ARCHITETTURE DISTRIBUITE

Vincenzo Calabrò

Paradigmi per la distribuzione dei dati

- Architetture client-server
  - separazione del database server dal client
- Database distribuiti
  - più database server utilizzati dalla stessa applicazione
- Database paralleli
  - più device di memorizzazione e processori che operano in parallelo per migliorare le performance
- Database replicati
  - stessi dati fisicamente memorizzati su più server
- Data warehouse
  - server specializzati per la gestione di dati dedicati al supporto alle decisioni

Tipologie di architetture

Due tipologie di sistemi

- OLTP (On-Line Transaction Processing)
  - rivolti alla gestione ottimizzata di transazioni affidabili su database server
  - specializzati per gestire centinaia o anche migliaia di transazioni al secondo
- OLAP (On-Line Analytical Processing)
  - rivolti alla analisi dei dati
  - operano su server di data warehouse, specializzati per la gestione di dati per sistemi di supporto alle decisioni

I server al loro interno contengono tipicamente sia funzioni OLTP sia funzioni OLAP

Proprietà dei sistemi distribuiti

Portabilità

- possibilità di trasportare programmi da un ambiente ad un altro
  - stabilita al tempo di compilazione
  - facilitata da linguaggi standard (es., SQL-2, SQL-3)

Interoperabilità

- capacità di interagire tra sistemi eterogenei
  - stabilita al tempo di esecuzione
  - facilitata da protocolli standard per l'accesso ai dati (es., *Database Connectivity* (*ODBC*) e *X-Open Distributed Transaction Processing* (*DTP*))

Architettura client-server (1)

Processi software che interagiscono sono divisi tra client (che richiedono servizi) e server (che forniscono servizi)

- richiede una definizione precisa di interfaccia di servizi, che elenca i servizi offerti dal server
- il client ha un ruolo attivo (richiede)
- il server ha un ruolo reattivo (risponde)
- generalmente, un processo client richiede pochi servizi in sequenza ad uno o più processi server
- generalmente, un processo server risponde a più richieste di molti processi client

Architettura client-server (2)

- il computer dedicato al client deve
  - essere appropriato all'interazione con l'utente
  - supportare strumenti di produttività (posta elettronica, word processing, fogli elettronici, accesso Internet, e workflow management)

- il computer server deve
  - avere una grande quantità di memoria (per supportare la gestione del buffer)
  - avere elevata capacità disco (per memorizzare l'intera base di dati)

Architettura client-server (3)

Abbastanza diffusa per basi di dati:

- le funzioni di client e server sono ben definite
- fornisce una conveniente separazione di attività di gestione
  - client adatto all'interazione con l'utente
  - server adatto alla gestione dei dati
- SQL offre un paradigma di programmazione ideale per l'identificazione dell'interfaccia dei servizi

Architettura client-server (4)

SQL offre un paradigma di programmazione ideale per la identificazione dell'interfaccia dei servizi

- interrogazioni SQL formulate dal client e mandate al server
- i risultati delle interrogazioni sono calcolati dal server e mandati al client
- la standardizzazione, portabilità, e interoperabilità di SQL permette lo sviluppo di applicazioni client che coivolgono sistemi server differenti

Architettura client-server (5)

Spesso il server è multi-threaded:

- si comporta come un processo singolo che lavora dinamicamente per conto di transazioni differenti
- ogni unità di esecuzione del processo server per una data transazione è chiamata thread

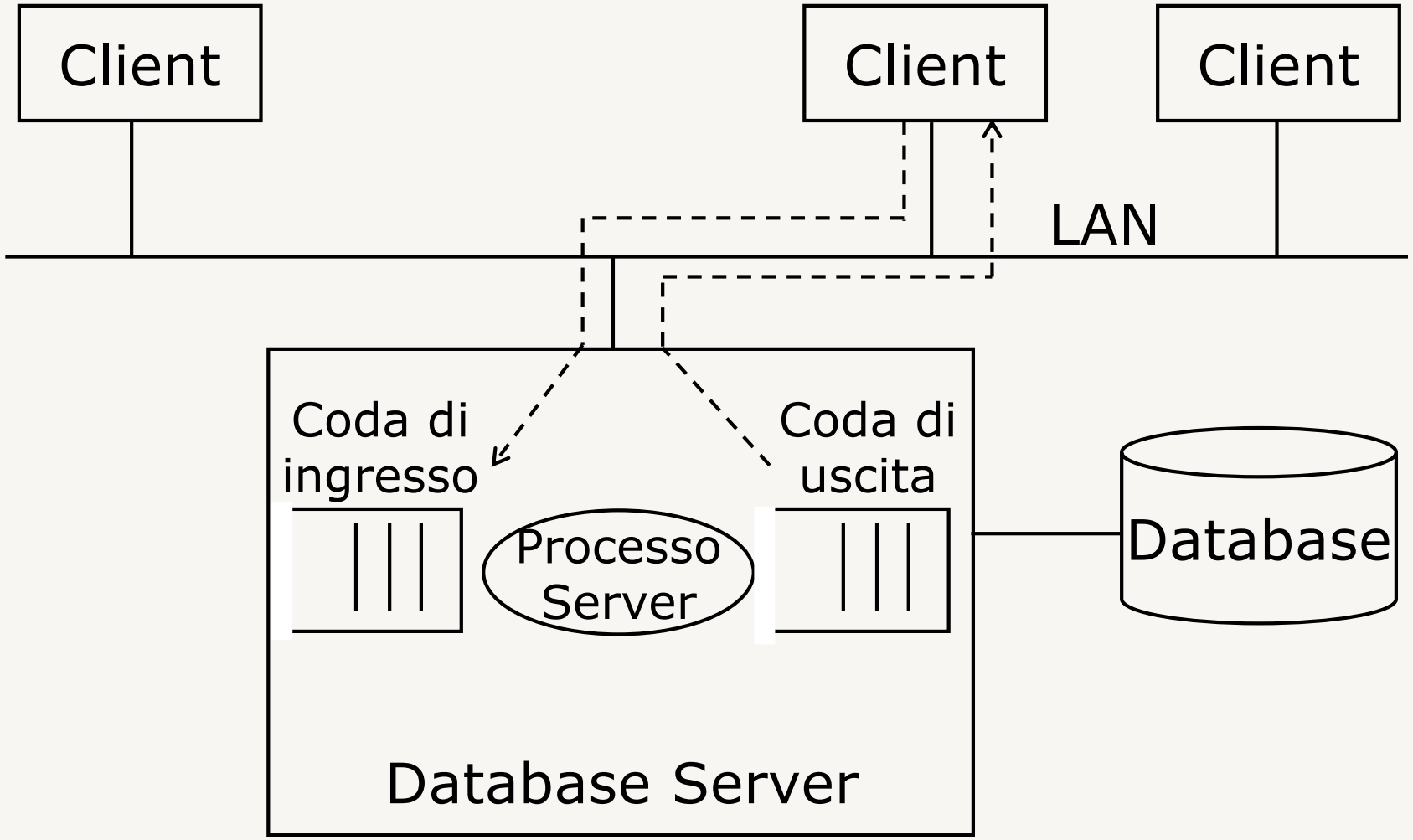
Architettura client-server (6)

- i server sono processi permanentemente attivi che controllano:
  - una coda di input per le richieste dei client
  - una coda di output per i risultati delle interrogazioni

- spesso, un processo dispatcher distribuisce richieste ai server e ritorna le risposte ai client

- quando i dispatcher possono definire dinamicamente il numero di processi server attivi in funzione del numero di richieste ricevute si dice che una classe di server è disponibile

Architettura client-server (7)



Architetture two-tier e three-tier

Two-tier

- il client funziona sia da interfaccia utente sia da gestore di applicazioni
  - il client supporta la logica delle applicazioni (thick client)

Three-tier

- un secondo server, chiamato application server, è responsabile per la gestione delle applicazioni comuni a molti client
  - il client è responsabile solo per l'interfaccia con l'utente finale (thin client) e può essere sviluppato attraverso un browser

Basi di dati distribuite

Sistema di base di dati nel quale almeno un client interagisce con più server per l'esecuzione di una applicazione

Basi di dati distribuite: vantaggi

- rispondono ai requisiti delle applicazioni
  - le organizzazioni sono strutturalmente distribuite
  - la distribuzione dei dati permette la loro gestione dove essi sono generati e utilizzati
- flessibilità e modularità
  - possono essere configurate con aggiunte e modifiche progressive dei componenti
- affidabilità
  - possono rispondere ai guasti con una riduzione delle prestazioni anziché con un blocco completo

Basi di dati distribuite: classificazione (1)

- tipo di DBMS coinvolti
  - DDB omogenea: tutti i server utilizzano lo stesso DBMS
  - DDB eterogenea: i server utilizzano diversi DBMS

- tipo di rete
  - Local Area Network (LAN)
  - Wide Area Network (WAN)

Basi di dati distribuite: classificazione (2)

Tipo di DBMS	Tipo di rete	
	LAN	**WAN**
Omogeneo	Applicazioni gestionali e finanziarie	Sistemi di prenotazione e applicazioni finanziarie
Eterogeneo	Applicazioni gestionali e interfunzionali	Sistemi di prenotazione integrati, sistemi interbancari

Autonomia locale e cooperazione

La base di dati distribuita può essere considerata dal punto di vista astratto come una unica base di dati

- deve essere progettata in modo da cercare di avere applicazioni che vengono eseguite su un singolo server, limitando
  - necessità di interazione
  - necessità di trasporto dei dati

Frammentazione dei dati (1)

Applica operazioni algebriche su una relazione *R* per dividerla in frammenti $R_1, \ldots, R_n$

- frammentazione orizzontale
  - ogni R_i ha come tuple un sottoinsieme delle tuple di *R*
  - ogni R_i può essere interpretato come il risultato di una selezione su *R*
- frammentazione verticale
  - ogni R_i ha come schema un sottoinsieme degli attributi di *R*
  - ogni R_i può essere interpretato come il risultato di una proiezione su *R*

Frammentazione dei dati (2)

Proprietà di correttezza

- completezza: ogni dato di R deve essere presente in un qualche suo frammento R_i
- ricostruibilità: R deve essere interamente ricostruibile a partire dai suoi frammenti

Generalmente

- i frammenti orizzontali sono disgiunti
  - non hanno tuple in comune
- i frammenti verticali includono la chiave primaria di R
  - garantisce ricostruibilità

Frammentazione orizzontale: esempio (1)

IMPIEGATO(Empnum,Nome,Dipnum,Salario,Tasse)

Frammenti:

- IMPIEGATO1 = $\sigma_{Empnum \leq 3}$ IMPIEGATO
- IMPIEGATO2 = $\sigma_{Empnum > 3}$ IMPIEGATO

Per ricostruire la relazione:

- IMPIEGATO = IMPIEGATO1 $\cup$ IMPIEGATO2

Frammentazione orizzontale: esempio (2)

IMPIEGATO

Empnum	**Nome**	**Dipnum**	**Salario**	**Tasse**
1	Roberto	Produzione	3.7M	1.2M
2	Giovanni	Amministrazione	3.5M	1.1M
3	Anna	Produzione	5.3M	2.1M
4	Carlo	Marketing	3.5M	1.1M

IMPIEGATO1 ($\sigma_{Empnum \leq 3}$ IMPIEGATO)

Empnum	**Nome**	**Dipnum**	**Salario**	**Tasse**
1	Roberto	Produzione	3.7M	1.2M
2	Giovanni	Amministrazione	3.5M	1.1M
3	Anna	Produzione	5.3M	2.1M

IMPIEGATO2 ($\sigma_{Empnum > 3}$ IMPIEGATO)

Empnum	**Nome**	**Dipnum**	**Salario**	**Tasse**
4	Carlo	Marketing	3.5M	1.1M

Frammentazione verticale: esempio (1)

IMPIEGATO(Empnum,Nome,Dipnum,Salario,Tasse)

Frammenti:

- IMPIEGATO1 = $\Pi_{Empnum,Nome}(IMPIEGATO)$
- IMPIEGATO2 = $\Pi_{Empnum,Dipnum,Salario,Tasse}(IMPIEGATO)$

Per ricostruire la relazione:

- IMPIEGATO = IMPIEGATO1 $\bowtie$ IMPIEGATO2

Frammentazione verticale: esempio (2)

IMPIEGATO

Empnum	Nome	Dipnum	Salario	Tasse
1	Roberto	Produzione	3.7M	1.2M
2	Giovanni	Amministrazione	3.5M	1.1M
3	Anna	Produzione	5.3M	2.1M
4	Carlo	Marketing	3.5M	1.1M

$$\text{IMPIEGATO1} = \Pi_{\text{Empnum,Nome}}(\text{IMPIEGATO})$$

$$\text{IMPIEGATO2} = \Pi_{\text{Empnum,Dipnum,Salario,Tasse}}(\text{IMPIEGATO})$$

IMPIEGATO1

Empnum	Nome
1	Roberto
2	Giovanni
3	Anna
4	Carlo

IMPIEGATO2

Empnum	Dipnum	Salario	Tasse
1	Produzione	3.7M	1.2M
2	Amministrazione	3.5M	1.1M
3	Produzione	5.3M	2.1M
4	Marketing	3.5M	1.1M

Schemi di allocazione

Descrive il mapping delle relazioni o dei frammenti ai server che li memorizzano

- ogni frammento corrisponde a un file a livello fisico ed è allocato su uno specifico server
  - i frammenti sono memorizzati
  - la relazione globale è una vista sui frammenti (virtuale)

- l'allocazione può essere
  - non-ridondante: ogni frammento o relazione è allocato su un solo server
  - ridondante: almeno un frammento o relazione è allocato su più di un server

Livelli di trasparenza (1)

Distinzione fra frammentazione e allocazione consente di scrivere applicazioni a diversi livelli

- dal più astratto e indipendente dalla frammentazione dei dati
- al più concreto dipendente anche dalla loro allocazione fisica

Livelli di trasparenza (2)

- frammentazione: il programmatore
  - non ha bisogno di conoscere la frammentazione
  - non ha bisogno di conoscere l'allocazione
- allocazione: il programmatore
  - deve conoscere la struttura dei frammenti
  - non ha bisogno di conoscere l'allocazione
- linguaggio: il programmatore
  - deve conoscere la struttura dei frammenti
  - deve conoscere l'allocazione
- assenza di trasparenza
  - ogni DBMS accetta il suo proprio 'dialetto' SQL: il sistema è eterogeneo e i DBMS non supportano uno standard comune per l'interoperabilità

Livelli di trasparenza: esempio

FORNITORE(Fnum,Nome,Città)

- frammenti orizzontali:
  - FORNITORE1 = $\sigma_{\text{città='Milano'}}$ (FORNITORE)
  - FORNITORE2 = $\sigma_{\text{città='Roma'}}$ (FORNITORE)
- allocazione dei frammenti orizzontali (con replicazione):
  - FORNITORE1@ditta.milano.it
  - FORNITORE2@ditta.roma1.it
  - FORNITORE2@ditta.roma2.it

vogliamo scrivere una procedura che dato un numero di fornitore ne ritorna il nome ...

Trasparenza di frammentazione: es.

- il programmatore
  - non ha bisogno di conoscere la frammentazione
  - non ha bisogno di conoscere l'allocazione

```
procedure Query1(:fnum,:nome);
      select Nome into :nome
      from Fornitore
      where Fnum = :fnum;
end procedure;
```

Trasparenza di allocazione: es.

- il programmatore
  - deve conoscere la struttura dei frammenti
  - non ha bisogno di conoscere l'allocazione
  - in caso di ridondanze, non deve indicare quale copia è scelta per l'accesso (trasparenza di replicazione)

```
procedure Query2(:fnum,:nome);
      select Nome into :nome
      from Fornitore1
      where Fnum = :fnum;
if :empty then
      select Nome into :nome
      from Fornitore2
      where Fnum = :fnum;
end procedure;
```

Trasparenza di linguaggio: es.

- il programmatore
  - deve conoscere la struttura dei frammenti
  - deve conoscere l'allocazione
  - in caso di ridondanze, deve indicare quale copia è scelta per l'accesso

```
procedure Query3(:fnum,:nome);
      select Nome into :nome
      from Fornitore1@ditta.milano.it
      where Fnum = :fnum;
if :empty then
      select Nome into :nome
      from Fornitore2@ditta.roma1.it
      where Fnum = :fnum;
end procedure;
```

Ottimizzazioni delle interrogazioni

L'applicazione può essere ottimizzata tramite:

- parallelismo
  - si sottomettono richieste in parallelo invece che in sequenza
  - diminuisce il tempo di risposta globale
- conoscenza sulle proprietà logiche dei frammenti
  - query al frammento dove i dati si trovano
  - aumenta efficienza ma diminuisce flessibilità

Ottimizzazioni delle interrogazioni: es.

```
procedure Query4(:fnum,:nome,:città);
case :città of
     "Milano":
          select Nome into :nome
          from Fornitore1
          where Fnum = :fnum;
     "Roma":
          select Nome into :nome
          from Fornitore2
          where Fnum = :fnum;
end procedure;
```

Classificazione di transazioni

Livelli progressivi di complessità

- richieste remote
- transazioni remote
- transazioni distribuite
- richieste distribuite

Richieste remote

- transazioni read-only composte di un numero arbitrario di query SQL
- tutte le query sono indirizzate a un singolo DBMS remoto
- il DBMS remoto può essere solo interrogato

Transazioni remote

- composte da un numero qualsiasi di comandi SQL (select, insert, delete, update)
- tutti i comandi sono indirizzati a un singolo DBMS remoto
- ogni transazione scrive su un singolo DBMS

Transazioni distribuite

- composte da un numero qualsiasi di comandi SQL (select, insert, delete, update) indirizzati a un numero arbitrario di DBMS remoti
- ogni comando SQL si riferisce a un singolo DBMS
- ogni transazione può aggiornare più di un DBMS
  - richiede protocollo two-phase-commit

Richieste distribuite

- transazioni arbitrarie composte da un numero qualsiasi di comandi SQL (select, insert, delete, update) indirizzati a un numero arbitrario di DBMS remoti
- ogni comando si può riferire a qualsiasi DBMS
- richiede un ottimizzatore di query distribuite

Transazione: esempio (1)

CC(Num,Nome,Saldo)

- CC1: $\sigma_{Num \leq 1000}(CC)$
- CC2: $\sigma_{Num > 1000}(CC)$
- assumiamo trasparenza di allocazione

Transazione distribuita

- trasferimento di 100 Euro dal conto 354 al conto 1487

Nota

è necessario garantire atomicità: o entrambe le modifiche sono eseguite o non lo è nessuna

Transazione: esempio (2)

```
begin transaction

      update CC1
      set Saldo = Saldo - 100
      where CCNum = ‘354’;

      update CC2
      set Saldo = Saldo + 100
      where CCNum = ‘1487’;

commit;
end transaction
```

Tecnologia di basi di dati distribuite (1)

La distribuzione dei dati non influenza

- consistenza: i vincoli di integrità descrivono solo proprietà locali
  - è un limite della corrente tecnologia DBMS
- persistenza: ogni sistema garantisce persistenza ai dati localmente memorizzati
  - meccanismi di recovery (log, checkpoint, dump) locali

La distribuzione dei dati influenza

- isolamento
- atomicità

Tecnologia di basi di dati distribuite (2)

La distribuzione dei dati richiede modifiche a

- ottimizzazione di interrogazioni
- controllo di concorrenza
  - isolamento
- controllo di affidabilità
  - atomicità

Ottimizzatore di query distribuite (1)

Sotto la responsabilità del DBMS che riceve la query

- decide la suddivisione della query in sub-query, ognuna indirizzata a un DBMS specifico
- costruisce una strategia (piano) di esecuzione distribuita:
  - esecuzione coordinata di vari programmi a vari DBMS
  - scambio di dati fra DBMS
- assicura ottimizzazione globale

Ottimizzatore di query distribuite (2)

Nel calcolo del costo delle interrogazioni distribuite ha particolare importanza la quantità di dati trasmessi sulla rete

$$C_{totale} = C_{I/O} \times n_{I/O} + C_{CPU} \times n_{CPU} + C_{tr} \times n_{tr}$$

- n_{tr} : quantità di dati trasmessi in rete
- C_{tr} : costo di trasmissione

Controllo di concorrenza

In un sistema distribuito, una transazione t_i può eseguire più sotto-transazioni a diversi nodi:

- t_{ij} esecuzione di t_i al nodo j

Esempio

- $t_1 : r_{11}(x)\ w_{11}(x)\ r_{12}(y)\ w_{12}(y)$
- $t_2 : r_{22}(y)\ w_{22}(y)\ r_{21}(x)\ w_{21}(x)$

Controllo di concorrenza: esempio

$S_1 : r_{11}(x)\ w_{11}(x)\ r_{21}(x)\ w_{21}(x)$

$S_2 : r_{22}(y)\ w_{22}(y)\ r_{12}(y)\ w_{12}(y)$

- localmente serializzabili (seriali)
- globalmente non serializzabili
  - il grafo dei conflitti ha un ciclo:
    - sul nodo 1, t_1 precede t_2 ed è in conflitto con t_2
    - sul nodo 2, t_2 precede t_1 ed è in conflitto con t_1

Atomicità

Atomicità di transazioni distribuite può essere compromessa da guasti/malfunzionamenti

- guasti a nodi (software/hardware)
- perdite di messaggi: lasciano l'esecuzione di un protocollo in uno stato non certo
  - ogni messaggio del protocollo (*msg*) è seguito da un messaggio di conferma di ricezione (*ack*)
  - la perdita di uno dei messaggi lascia il mittente insicuro circa la sua ricezione
- guasti a link di comunicazione: possono causare partizionamenti della rete
  - una transazione può essere simultaneamente attiva in più di una sotto-rete

Serializzabilità globale

Serializzabilità locale non garantisce serializzabilità globale

Esempio

- $S_1 : r_{11}(x)\ w_{11}(x)\ r_{21}(x)\ w_{21}(x)$
  $S_2 : r_{22}(y)\ w_{22}(y)\ r_{12}(y)\ w_{12}(y)$
  - localmente serializzabili (seriali)
  - globalmente non serializzabili

La serializzabilità globale richiede l'esistenza di uno schedule seriale *S* equivalente a tutti gli schedule locali S_i risultanti a ogni nodo

- la proiezione di *S* sul nodo *i* deve essere uguale a S_i

Serializzabilità globale: proprietà

Schedule globale conflict-serializzabile

- garantita se ogni scheduler usa 2PL stretto e esegue il commit in modo atomico quando tutte le sotto-transazioni ai vari nodi hanno tutte le risorse

Schedule globale seriale

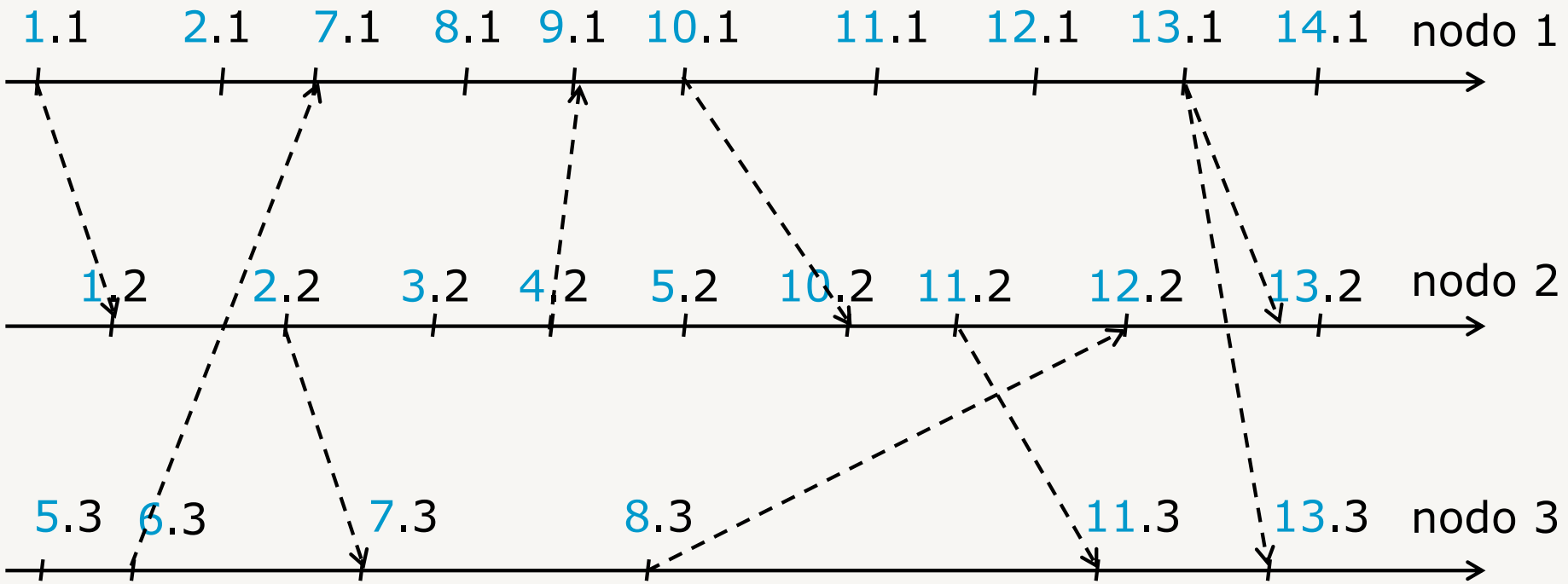
- garantita se ogni transazione distribuita acquisisce un singolo timestamp e lo usa in tutte le richieste a tutti gli scheduler che fanno controllo della concorrenza basato su timestamp
  - richiede assegnamento di timestamp globale

Metodo di Lamport

Permette di assegnare timestamp che riflettano la precedenza fra eventi in un sistema distribuito

- timestamp ha due gruppi di cifre
  - meno significative identificano un nodo
  - più significative identificano gli eventi che accadono al nodo (contatore locale)
- ogni volta che due nodi si scambiano un messaggio, i timestamp vengono sincronizzati:
  - l'evento ricevente deve avere un timestamp maggiore o uguale dell'evento mittente
    - può richiedere l'incremento del contatore locale al nodo ricevente

Metodo di Lamport : esempio

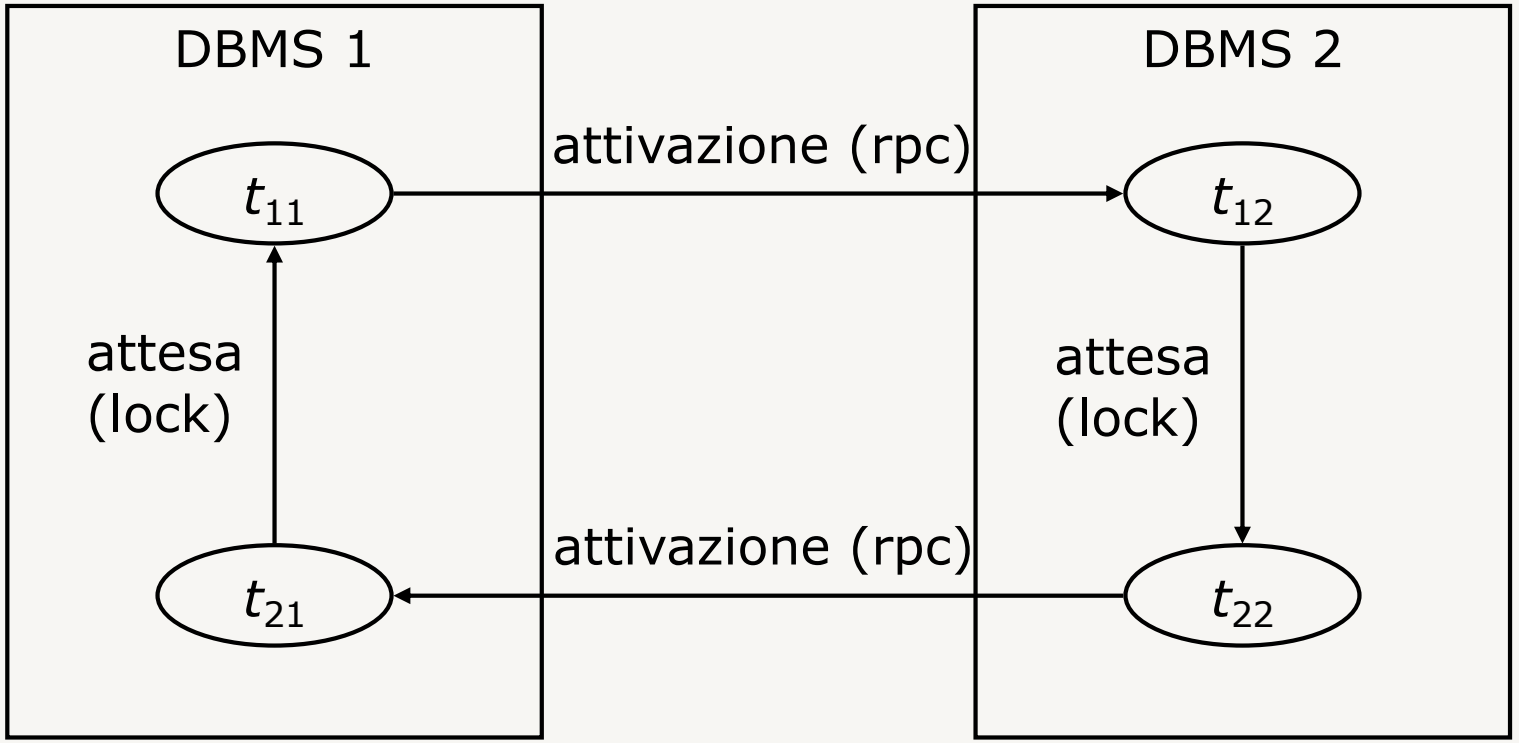


Deadlock distribuiti

Situazioni circolari di attesa tra due o più nodi, soluzioni

- time-out
  - utilizzata da DBMS distribuiti
- rilevazione di deadlock e risoluzione
  - può essere fatta con un protocollo asincrono e distribuito
  - assumiamo che le sottotransazioni siano attivate usando una remote procedure call (rpc)

Deadlock distribuiti: esempio



t_{11} attende t_{12}, attivata tramite una chiamata a procedura remota;

t_{12} attende una risorsa controllata da t_{22};

t_{22} attende t_{21}, attivata tramite una chiamata a procedura remota;

t_{21} attende una risorsa controllata da t_{11}.

Rappresentazione di condizioni di attesa

Condizioni di attesa (anche transitiva) a ogni DBMS rappresentate con:

- $E_{in} \rightarrow t_i \rightarrow t_j \rightarrow E_{out}$

Esempio

- DBMS1: $E_2 \rightarrow t_2 \rightarrow t_1 \rightarrow E_2$
- DBMS2: $E_1 \rightarrow t_1 \rightarrow t_2 \rightarrow E_1$

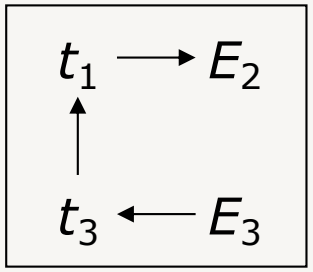
Rilevazione di deadlock distribuiti

Attivata periodicamente ai vari DBMS del sistema, ogni DBMS

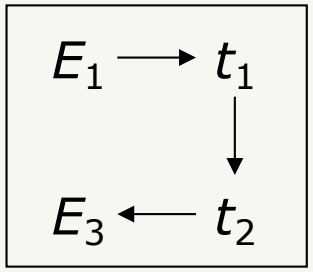
- integra nuove sequenze di attesa con le condizioni di attesa locali
- analizza il grafo risultante per rilevare deadlock
- comunica le sequenze di attesa ad altri DBMS
  - per evitare replicazione:
    - comunicazioni in “avanti” (dove è attiva la sotto-transazione attesa da t_j)
    - solo se $i > j$

Rilevazione di deadlock distribuiti: es.

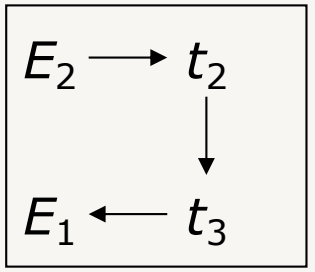
DBMS1



DBMS2

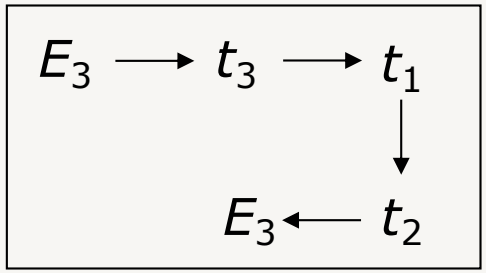


DBMS3



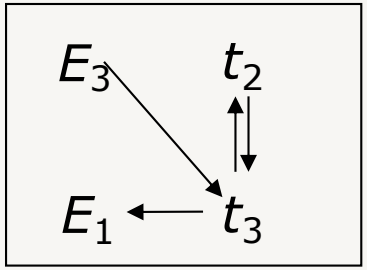
DBMS1 comunica al DBMS2 ($E_3 \rightarrow t_3 \rightarrow t_1 \rightarrow E_2$)

DBMS2



DBMS2 comunica al DBMS3 ($E_3 \rightarrow t_3 \rightarrow t_2 \rightarrow E_3$)

DBMS3



deadlock!

Protocolli per il commit distribuito

Permettono a una transazione di raggiungere la decisione corretta di commit o abort su tutti i nodi che partecipano alla transazione

- il più diffuso è il two-phase commit

Protocollo two-phase commit (1)

- la decisione commit/abort presa dalle parti è registrata da una terza parte, che ratifica la decisione; distingue:
  - i server che partecipano alla decisione: gestori di risorse (RM)
  - processo coordinatore: gestore della transazione (TM)
- funziona tramite scambio rapido di messaggi (broadcast o seriale) tra TM e RM e scritture dei record nei log
- TM e RM mantengono log per assicurare resistenza ai guasti

Protocollo two-phase commit (2)

Log: transaction manager

- `prepare`
  - contiene l'identità di tutti i processi RM (identificatori dei nodi e dei processi)

- `global commit` o `global abort`
  - descrive la decisione globale
  - la decisione del TM è finale al momento della scrittura nel log

- `complete`
  - registra la fine del protocollo di commit a due fasi

Protocollo two-phase commit (3)

Log: resource manager

- `ready`
  - indica la disponibilità irrevocabile a partecipare al protocollo two-phase commit, e quindi di contribuire alla decisione finale
  - può essere scritto solo quando RM si trova in uno stato affidabile, cioè ha il lock su tutte le risorse che devono essere scritte
  - riporta l'identificatore (di processo e di nodo) del TM

- `begin`, `insert`, `delete`, e `update`
  - della transazione locale

Protocollo two-phase commit (4)

- quando un RM si dichiara `ready` per una transazione, perde la propria autonomia e deve rimanere soggetto alla decisione del TM
  - finestra di incertezza (deve essere lasciata al minimo)
  - le risorse acquisite dalla transazione sono bloccate

- prima della dichiarazione della decisione o se si è dichiarato `non-ready`, RM può abortire autonomamente facendo undo dei suoi effetti, senza partecipare al protocollo two-phase commit

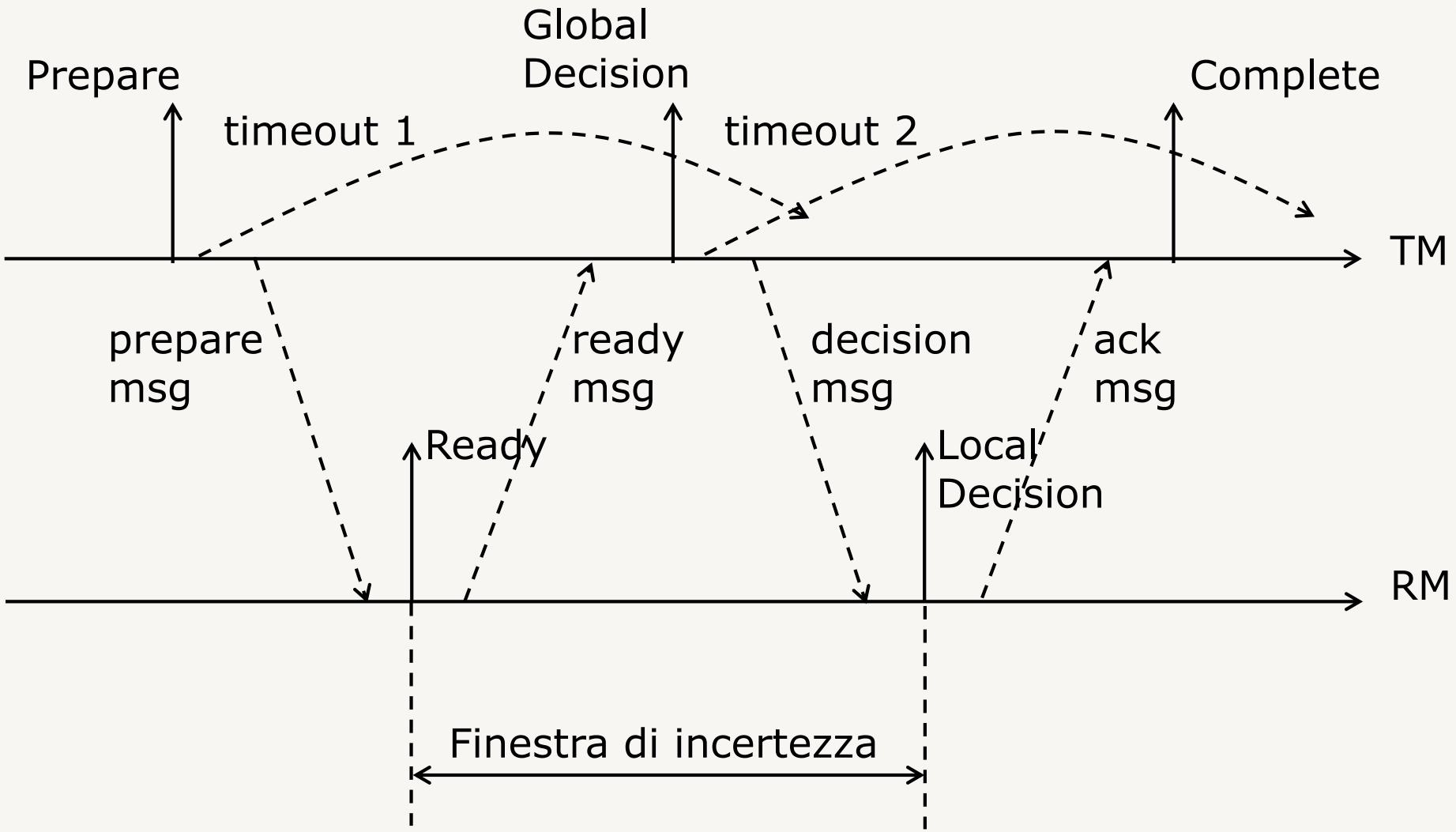
Protocollo: prima fase

- TM
  - scrive il record `prepare` nel suo log e manda un messaggio `prepare` a tutti gli RM; imposta timeout
- ogni RM
  - se in stato affidabile: scrive nel log il record `ready` e trasmette al TM il messaggio `ready`, che indica la scelta di partecipare al protocollo
  - se in stato non affidabile: manda un messaggio `non-ready` e termina la propria partecipazione al protocollo
- TM
  - raccoglie i messaggi di risposta e scrive log
    - `global commit` se tutti gli RM hanno risposto `ready`
    - `global abort` se almeno un RM ha risposto `non-ready` o timeout scattato e non tutti i messaggi ricevuti

Protocollo: seconda fase

- TM
  - trasmette la sua decisione globale (`commit` o `abort`) a tutti gli RM; imposta timeout
- ogni RM in stato `ready`
  - scrive nel log il record relativo alla decisione globale e manda un acknowledgement (`ack`) al TM
  - esegue in loco decisione globale
- TM
  - raccoglie tutti gli `ack` dagli RM coinvolti nella seconda fase
  - se timeout scade, stabilisce un altro timeout e ripete messaggio a tutti gli RM dai quali non ha ricevuto ack
  - quando tutti gli ack sono arrivati, scrive il record `complete` nel suo log

Protocollo two-phase commit



Gestione dei guasti e ottimizzazioni

- un RM in stato ready perde la sua autonomia in attesa della decisione del TM
  - finestra di incertezza
- eventuali guasti possono compromettere la corretta esecuzione del protocollo

⇒ protocolli di ripristino dai guasti

⇒ ottimizzazioni per gestione dei guasti e della finestra di incertezza

Protocolli di ripristino (2)

Ripristinano la correttezza dello stato dei nodi a seguito caso di guasti durante l'esecuzione del protocollo two-phase commit

- caduta di un RM
- caduta del TM
- perdita di messaggi
- partizionamento della rete

Caduta di un RM

Ripresa a caldo, dipende da ultimo record di log

- `abort` o azione: undo della transazione
- `commit`: redo della transazione
- `ready`: guasto successo durante il two-phase commit; bisogna chiedere al TM

Durante la ripresa a caldo:

- gli identificatori delle transazioni in dubbio sono raccolte in un insieme READY
- per ognuna di queste la decisione finale deve essere richiesta al TM

Caduta del TM

Ripresa a caldo, dipende da ultimo record di log

- `prepare`: alcuni RM potrebbero essere in uno stato di blocco; due opzioni
  - scrivere un `global abort` nel log, e eseguire la seconda fase del protocollo
  - ripetere la prima fase
- `global commit/abort`: alcuni RM potrebbero essere stati correttamente informati mentre altri potrebbero essere ancora bloccati
  - TM deve ripetere la seconda fase
- `complete`: non ha effetti sulla transazione

Perdita di messaggi

- la perdita di un messaggio `prepare` o del successivo `ready` non sono distinguibili dal TM
  - in entrambi casi, il timeout scade e una decisione globale di abortire viene presa

- la perdita di un messaggio di decisione (`commit`/`abort`) o di un `ack` non sono distinguibili
  - in entrambi i casi il timeout della seconda fase scade e la seconda fase è ripetuta

Partizione della rete

Non causa ulteriori problemi

- la transazione avrà successo solo se il TM e gli RM appartengono alla stessa partizione

Ottimizzazioni

Il protocollo two-phase commit è abbastanza oneroso soprattutto a causa delle scritture sincrone (force) richieste su ogni log

I sistemi utilizzano generalmente il protocollo con due ottimizzazioni

- abort presunto
- read only

Abort presunto

Un TM che riceve una richiesta di recovery remoto da un RM incerto e non ha alcuna informazione sulla transazione, ritorna un `global abort` come default

- evita la scrittura (force) dei record `prepare` e `global abort` (ripristino con loro equivale al default)
- record di `complete` non è critico e può essere omesso (al più si ripete la seconda fase)

Read-only

Un partecipante che ha eseguito solo operazioni di lettura nella transazione può dichiararsi `read-only` e lasciare il protocollo

- il coordinatore ignora i partecipanti `read-only` nella seconda fase del protocollo

Alcune varianti

- Commit a quattro fasi
  - il TM è replicato da un processo di backup ad un nodo differente
  - ad ogni fase del protocollo, il TM prima informa il backup della sua decisione e poi la comunica agli RM
  - il backup può sostituire il TM nel caso di guasto

- Commit a tre fasi
  - introduce una terza fase di pre-commit
  - se il TM cade, un partecipante può essere eletto come nuovo TM
  - inutilizzabile in pratica perché allunga la finestra di incertezza

Basi di dati distribuite – Esercizi

Esercizio 1

PRODUZIONE(NumSerie,TipoParte, Modello,Qta,Macchina)
PRELIEVO(NumSerie,Lotto,Cliente,Venditore,Ammontare)
CLIENTE(Nome,Citta,Indirizzo)
VENDITORE(Nome,Citta,Indirizzo)

Progettare la frammentazione orizzontale con i seguenti criteri:

- PRODUZIONE e PRELIEVO divisi in quattro frammenti base al tipo di parte (che assume quattro valori: Tastiera, Schermo, Box-cpu e Cavi), allocati a quattro stabilimenti di produzione (uno per ogni componente) disposti a Torino, Savona, Bergamo e Crema, rispettivamente
- CLIENTE e VENDITORE divisi in tre frammenti dati dalla citta del cliente/venditore e allocati al rispettivo nodo (non c'e' venditore a Savona, i clienti di Savona gestiti da Torino)

Distribuzione Produzione

PRODUZIONE_T := $\sigma_{\text{TipoParte='Tastiera'}}$ PRODUZIONE

PRODUZIONE_S := $\sigma_{\text{TipoParte='Schermo'}}$ PRODUZIONE

PRODUZIONE_B := $\sigma_{\text{TipoParte='Box-cpu'}}$ PRODUZIONE

PRODUZIONE_C := $\sigma_{\text{TipoParte='Cavi}}$ PRODUZIONE

Allocati rispettivamente a Torino, Savona, Bergamo e Crema

Distribuzione Prelievo

$$PRELIEVO\_T := \Pi_{NumSerie,Lotto,Cliente,Venditore,Ammontare}(PRELIEVO \bowtie PRODUZIONE\_T)$$

$$PRELIEVO\_S := \Pi_{NumSerie,Lotto,Cliente,Venditore,Ammontare}(PRELIEVO \bowtie PRODUZIONE\_S)$$

$$PRELIEVO\_B := \Pi_{NumSerie,Lotto,Cliente,Venditore,Ammontare}(PRELIEVO \bowtie PRODUZIONE\_B)$$

$$PRELIEVO\_C := \Pi_{NumSerie,Lotto,Cliente,Venditore,Ammontare}(PRELIEVO \bowtie PRODUZIONE\_C)$$

Allocati rispettivamente a Torino, Savona, Bergamo e Crema

Distribuzione Clienti

$CLIENTI\_TOSV := \sigma_{Citta='Torino'\ OR\ Citta='Savona'}\ CLIENTI$

$CLIENTI\_BG := \sigma_{Citta='Bergamo'}\ CLIENTI$

$CLIENTI\_CR := \sigma_{Citta='Crema'}\ CLIENTI$

Allocati rispettivamente a Torino, Bergamo e Crema

Distribuzione Venditori

VENDITORI_TO := $\sigma_{Citta='Torino'}$ VENDITORI

VENDITORI_BG := $\sigma_{Citta='Bergamo'}$ VENDITORI

VENDITORI_CR := $\sigma_{Citta='Crema'}$ VENDITORI

Allocati rispettivamente a Torino, Bergamo e Crema

Query 1

Determinare la quantità totale prodotta dal componente con numero di serie ‘123456’

Query 1 - trasparenza frammentazione

Determinare la quantità totale prodotta dal componente con numero di serie ‘123456’

```
SELECT Qta
FROM Produzione
WHERE NumSerie = ‘123456’
```

Query 1 - trasparenza allocazione

Determinare la quantità totale prodotta dal componente con numero di serie ‘123456’

```
SELECT Qta FROM Produzione_T
WHERE NumSerie = ‘123456’

if :empty then

SELECT Qta FROM Produzione_S
WHERE NumSerie = ‘123456’

if :empty then

SELECT Qta FROM Produzione_B
WHERE NumSerie = ‘123456’

if :empty then

SELECT Qta FROM Produzione_C
WHERE NumSerie = ‘123456’
```

Query 1 - trasparenza linguaggio

Determinare la quantità totale prodotta dal componente con numero di serie ‘123456’

```
SELECT Qta FROM Produzione_T@Torino
WHERE NumSerie = ‘123456’
if :empty then
SELECT Qta FROM Produzione_S@Savona
WHERE NumSerie = ‘123456’
if :empty then
SELECT Qta FROM Produzione_B@Bergamo
WHERE NumSerie = ‘123456’
if :empty then
SELECT Qta FROM Produzione_C@Crema
WHERE NumSerie = ‘123456’
```

Query 2

Determinare i clienti che hanno comperato qualche lotto dal rivenditore `Bianchi' che ha ufficio a `Crema'

Query 2 - trasparenza frammentazione

Determinare i clienti che hanno comperato qualche lotto dal rivenditore `Bianchi' che ha ufficio a `Crema'

```
SELECT Cliente
FROM Prelievo
WHERE Venditore = 'Bianchi'
```

Query 2 - trasparenza allocazione

Determinare i clienti che hanno comperato qualche lotto dal rivenditore `Bianchi' che ha ufficio a `Crema'

```
SELECT Cliente FROM Prelievo_T
WHERE Venditore = 'Bianchi'
UNION
SELECT Cliente FROM Prelievo_S
WHERE Venditore = 'Bianchi'
UNION
SELECT Cliente FROM Prelievo_B
WHERE Venditore = 'Bianchi'
UNION
SELECT Cliente FROM Prelievo_C
WHERE Venditore = 'Bianchi'
```

Query 2 - trasparenza linguaggio

Determinare i clienti che hanno comperato qualche lotto dal rivenditore `Bianchi' che ha ufficio a `Crema'

```
SELECT Cliente FROM Prelievo_T@Torino
WHERE Venditore = 'Bianchi'
UNION
SELECT Cliente FROM Prelievo_S@Savona
WHERE Venditore = 'Bianchi'
UNION
SELECT Cliente FROM Prelievo_B@Bergamo
WHERE Venditore = 'Bianchi'
UNION
SELECT Cliente FROM Prelievo_C@Crema
WHERE Venditore = 'Bianchi'
```

Query 3

Determinare le macchine utilizzate per la produzione dei pezzi di tipo `Tastiera' venduti al cliente `Rossi'

Query 3 - trasparenza frammentazione

Determinare le macchine utilizzate per la produzione dei pezzi di tipo `Tastiera' venduti al cliente `Rossi'

```
SELECT Macchina
FROM Produzione as Pro
      JOIN Prelievo as Pre
      ON Pro.NumSerie=Pre.NumSerie
WHERE Cliente = 'Rossi' AND TipoParte=`Tastiera'
```

Query 3 - trasparenza allocazione

Determinare le macchine utilizzate per la produzione dei pezzi di tipo `Tastiera' venduti al cliente `Rossi'

```
SELECT Macchina
FROM Produzione_T AS ProT
      JOIN Prelievo_T as PreT
      ON ProT.NumSerie=PreT.NumSerie
WHERE Cliente = 'Rossi'
```

Query 3 - trasparenza linguaggio

Determinare le macchine utilizzate per la produzione dei pezzi di tipo `Tastiera' venduti al cliente `Rossi'

```
SELECT Macchina
FROM Produzione_T@Torino AS ProT
      JOIN Prelievo_T@Torino as PreT
      ON ProT.NumSerie=PreT.NumSerie
WHERE Cliente = 'Rossi'
```

Query 4

Modificare l'indirizzo del rivenditore `Rossi' che si trasferisce da `Via Verdi' di `Torino' a `Via Bramante' di `Crema'

Query 4 - trasparenza frammentazione

Modificare l'indirizzo del rivenditore `Rossi' che si trasferisce da `Via Verdi' di `Torino' a `Via Bramante' di `Crema'

```
UPDATE Venditore
SET Citta = 'Crema', Indirizzo= 'Via Bramante'
WHERE Nome = 'Rossi'
```

Query 4 - trasparenza allocazione

Modificare l'indirizzo del rivenditore `Rossi' che si trasferisce da `Via Verdi' di `Torino' a `Via Bramante' di `Crema'

```
DELETE FROM Venditore_TOSV
WHERE Nome = 'Rossi'

INSERT INTO Venditore_CR
VALUES ('Rossi, 'Crema', 'Via Bramante')
```

Query 4 - trasparenza linguaggio

Modificare l'indirizzo del rivenditore `Rossi' che si trasferisce da `Via Verdi' di `Torino' a `Via Bramante' di `Crema'

```
DELETE FROM Venditore_TOSV@Torino
WHERE Nome = 'Rossi'

INSERT INTO Venditore_CR@Crema
VALUES ('Rossi, 'Crema', 'Via Bramante')
```

Query 5

Calcolare l’ammontare totale di tutti gli ordini ricevuto a Torino, Savona, Bergamo e Crema

Query 5 - trasparenza frammentazione

Calcolare l'ammontare totale di tutti gli ordini ricevuto a Torino, Savona, Bergamo e Crema

```
SELECT sum(Ammontare) AS Totale
FROM Prelievo
```

Query 5 - trasparenza allocazione

Calcolare l'ammontare totale di tutti gli ordini ricevuto a Torino, Savona, Bergamo e Crema

```
CREATE VIEW TotaliAmm(Somma) AS
      SELECT sum(Ammontare) FROM Prelievo_T
      UNION ALL
      SELECT sum(Ammontare) FROM Prelievo_S
      UNION ALL
      SELECT sum(Ammontare) FROM Prelievo_B
      UNION ALL
      SELECT sum(Ammontare) FROM Prelievo_C

SELECT sum(Somma) AS Totale
FROM TotaliAmm
```

Query 5 - trasparenza linguaggio

Calcolare l'ammontare totale di tutti gli ordini ricevuto a Torino, Savona, Bergamo e Crema

```
CREATE VIEW TotaliAmm(Somma) AS
      SELECT sum(Ammontare) FROM Prelievo_T@Torino
      UNION ALL
      SELECT sum(Ammontare) FROM Prelievo_S@Savona
      UNION ALL
      SELECT sum(Ammontare) FROM Prelievo_B@Bergamo
      UNION ALL
      SELECT sum(Ammontare) FROM Prelievo_C@Crema

SELECT sum(Somma) AS Totale
FROM TotaliAmm
```

VINCENZO CALABRÒ

LinkedIn vincenzocalabro

www.vincenzocalabro.it