DBMS ed Applicazioni

Motivazioni

- Sin'ora abbiamo visto SQL come linguaggio per interrogare DBMS da interfaccia interattiva
- Nella pratica, un efficace sfruttamento delle potenzialità dei DBMS deriva dalla loro integrazione con applicazioni scritte in generici linguaggi di programmazione
- L'integrazione tra DBMS ed applicazioni può avvenire secondo diversi schemi:
  - SQL incapsulato
  - interfacce ODBC/JDBC
  - Accesso da Internet

SQL incapsulato

- Si parla di SQL incapsulato (embedded) quando i comandi SQL vengono usati all'interno di un linguaggio ospite (JAVA, c, C++, …)
- I comandi SQL vengono segnalati con opportune direttive in modo che il processore possa trasformarli in funzioni del linguaggio ospite prima della compilazione del codice

```
EXEC SQL BEGIN DECLARE SECTION
…
EXEC SQL END DECLARE SECTION
```


SQL incapsulato

- Nell'applicazione, il risultato di una select SQL viene rappresentato attraverso i cursori
- Questi sono necessari per la mancanza—in molti linguaggi di alto livello—di un tipo di dati che rappresenti gli insiemi di valori
- Un cursore consente di leggere una per volta le righe di una tabella

SQL incapsulato

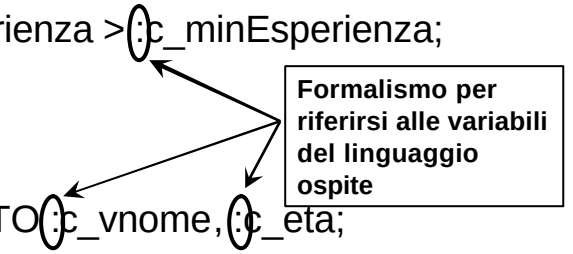
- Una volta dichiarato un cursore lo si può:
  - Aprire (posizionamento sulla prima riga)
  - Leggere (il contenuto della riga corrente)
  - Muovere (all'n-ma riga)
  - Chiudere

SQL incapsulato

```
DECLARE vinfo CURSOR FOR
SELECT V.vnome, V.età
FROM Velisti V
WHERE V.esperienza > :c_minEsperienza;

…

OPEN vinfo;
FETCH vinfo INTO :c_vnome, :c_eta;
CLOSE vinfo;
```



ODBC e JDBC

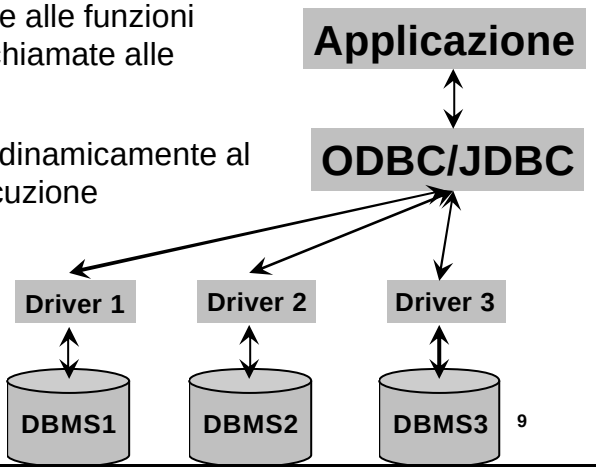
- La compilazione richiede l'uso di particolari librerie che sono specifiche per ciascun tipo di DBMS:
  - Una volta compilato, il codice funziona solo con un particolare DBMS
- Una evoluzione rispetto a questa soluzione è quella di fare riferimento a interfacce standard:
  - ODBC, JDBC

ODBC e JDBC

- ODBC e JDBC (Open/Java DataBase Connectivity) presentano al programmatore una interfaccia standardizzata per accedere al DBMS
- La specifica di una operazione da fare sulla base dati avviene attraverso l'uso delle funzioni supportate da ODBC (JDBC)
- L'eseguibile che viene generato può accedere a diversi DBMS, senza ricompilazione

ODBC e JDBC

- Questo è possibile grazie ad un driver (specifico per ciascun DBMS) che traduce le chiamate alle funzioni ODBC (JDBC) in chiamate alle funzioni del DBMS
- Il driver è caricato dinamicamente al momento dell'esecuzione dell'applicazione



ODBC e JDBC

- In particolare, l'interazione attraverso ODBC/JDBC prevede le seguenti fasi:
  - Selezione della sorgente dati (base dati)
  - Caricamento dinamico del driver opportuno per la sorgente dati
  - Connessione con la sorgente dati
  - Sessione di interazione attraverso comandi SQL
  - Chiusura della connessione

JDBC esempio

```
import java.sql.*;
public class JDBCSample
{
  public static void main(java.lang.String[] args)
  {
    try
    {  Class.forName(“com.mysql.jdbc.Driver”); }
     catch (ClassNotFoundException e)
     {  System.out.println(“Unable to load Driver Class”);
        return; }
     try
    {  Connection con =
   DriverManager.getConnection("jdbc:mysql://localhost/bank",“user",“passw");
    …
```

Selezione sorgente e carica driver

Stabilisce connessione

JDBC esempio

- Con JDBC l’esecuzione dei comandi SQL può avvenire attraverso tre classi:
  - Statement: classe per generici comandi SQL
  - PreparedStatement: genera comandi SQL con struttura prefissata (quando l’oggetto è creato). Eventuali parametri del comando sono indicati con il carattere “?”
  - CallableStatement: necessaria per accedere alle *stored procedures*

JDBC esempio

```
String sql="INSERT INTO Libri VALUES(?,?,?)";
PreparedStatement ps=con.prepareStatement(sql);
ps.clearParameters();
ps.setString(1, isbn);
ps.setString(2, titolo);
ps.setString(3, autore);
...
int numRows = ps.executeUpdate();
```


JDBC esempio

- In Java/JDBC il ruolo dei cursori è svolto dagli oggetti di tipo ResultSet
- Questi consentono sia la scansione del risultato di una query che operazioni di cancellazione ed inserimento tuple nel risultato

JDBC esempio

```
String sqlQuery;
Statement st;
ResultSet rs = st.executeQuery(sqlQuery);
while(rs.next()){
    isbn=rs.getString(1);
    title=rs.getString("Title");
    ...
}
```


Stored procedures

- Spesso è vantaggioso poter eseguire blocchi di comandi direttamente nello spazio del processo del DBMS
  - Utilizzo della potenza del server
  - Riduzione del trasferimento dati client/server (non si spediscono i risultati delle elaborazioni parziali, solo il risultato finale)
- Le stored procedures sono programmi memorizzati nel DBMS la cui esecuzione può essere richiesta con un singolo comando SQL

Stored procedures

- Ulteriori vantaggi:
  - Possibilità di rendere condivisibile un programma a più utenti
  - Se tutti gli accessi alla base dati avvengono attraverso SP, gli utenti non devono conoscere lo schema della base dati (maggior sicurezza, maggior semplicità d'uso)

Stored procedures

- Una stored procedure si caratterizza per un nome:

```
CREATE PROCEDURE MostraNumeroOrdini
SELECT C.cid, C.nome, count(distinct *)
    FROM Clienti C, Ordini O
    WHERE C.cid=O.cid
    GROUP BY C.cid, C.cnome
```

Stored procedures

- Una stored procedure può anche avere dei parametri. Questi possono essere di tre tipi:
  - IN: argomenti per la procedura
  - OUT: valori di uscita
  - INOUT: parametri passati alla procedura e da essa modificabili

Stored procedures

```
CREATE PROCEDURE AggiungiInventario(
  IN libro_isbn CHAR(10),
  IN aggiungiQuota INTEGER)
  UPDATE Libri
  SET quotaStock = quotaStock+aggiungiQuota
  WHERE libro_isbn=isbn
```

Stored procedures

- Le stored procedures non devono necessariamente essere scritte in SQL:
  - Possono essere scritte usando un linguaggio apposito del DBMS (SQL/PSM, PL/SQL)
  - Oppure in qualsiasi linguaggio ospite:

```
CREATE PROCEDURE Posizione(IN numero INTEGER)
  LANGUAGE Java
  EXTERNAL NAME 'file:///c:/procedureInterne/posizione.jar'
```


Stored procedures

- Chiamate alle stored procedures:
  - Con SQL interattivo: comando CALL:
    CALL storedProc(arg1, arg2, …, argn)
  - Con JDBC: classe CallableStatement

```
CallableStatement cs;
cs=con.prepareCall("{call MostraNumOrdini}");
ResultSet rs = cs.executeQuery();
while(rs.next){
…}
```

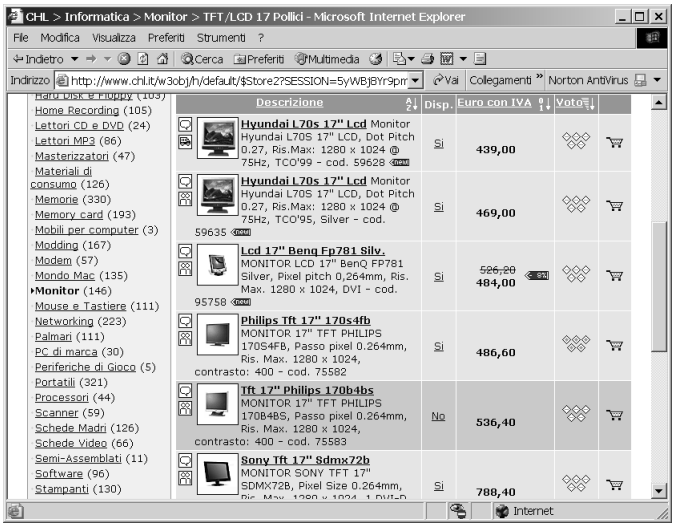
Applicazioni Internet

Applicazioni Internet

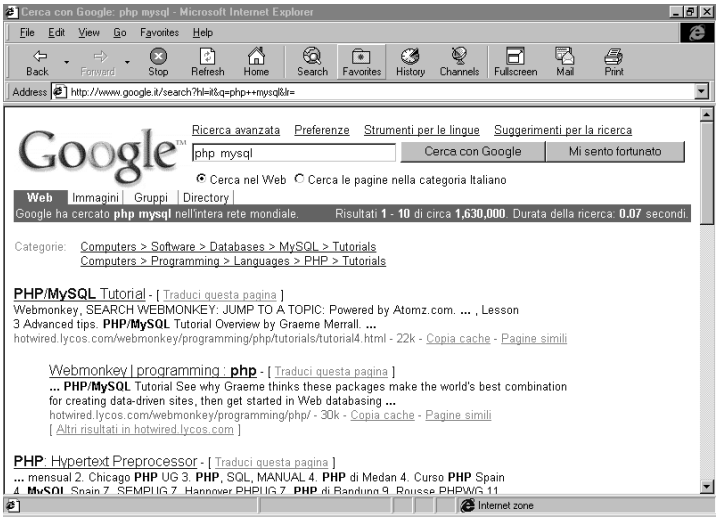
- A differenza dei siti Internet di prima generazione, molti dei siti oggi sfruttano pagine Web generate dinamicamente (non statiche)
- Di solito, tali pagine sono generate automaticamente attraverso l'estrazione di informazioni da basi di dati

Pagine Web dinamiche

Pagine Web dinamiche

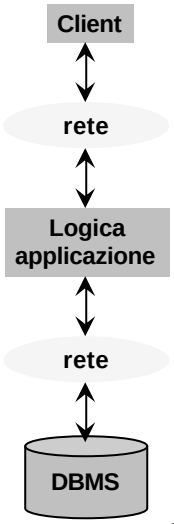


Pagine Web dinamiche



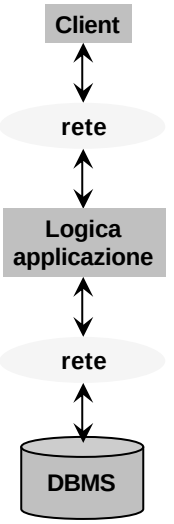
Architettura a tre livelli

- La tipologia di architettura più diffusa per le applicazioni che accedono a basi dati da Internet è quella a tre livelli:
  - Livello presentazione
  - Livello intermedio
  - Livello gestione dati
- Soluzioni alternative (client side): applet ed applicazioni JAVA

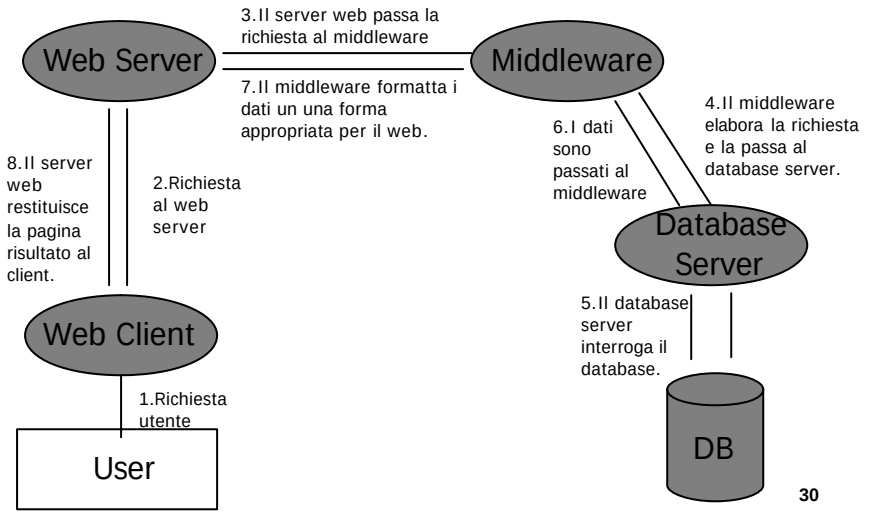


Architettura a tre livelli

- Presentazione: client con interfaccia utente, tipicamente browser Web (javascript, VBscript)
- Middleware: logica dell'applicazione vera e propria (application servers, servlet, JSP, PHP, ASP, XSLT)
- Gestione dati: il DBMS

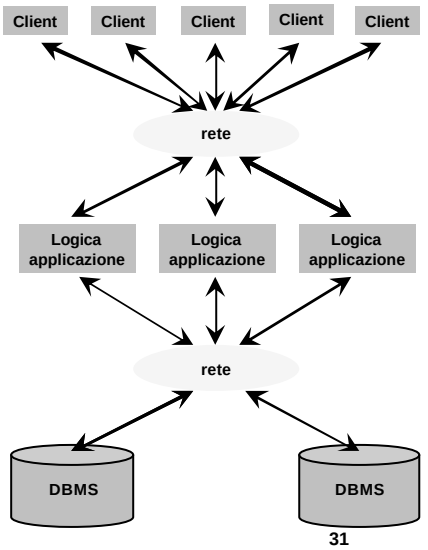


Schema di funzionamento



Architettura a tre livelli

- Vantaggi:
  - E' possibile integrare sistemi basati su piattaforme diverse
  - Non sono richieste eccessive risorse di calcolo da parte del client
  - E' possibile accedere ad informazioni da sorgenti dati diverse, e questo in maniera trasparente al client
  - Il livello middleware può essere distribuito su server diversi in modo da avere un sistema scalabile in base al numero di client connessi



Estensioni del Web server

- Applicazioni Web su larga scala, progettate per supportare e-commerce devono assicurare:
  - Affidabilità
  - Velocità
  - Scalabilità
- Per assicurare questi requisiti le applicazioni devono avere un'architettura modulare in cui ogni componente possa essere:
  - Replicato, per aumentare le prestazioni
  - Sostituito, per risolvere guasti

Estensioni del Web server

- In questo contesto si collocano gli Application Server
- Sono delle piattaforme software separate dal Web server impiegate per garantire **velocità**, **affidabilità** e **scalabilità**
- Costituiscono un ambiente di esecuzione in grado di facilitare la **costruzione di applicazioni scalabili ed affidabili** attraverso:
  - Distribuzione di componenti e bilanciamento di carico
  - Ripristino guasti
  - Condivisione risorse
  - Interoperabilità con applicazioni pre-esistenti

Application server

- Distribuzione di componenti e bilanciamento di carico:
  - Le applicazioni programmate dagli utenti vengono installate sull'application server che può essere distribuito su diverse macchine
  - L'application server gestisce automaticamente la creazione dei processi e la loro automatica replica, adattando dinamicamente il numero di processi sulle diverse macchine in modo da ottimizzare il bilanciamento del carico

Application server

- Ripristino guasti:
  - L'application server può monitorare gli utenti attivi ed i processi
  - E' in grado di rilevare il guasto di alcuni componenti software e ridirigere le richieste verso gli oggetti applicativi replicati non guasti

Application server

- Condivisione risorse:
  - L'application server può gestire la condivisione di risorse dispendiose tra diversi oggetti applicativi: per esempio può gestire le connessioni con DBMS

Application server

- Interoperabilità con applicazioni pre-esistenti:
  - L'application server può essere dotato di strumenti per gestire l'interfacciamento con applicazioni sviluppate su piattaforme obsolete o con tecnologie superate