager per richiedere le transazioni

Scarselli Franco

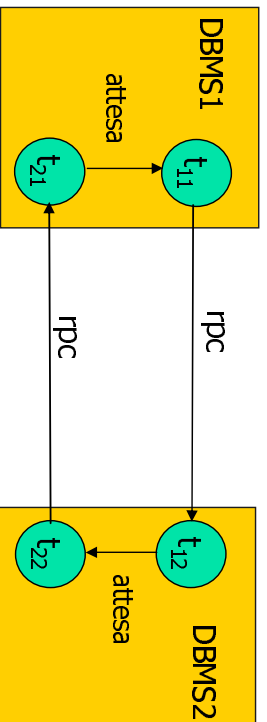
Sistemi per basi di dati 2006-2007

234

# Rilevazione dei deadlock distribuita

La rilevazione dei deadlock a livello di nodo non è sufficiente

- es:
  - $t_{11}$  attende  $t_{12}$  (attivata tramite una chiamata di procedura remota)
  - $t_{12}$  attende una risorsa controllata da  $t_{22}$
  - $t_{22}$  attende  $t_{21}$  (attivata tramite una chiamata di procedura remota)
  - $t_{21}$  attende una risorsa controllata da  $t_{11}$



Tali condizioni dai attesa  
saranno rappresentate come

- $E_1 \rightarrow t_{12} \rightarrow t_{22} \rightarrow E_2$
- $E_2 \rightarrow t_{21} \rightarrow t_{11} \rightarrow E_1$

Scarselli Franco

Sistemi per basi di dati 2006-2007

235

## Rilevazione dei deadlock distribuita II

Alcune soluzioni

- Time out
  - La soluzione piu' semplice
  - Possono essere rilevati deadlock fantasma "phantom deadlocks" a causa di ritardi sulla rete
- Un server è preposto alla rilevazione dei deadlock
  - riceve periodicamente dagli altri server i grafi locali dei conflitti
  - li unisce per decidere se esiste uno stallo
- Soluzione distribuita
  - i server si scambiano i grafi dei conflitti fino a quando uno di essi non è in grado di identificare uno stallo

Scarselli Franco

Sistemi per basi di dati 2006-2007

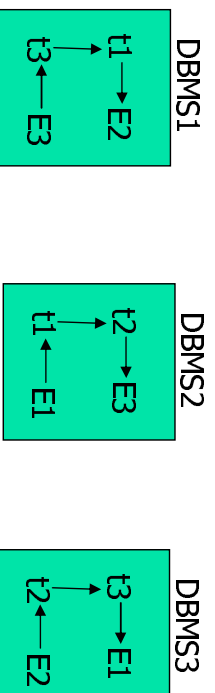
236

# Soluzione distribuita

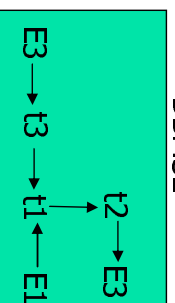
## L'algoritmo

1. Ad ogni nodo si calcolano tutte le condizioni del tipo  
 $E_{in} \rightarrow t_i \rightarrow t_j \rightarrow E_{out}$
2. Si invia al nodo out (in avanti) tutte le condizioni per cui  $i > j$   
(serve ad evitare che piu' nodi possano scoprire lo stesso deadlock)
3. ogni nodo svolge la ricerca di deadlock
4. si ritorna ad 1

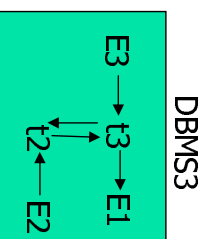
## Soluzione distribuita: un esempio



DBMS1 spedisce a DBMS2  
 $E3 \rightarrow t3 \rightarrow t1 \rightarrow E2$



DBMS2 spedisce a DBMS3  
 $E3 \rightarrow t3 \rightarrow t2 \rightarrow E3$





# Basi di dati replicate

I dati di un archivio possono essere replicati per

- garantire la disponibilità dei dati in caso di malfunzionamento
- migliorare la velocità di accesso

La replicazione permette di creare copie fra server diversi di tutto un database o di fragmenti dei dati

- **copia verticale**  
si copiano alcune colonne di una tabella, ad esempio il nome e cognome di ogni cliente
- **copia orizzontale**  
si copiano alcune righe di una tabella, ad esempio tutti gli studenti lavoratori

Scarselli Franco

Sistemi per basi di dati 2006-2007

239



# Basi di dati replicate II

## Replicazione sincrona

- I dati sono replicati in maniera sincrona, garantendo che ogni server contenga esattamente sempre la stessa informazione
- Ogni modifica ai dati viene operata su tutti le copie, magari utilizzando un commit a due fasi
- non funziona molto bene, se i server sono lontani e le transazioni distribuite sono difficili

## Replicazione asincrona

- le modifiche vengono propagate in maniera asincrona
- i dati sono replicati ad intervalli regolari o in maniera continua
- i server possono contenere dati diversi, ma con il passare del tempo i dati tendono a diventare uguali

Scarselli Franco

Sistemi per basi di dati 2006-2007

240



# Basi di dati replicate III

## Server primario e server secondari

- esiste un server primario che è proprietario di alcuni fragmenti dei dati: solo il server primario può modificare i dati che esso pubblica
- i server secondari si registrano presso il server primario: ricevano una copia dei dati e delle eventuali modifiche
- un server primario per un certo fragmento, può essere secondario per un altro

## Peer-to-peer

- si permette a più server di modificare gli stessi dati
- occorre definire un meccanismo di risoluzione dei conflitti: così in cui una dato viene modificato da due server
- la risoluzione dei conflitti è e spesso basata su regole ad hoc

Scarselli Franco

Sistemi per basi di dati 2006-2007

241



# Basi di dati replicate IV

Nelle basi di dati replicate si fanno due tipi di copie: snapshot e transazionale

- lo snapshot consiste di una copia dei dati contenuti nel fragmento
- una copia transazionale contiene le transazioni fatte su un certo fragmento
- Inizialmente, il serve primario fa e distribuisce uno snapshot
- Successivamente, può distribuire le copie transazionali: in questo caso, i server secondari applicano le transizioni ai propri dati

Le copie transazionali possono essere catturate

- leggendo il log (solo copie transazionali)
- con dei trigger che si attivano quando viene fatta una modifica sul database

Scarselli Franco

Sistemi per basi di dati 2006-2007

242