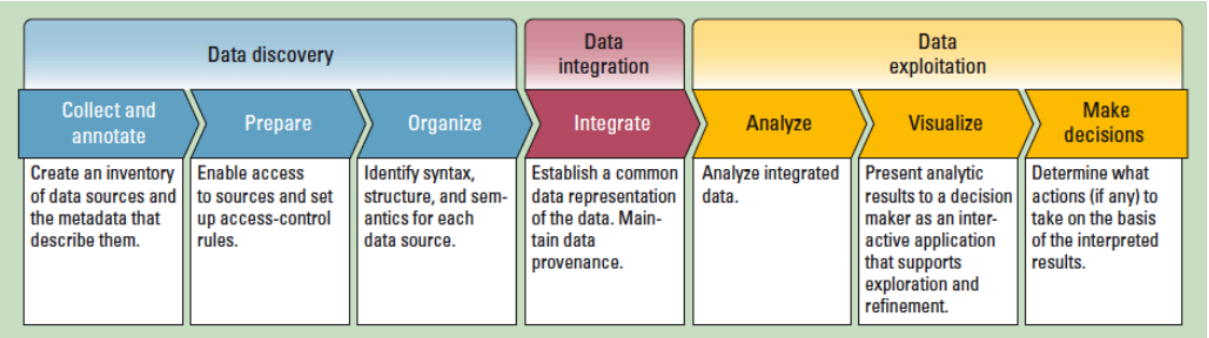
Lezione 1 04/03/2024

Gli scenari applicativi dei database variano per:

- Workload (letture/scritture, poche/tante operazioni, complessità delle operazioni)
- Volumi di dati
- Eterogeneità (più database di tipo diverso)

Bisogna quindi individuare la soluzione migliore per far funzionare il sistema.

Ciclo di vita dei dati



L'obiettivo del corso è quello di fornire allo studente competenze per progettare l'architettura dati più adatta in base al contesto, comprendere i meccanismi di gestione dei moderni sistemi di gestione dati, conoscere i modelli di rappresentazione di dati oltre al modello relazionale, i principali DBMS utilizzati e i principi di management dei dati.

Use case distribuito relazionale

Lo scenario è quello dove dobbiamo gestire le operazioni bancarie come transazioni bancarie e bonifici multi banca.

Workload:

- In questo caso le letture/scritture sono più meno bilanciate, non è un caso limite.
- Le operazioni sono tante

- Le query SQL sono semplici, ma a causa del fatto che ci sono tante persone che fanno tante query, la cosa diventa complicata.

Una metrica di qualità per un database di questo tipo è il tempo di esecuzione delle query.

I dati saranno tanti, i sistemi saranno eterogenei dato che le banche nascono da fusioni e possono rimanere anche più di un sistema informativo. Probabilmente avremo quindi bisogno di un sistema distribuito.

Le scritture possono avere problemi come la concorrenza, perdita di dati.

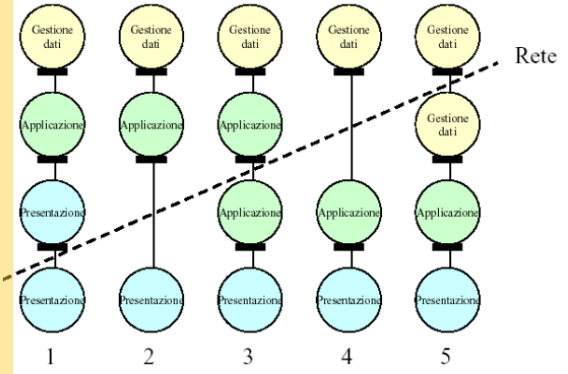
Se scrivo su due server diversi, distribuiti geograficamente, questo richiede tempo perché devo aspettare la conferma da entrambi.

Potrei fare operazioni su un database e usare i file di log per ripetere le operazioni sugli altri.

Quindi è necessario un middleware che gestisca il secondo database, in questo caso è sufficiente una coda.

Base di dati distribuita

- Nella realizzazione di un sistema con caratteristiche di distribuzione si pone il **problema di cosa distribuire.**
- Sistema distribuito sistemi per cui si verifica almeno una delle seguenti condizioni:
  - Le applicazioni, fra loro cooperanti, risiedono su più nodi elaborativi (elaborazione distribuita)
  - L'archivio informativo è distribuito su più nodi (base di dati distribuita)
- Ad esempio *il client* ed un *server* possono ripartirsi le funzionalità di elaborazione, oppure su una schiera di sistemi server possono essere ripartiti diversi sottoprogrammi che implementano diverse funzionalità

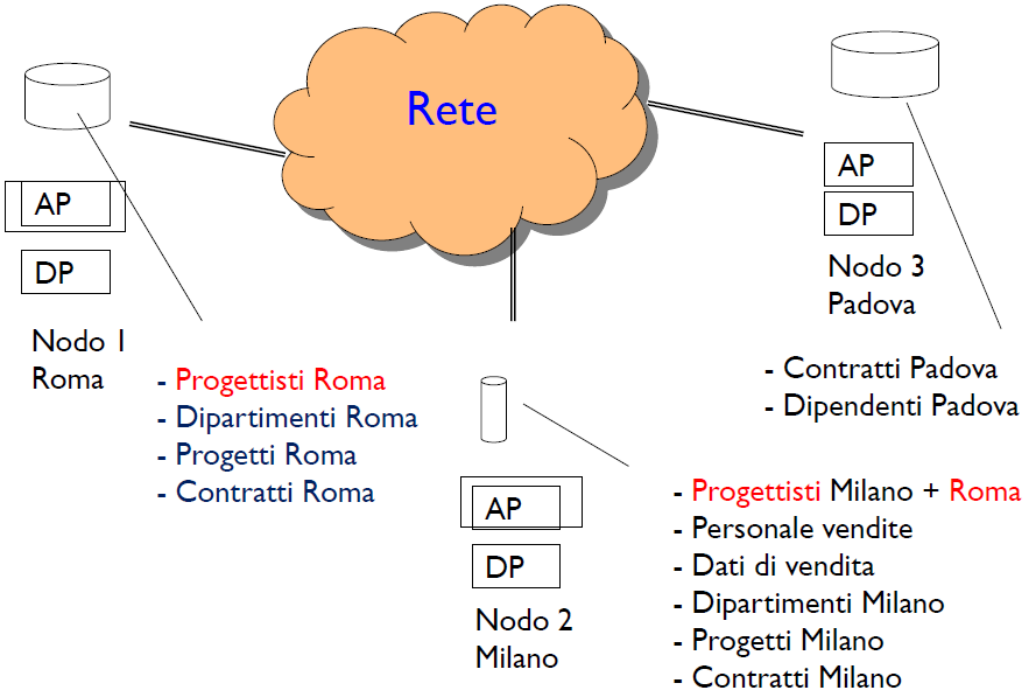


Quindi devo decidere cosa mettere da un lato e cosa dall'altro.

Per esempio pensiamo ad un'azienda che ha 3 sedi, con un'unica base di dati localizzata a Milano.

Come posso distribuirlo? (dato che ci possono essere problemi di rete). Con una basi dati in ciascuna località che contiene i dati relativi a quella località. Gli altri dati saranno disponibili tramite rete.

Il pericolo in questo esempio è la duplicazione dei dati, che in alcuni casi può avere senso ma va gestito.



Modelli architetturali per DDBMS

Un DBMS Distribuito Eterogeneo Autonomo è in generale una federazione di DBMS che collaborano nel fornire servizi di accesso ai dati con livelli di trasparenza definiti (la proprietà generale di nascondere le diversità tra le basi di dati nei nodi del sistema, in quanto a distribuzione, eterogeneità, autonomia).

L'autonomia è il grado di libertà di ciascun nodo. Nell'esempio precedente ci sarà un nodo amministratore che gestisce il sistema, quindi l'autonomia non è alta.

Tipi di autonomia:

- di progetto: ogni nodo adotta un proprio modello dei dati e sistema di gestione delle transazioni
- di condivisione: ogni nodo sceglie la porzione di dati che intende condividere con altri nodi
- di esecuzione: ogni nodo decide in che modo eseguire le transazioni che gli vengono sottoposte

Per la distribuzione si possono distinguere:

- DBMS Strettamente integrati (nessuna autonomia)
  - Dati *logicamente* centralizzati
  - Un unico data manager responsabile delle transazioni applicative
  - I data manager locali non operano in modo autonomo
- Semi-autonomi
  - Ogni data manager e' autonomo ma partecipa a transazioni globali
  - Una parte dei dati e' condivisa
  - Richiedono modifiche architetturali per poter fare parte della federazione
- Totalmente autonomi (o Peer to Peer)
  - Ogni DBMS lavora in completa autonomia ed e' inconsapevole dell'esistenza degli altri

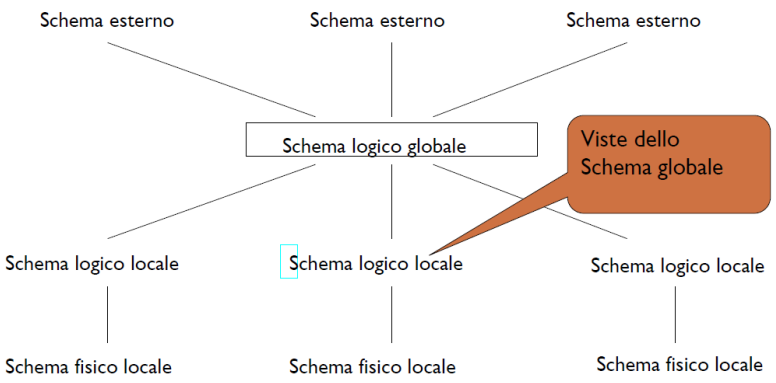
- Distribuzione client/server, in cui la gestione dei dati e' concentrata nei server, mentre i client forniscono l'ambiente applicativo e la presentazione.
- Distribuzione peer-to-peer, in cui non c'e' distinzione tra client e server, e tutti i nodi del sistema hanno identiche funzionalita' DBMS
- Nessuna distribuzione

Eterogeneità può riguardare:

- Modello dei dati
  - Es. Relazionale, XML, Object Oriented, NoSql
- Linguaggio di query
  - Es. Diversi dialetti SQL, Query by Example, Linguaggi di interrogazione OO o XML
- Gestione delle transazioni
  - Es. Diversi protocolli per il concurrency control e per il recovery
- Schema concettuale e schema logico
  - Es. Un concetto rappresentato in uno schema come attributo e in un altro come entita'

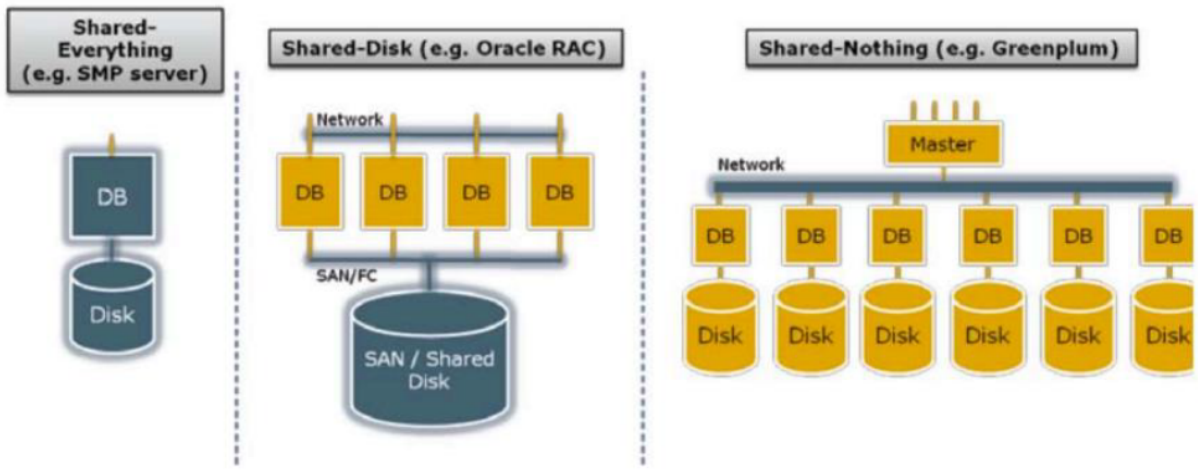
Progettazione di un DDBMS

Se prima avevo uno schema logico e uno fisico, ora ho tanti schemi fisici e logici (per ciascun nodo).



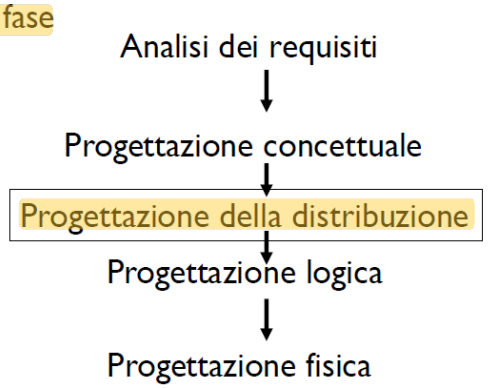
Lo schema logico locale sarà un sottoinsieme dello schema logico globale.

Architettura funzionale di un DDBMS:



Di solito viene usato l'ultimo, lo share nothing.

- La natura fortemente integrata dei DDBMS porta ad adottare per essi un approccio top-down alla progettazione, che rispetto alla progettazione di applicazioni DBMS, introduce una nuova fase



La **portabilità** è la capacità di eseguire le stesse applicazioni DB su ambienti runtime diversi. Questa cosa in realtà non succede, (per esempio per eseguire una query oracle su un database di tipo diverso) anche per obbligarti a rimanere con quel DBMS.

Interoperabilità: capacità di eseguire applicazioni che coinvolgono contemporaneamente sistemi diversi ed eterogenei.

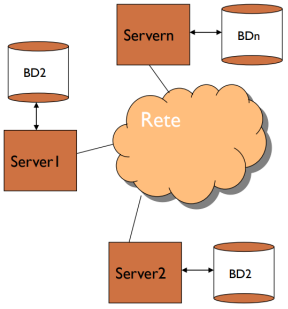
Vantaggi dei DDBMS:

- Località, i dati si trovano "vicino" alle applicazioni che li utilizzano più frequentemente ma sono globalmente raggiungibili.
- Modularità/flessibilità, le modifiche alle applicazioni e ai dati possono essere effettuate a basso costo. Distribuzione dei dati incrementale e progressiva: la configurazione si adatta alle esigenze delle applicazioni.
- Resistenza ai guasti: nonostante la presenza di elementi in rete aumenti la fragilità, la ridondanza aumenta la resistenza ai guasti.
- Prestazioni/Efficienza, distribuendo un DB su nodi diversi, ogni nodo gestisce un DB di dimensioni più ridotte:

+ Piu' semplice da gestire e ottimizzare nelle applicazioni locali
+ Ogni nodo puo' essere ottimizzato indipendentemente dagli altri
+ Carico totale (transazioni /sec) distribuito sui nodi
+ Parallelismo tra transazioni locali che fanno parte di una stessa transazione distribuita
- Necessita' di coordinamento tra i nodi
- Presenza di traffico di rete

Funzionalità specifiche dei DDBMS rispetto ai DBMS centralizzati

- Ogni server mantiene la capacità di gestire applicazioni in modo indipendente
- Le interazioni con altri server e applicazioni remote rappresentano un carico supplementare sul sistema

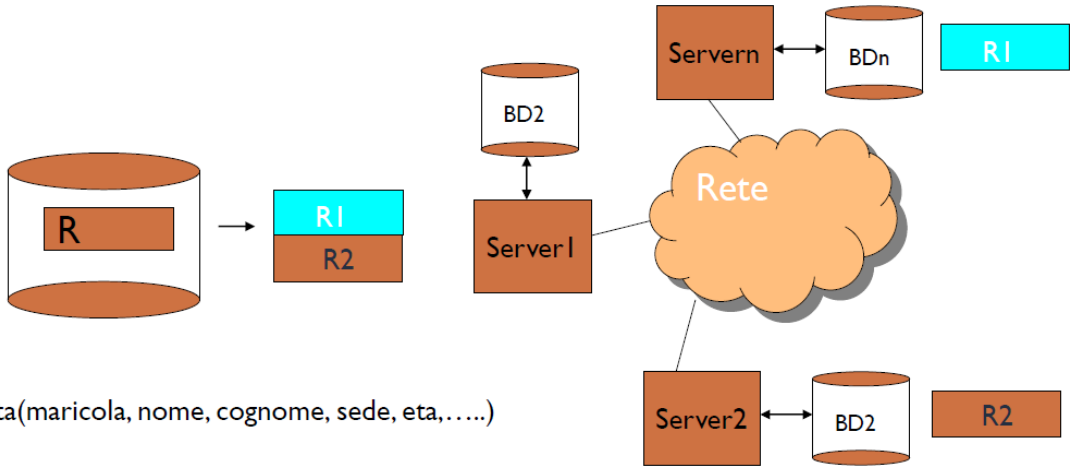


- Ottimizzazione: l'elemento critico è la rete
- Esigenza di distribuire i dati in modo che:
  - La maggior parte delle transazioni sia locale, o eviti trasmissione di dati tra nodi

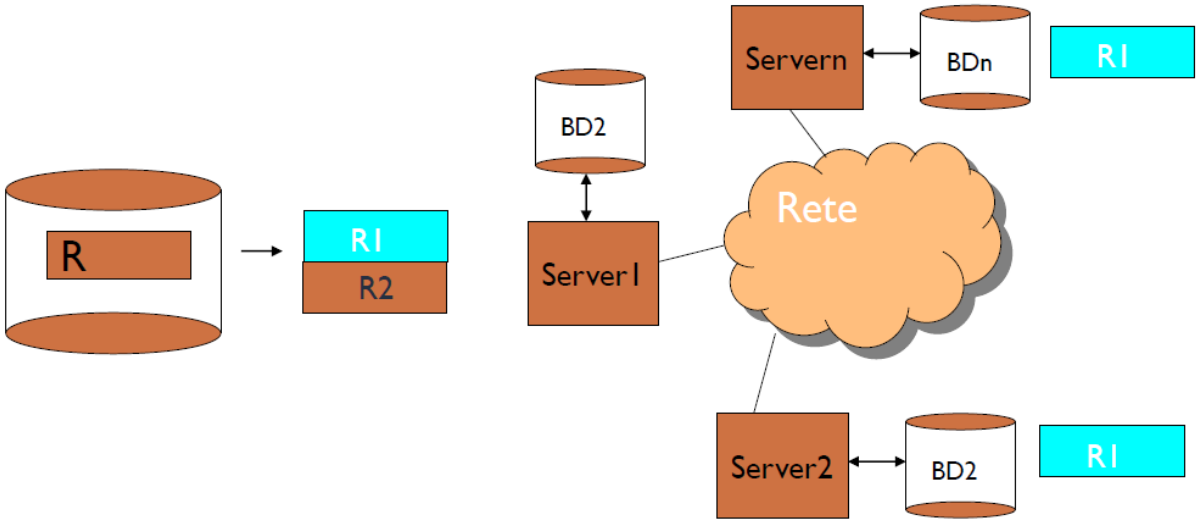
Caratteristiche strutturali dei DDBMS

Come distribuisco i dati?

Posso **frammentare** facendo una query per esempio che divide i dati tra le due sedi di un'azienda, in base proprio all'attributo sede. I due schemi R1 R2 sono uguali. (la frammentazione può essere orizzontale ma anche verticale, ovvero salvando attributi diversi)



Posso **replicare**, ovvero allocare stesse porzioni di database su nodi diversi



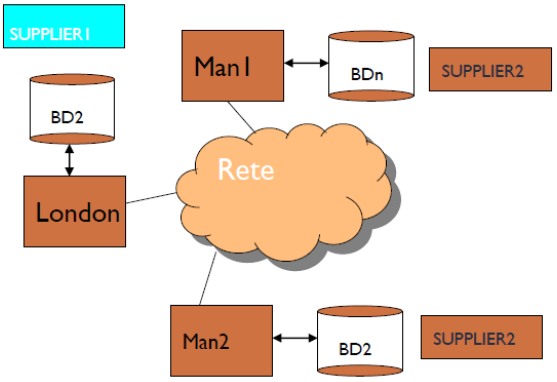
Quando faccio una query, il livello di trasparenza più alto è quello dove il DBMS unisce i pezzi distribuiti.
Quindi la **trasparenza** è la possibilità per l'applicazione di accedere ai dati senza sapere dove sono.

Ci sono 3 tipi di trasparenza:

- **Trasparenza di frammentazione** (è a carico del sistema la traduzione dalla query globale a quelle locali):
  - L'applicazione ignora del tutto l'esistenza di frammenti
  - E' lo scenario migliore dal punto di vista della programmazione applicativa
  - L'applicazione e' scritta in SQL standard
- **Trasparenza di allocazione**

- L'applicazione è consapevole dei frammenti, ma ne ignora l'allocazione sui nodi

```
procedure Query2(:snum,:name);
select Name into :name
  from SUPPLIER1
  where SNum = :snum;
if isEmpty(:name) then
  select Name into :name
    from SUPPLIER2
    where SNum = :snum;
end procedure;
```



- **Trasparenza di linguaggio**
  - L'applicazione deve specificare sia i frammenti che il loro nodo
  - E' il livello minimo di trasparenza tra i tre descritti
  - Un nodo può offrire interfacce che non sono standard SQL
  - Tuttavia, l'applicazione è scritta in SQL standard, a prescindere da eventuali altri linguaggi locali al nodo