Architetture delle applicazioni con DBMS

- Architettura classica: client-server
- DBMS è normalmente un server, le applicazioni sono dei client
  - Esempi: MySQL, PostgreSQL, DB2, Oracle, SQL Server
- Esistono DBMS anche sotto forma di librerie
  - SQLite (C),
  - Derby (Java)

Esempio DBMS come librerie: SQLite

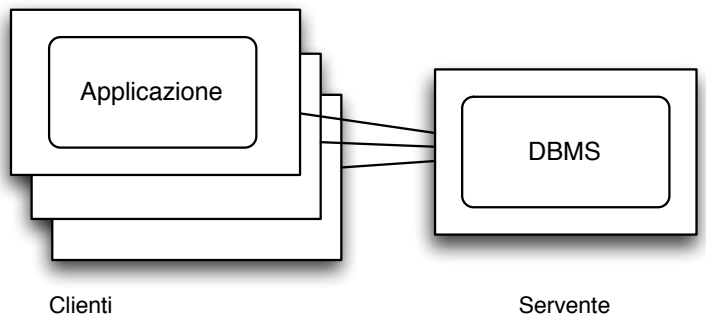
- “The Most Widely Deployed SQL Database” (~500 Mil. su ~100 Mil di altri DBMS)
- Firefox, Safari, iPhone, SymbianOS, MacOSX, ecc. ecc.
- Tutte le funzionalità di SQL (DDL e DML)
- Il database è un unico file
- Database in memoria
- Transazioni per affidabilità, accesso concorrente a livello di file/database

DBMS come server

- Processo in ascolto con qualche meccanismo di comunicazione fra processi, ad esempio su una porta, un socket, ecc.
- Ogni sistema ha un proprio protocollo di connessione, attraverso cui riceve comandi SQL e restituisce i risultati
- Le librerie di interfacce come JDBC e ODBC “mascherano” le differenze dei protocolli fra i vari sistemi.

Architetture client-server a due livelli

- Usata nelle situazioni più semplici



Architetture client-server a due livelli

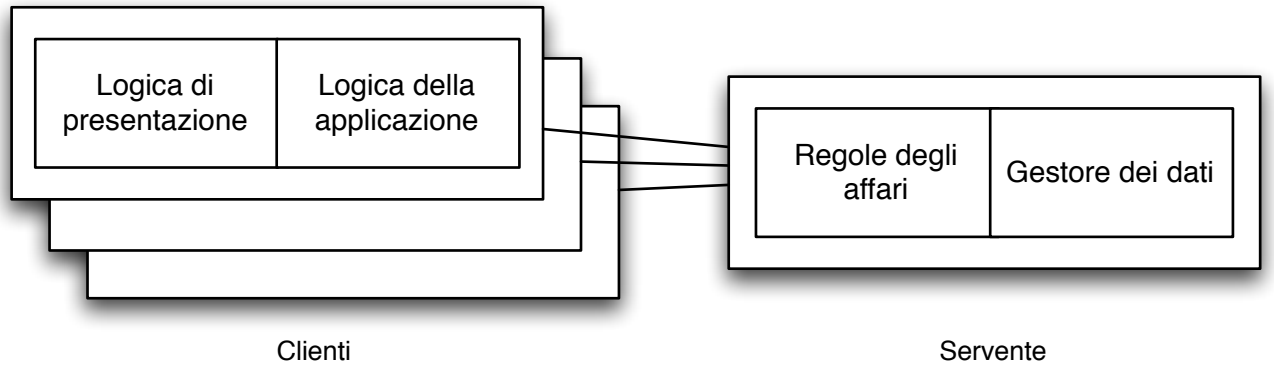
- Problemi di:
  - Scalabilità
    - Al crescere dei clienti aumenta il carico
    - Trasferimento di grandi quantità di dati
  - Sicurezza
    - Le credenziali di accesso sono contenute nell'applicazione
  - Gestione e manutenzione delle applicazioni
    - Ad ogni modifica dell'applicazione deve essere fatta manutenzione su tutte le macchine client
  - Distribuzione ottimale del codice
    - Il carico non è distribuito in maniera ottimale: i clienti come il servente possono essere sottoutilizzati o non in grado di eseguire l'intera applicazione

Componenti di un'applicazione

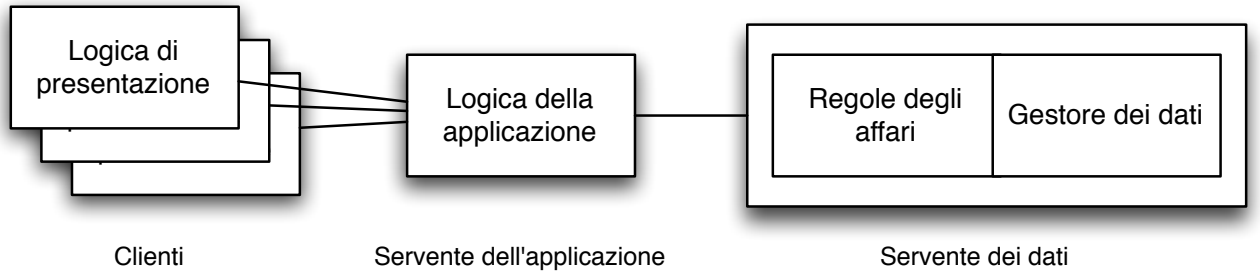
- Gestione dei dati (DBMS)
- Regole degli affari (Business rules)
  - Codice che regola il comportamento e i vincoli
  - Negli oggetti: metodi
  - Nei sistemi relazionali: procedure memorizzate e trigger
  - Usate negli Object-Relational Mapping
- Logica dell'applicazione
- Logica di presentazione

Architetture a più livelli

- Architettura client-server con suddivisione dei livelli

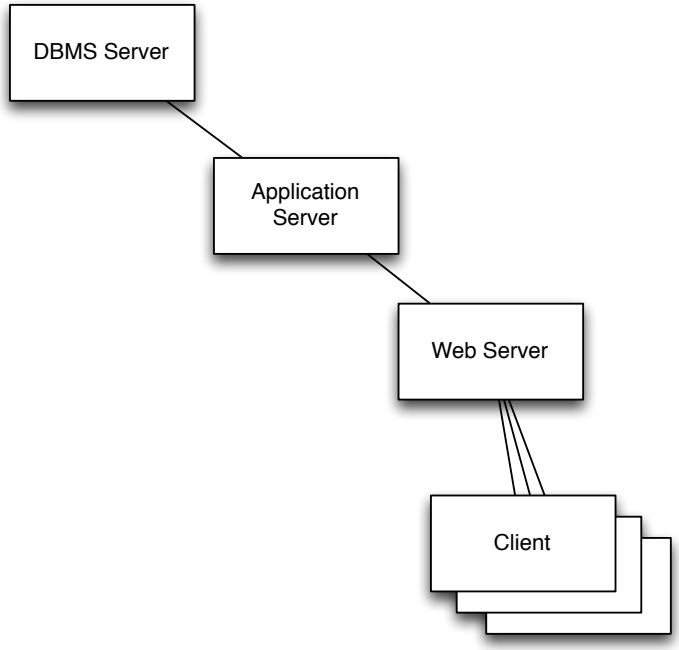


Architetture a tre livelli



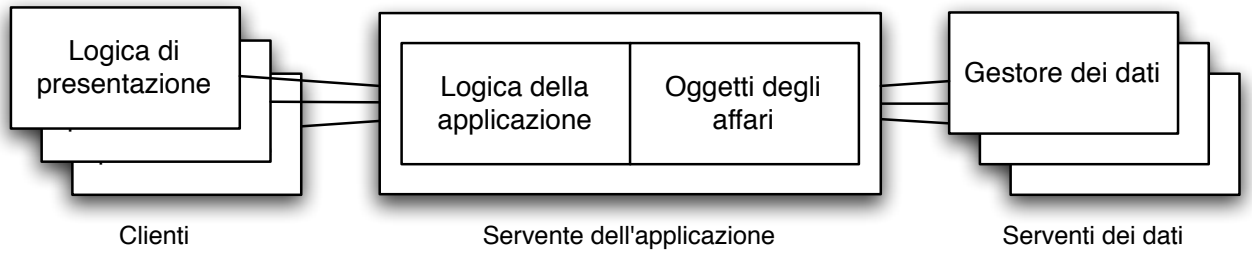
- Vantaggi:
  - Sicurezza
  - Gestione e manutenzione delle applicazioni
  - Suddivisione efficiente del carico
  - Indipendenza dalla piattaforma (dispositivi mobili)

Architetture a più livelli: il caso del web



- Anche l'application server può essere suddiviso per diverse funzionalità

Architetture a più livelli distribuite



- Il DBMS può avere:
  - distribuzione dei dati
  - replicazione dei dati
- Gestori del carico di lavoro (TPM: Transaction Processing Monitor)
- Gestori delle transazioni distribuite (protocollo standard XA)

Problemi delle architetture a più livelli

- Contrasto fra esigenze diverse:
  - Minimizzare il carico di rete
  - Distribuire il carico di elaborazione fra i diversi livelli e all'interno di un livello
  - Rendere il software adatto a dispositivi diversi (spostando parte della logica di presentazione verso l'applicazione)
- Realizzazione più complessa e costosa (spesso si usano strumenti eterogenei complessi)

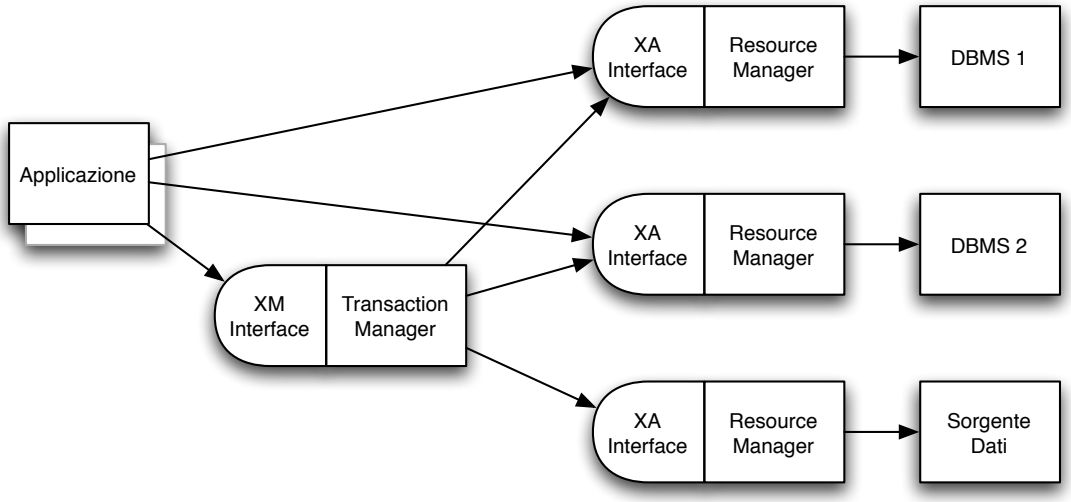
Strumenti di *middleware*

- Accesso a DBMS diversi
  - Metodo semplice: uso di JDBC con più driver e accessi a sistemi diversi
  - Metodo ancora più semplice: *Gateway* per basi di dati
  - Collegamenti ad un DBMS immersi in un altro DBMS:

```
CREATE DATABASE LINK databaseRemoto;
CONNECT TO CURRENT USER
USING 'Nome database esterno';
CREATE SYNONYM Impiegati FOR RelImpiegati@databaseRemoto;

UPDATE Impiegati
SET Stipendio = Stipendio * 1.1
WHERE CodDipartimento IN
        (SELECT CodiceFROM Dipar timenti WHERE Citta = "Roma");
```

Integrazione con TP Monitor



- Esistono standard per il middleware di gestione di sistemi distribuiti
- Servizi di comunicazione, transazioni distribuite, gestione del carico
- XA-*interface*: gestione delle risorse
- XM-*interface*: gestione delle transazioni

Transazione distribuita

- Un'unica transazione che accede a dati gestiti da DBMS diversi
- Deve essere atomica: se c'è un malfunzionamento è necessario disfare le modifiche fatte su tutti i DBMS
- Ci possono essere fallimenti nei singoli nodi della transazione
- Ci può essere anche un fallimento dovuto a problemi di comunicazione (si interrompe l'accesso ad un nodo)
- Il commit avviene in due fasi:
  - Tutte le "sottotransazioni" si dichiarano disposte a fare commit
  - Tutte sono a conoscenza di questa disponibilità: inizia la fase di commit
  - Alla fine si verifica che tutte abbiano effettivamente fatto commit

TP Monitor (2)

- XA-interface: Interazione con un *singolo* gestore dei dati
  - xa-open e xa-close, per iniziare e terminare le sessioni di lavoro
  - xa-start e xa-end, per iniziare e terminare transazioni
  - xa-precomm, richiesta di preparazione per terminazione
  - xa-commit e xa-abort, comunicano la decisione globale
  - xa-recover e xa-forget, ripresa comunicazione e ripristino stato
- XM-interface: Coordinamento della transazione distribuita globale
  - tm-init e tm-exit, sessione di lavoro con il TP
  - tm-open e tm-close, sessione di lavoro con più Resource Manager
  - tm-begin, inizio transazione globale
  - tm-commit e tm-abort, termine transazione globale