Progettazione di siti web centrati sui dati (Data-Intesive Web Applications) Dispensa del corso di Basi di dati e Multimedia 2003-2004

Alberto Belussi
Università degli Studi di Verona
Dipartimento di Informatica

Aprile 2004

Indice

1 Introduzione **2**

2 Le tecnologie per la realizzazione di un sito web centrato sui dati **4**

2.1 Interazione tra browser e web server 4

2.2 Le tecnologie per la realizzazione di pagine dinamiche 7

3 La realizzazione di una pagina web dinamica usando la tecnologia JavaServlet di Sun **10**

3.1 Gli oggetti delle classi `HttpServletRequest` e `HttpServletResponse` 11

3.2 JDBC: una libreria di classi Java per l'interazione con un database server 12

3.3 Esempio di servlet che realizza una pagina dinamica 16

1 Introduzione

L'argomento trattato in questa dispensa riguarda la realizzazione di un *sito web centrato sui dati*. In particolare l'obiettivo è quello di presentare allo studente le soluzioni tecnologiche per la realizzazione di un sito di questa categoria.

Con il termine sito web centrato sui dati (o data-intesive web site) si intende indicare un sito web nelle cui pagine si presenta un insieme di *dati strutturati* che descrivono le informazioni di interesse per un certo ambito applicativo. Un insieme di dati strutturati è caratterizzato dall'avere una struttura ben definita e stabile nel tempo; in altre parole, esso si presenta come un insieme di elementi di informazione tutti con le medesime caratteristiche. Ad esempio, l'elenco telefonico è un insieme di dati strutturati visto che è costituito da una lista di elementi tutti con la stessa struttura: nome, cognome, indirizzo e numero di telefono. Ogni insieme di dati strutturati ben si presta alla rappresentazione in una base di dati (in accordo con quanto lo studente ha già visto nella prima parte del corso di Basi di dati e Multimedia).

Nelle situazioni più frequenti la progettazione di un sito web centrato sui dati parte da uno stato iniziale in cui è già presente una base di dati, che gestisce le informazioni di interesse dell'organizzazione per la quale si vuole allestire un sito web. In particolare, spesso accade che l'insieme di dati strutturati da pubblicare coincida in buona parte con quanto già memorizzato e gestito nella base di dati stessa.

Si noti che il sistema che gestisce la base di dati è diverso da quello che governa il sito web: vale a dire, si tratta di due sistemi software distinti. Quindi la realizzazione di un sito web centrato sui dati si occuperà anche di far comunicare questi due sistemi.

In generale, quando i dati da gestire presentano una struttura ben definita e stabile nel tempo, è conveniente adottare per la loro gestione un sistema dedicato alla rappresentazione e interrogazione di basi di dati. Tale sistema, detto anche Data Base Management Systems (DBMS), consente infatti di gestire grandi quantità di dati persistenti garantendo affidabilità, privatezza e accesso efficiente. Quindi, anche quando nello stato iniziale del progetto non è presente un sistema per la gestione di basi di dati, supporremo che, per la realizzazione di un sito web centrato sui dati, tale sistema venga allestito e i dati vengano rappresentati in una base di dati. Vale a dire, l'ipotesi che vogliamo fissare è che una base di dati sia sempre presente.

In sintesi, nella architettura di un sito web centrato sui dati abbiamo due *attori*: il sistema che gestisce le richieste di pagine web che provengono dai browser (server web o http server[1]) e il sistema che gestisce la base di dati contenente i dati da pubblicare (server dati o database server). Si veda lo schema mostrato in Figura 1.

Nella figura si può notare che il server web (http server) contiene uno specifico modulo, detto *Engine for dynamic web pages*, che gestisce la generazione di pagine dinamiche e interagisce con il database server. Lo strumento che verrà utilizzato nelle esercitazioni svolte nel corso di Laboratorio di Basi di dati e Multimedia contiene esattamente un modulo di questo tipo.

Nella prima parte del corso di Basi di dati e Multimedia sono stati introdotti i concetti fondamentali relativi alla progettazione di una base di dati. Nelle prime lezioni del corso di Laboratorio si illustreranno le nozioni fondamentali per la corretta stesura di un documento HTML (*Hyper Text Markup Language* `http://www.w3c.org`) per la generazione di una pagina web statica. In questa dispensa vedremo come integrare queste conoscenze e, in particolare, come far sì che i dati estratti da una base di dati di un server dati possano essere automaticamente trasferiti in un file HTML gestito da un server web. Infatti, la caratteristica fondamentale dei siti centrati sui dati è la dinamicità del contenuto delle pagine che li

[1] dal nome del protocollo che sta alla base del suo funzionamento: Hyper Text Transfer Protocol - HTTP `http://www.w3c.org/Protocols`

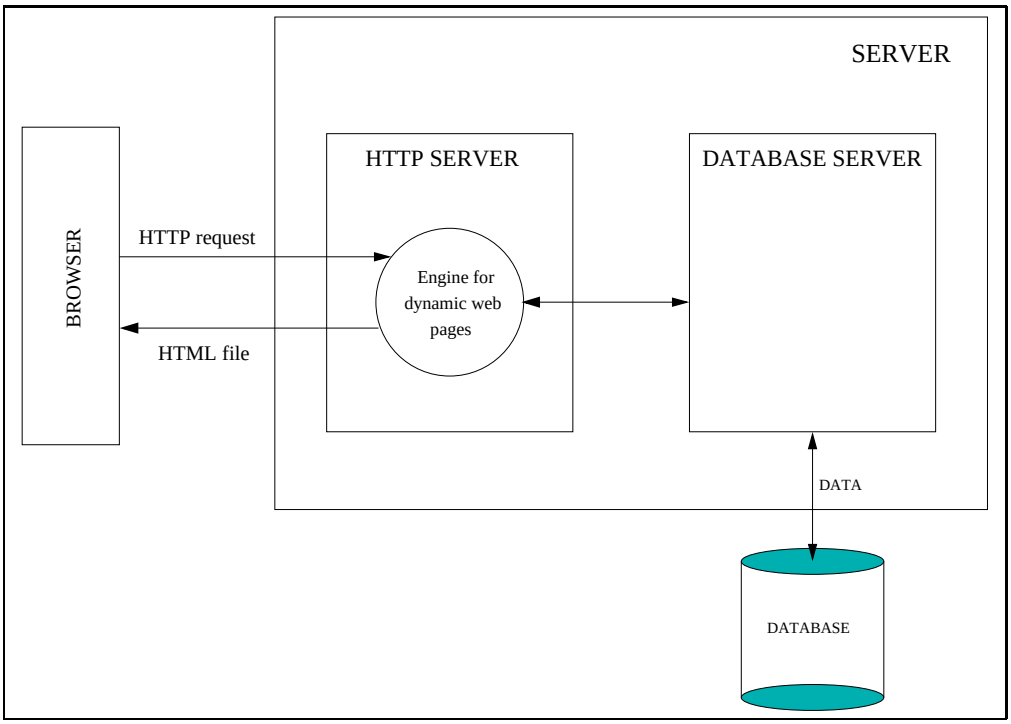

Figura 1: Architettura di un sistema che gestisce un sito web centrato sui dati.

compongono. Mentre in un sito tradizionale le pagine web che lo costituiscono sono file statici scritti nel linguaggio HTML e il server web, alla richiesta di una certa pagina da parte di un browser, non fa altro che spedire il file così come è memorizzato nel server web stesso, nei siti centrati sui dati la pagina web viene generata al momento della richiesta combinando lo schema base della pagina (parte statica) con i dati estratti dalla base di dati (parte dinamica) e producendo, solo al momento della richiesta, il file HTML che verrà inviato al browser.

In questa dispensa si intendono analizzare le tecnologie che consente di realizzare pagine web dinamiche, presentando in particolare con maggiore dettaglio la tecnologia JavaServlet di Sun.

Per affrontare lo studio di questa dispensa sono necessari i seguenti prerequisiti:

- una sufficiente conoscenza dei principali tag del linguaggio HTML (*Hyper Text Markup Language*);
- una buona conoscenza delle basi di dati e in particolare si richiede la capacità di: progettare lo schema concettuale di una base di dati nel modello Entità-Relazione (ER), tradurre lo schema ER in uno schema relazionale per la rappresentazione dei dati a livello logico e creare/interrogare una base di dati relazionale usando il linguaggio SQL.

La dispensa è strutturata come segue: nella sezione successiva si trattano le tecnologie per la realizzazione di un sito web centrato sui dati, mentre nell'ultima sezione si presentano i dettagli per la generazione di pagine web dinamiche con la tecnologia JavaServlet di Sun.

2 Le tecnologie per la realizzazione di un sito web centrato sui dati

In questo capitolo si presenteranno brevemente le caratteristiche delle principali tecnologie oggi disponibili per realizzare un sito web centrato sui dati. In particolare, l'obiettivo è quello di presentare un insieme di concetti comuni a tutte le tecnologie per la generazione di pagine web dinamiche e di focalizzare poi sulla tecnologia *JavaServlet* di Sun. Infatti, come descritto nell'introduzione, una sito web centrato sui dati è costituito da un insieme di *pagine web a contenuto dinamico* (presentati dati estratti da una base di dati), dette più semplicemente *pagine web dinamiche*

Poiché non si ha la pretesa di coprire interamente l'argomento relativo alla generazione di pagine web dinamiche, che comprende necessariamente anche un corso di programmazione con linguaggi imperativi, in questo capitolo si presenteranno in modo informale e discorsivo molti concetti che in una trattazione più approfondita avrebbero richiesto molto più spazio e rigore.

Il capitolo è organizzato come segue: nella prima sezione si descrive brevemente il protocollo HTTP, che regola l'interazione tra browser e web server presentando, in particolare, come si realizza tale interazione nel caso in cui il sito sia costituito da pagine web dinamiche. Nella sezione successiva si presentano le principali differenze tra le tecnologie più diffuse per la realizzazione di pagine web dinamiche.

2.1 Interazione tra browser e web server

La navigazione sulla rete Internet si basa sull'interazione tra l'applicazione di visualizzazione di ipertesti (detta comunemente *browser*) e i web server connessi alla rete Internet. Ogni volta che sul nostro browser digitiamo un indirizzo di una pagina web (ad esempio, `http://www.scienze.univr.it`) stiamo instaurando con uno specifico web server una interazione che prevede lo scambio di dati sulla rete.

Il protocollo che regola tale interazione si chiama Hyper Text Transfer Protocol (HTTP). Tale protocollo, come indica il suo nome, regola lo scambio di ipertesti tra browser e web server. Non si vuole qui analizzare nel dettaglio tale protocollo, ma soltanto chiarire le caratteristiche fondamentali del suo funzionamento e, in particolare, descrivere cosa accade quanto si è in presenza di pagine web dinamiche.

Va innanzitutto precisato che l'interazione tra browser e web server si ha, non solo quando si digita esplicitamente l'indirizzo di una pagina web (tale indirizzo viene anche detto Universal Resource Locator - URL), ma anche quando si clicca su un link presente nella pagina visualizzata correntemente dal browser. Infatti, se ricordiamo la sintassi per la specifica di un link nel linguaggio HTML, essa contiene sempre la specifica di un URL. Si noti che un URL può rappresentare in generale richieste basate su protocolli diversi da HTTP (ad esempio basate su protocollo FTP - File Transfer Protocol). L'indicazione del protocollo occupa infatti la prima parte dell'URL la quale, per le pagine web, inizia sempre con la stringa http://, visto che il protocollo usato in questo caso è esattamente l'HTTP.

Quando il sito è costituito da pagine statiche, l'interazione browser/web server risulta molto semplice. I passi in cui si articola tale interazione sono i seguenti:

- Il browser invia la richiesta di una pagina web al server specificando, oltre all'URL, una serie di altre informazioni tra cui il tipo di richiesta (fra i tipi possibili segnaliamo il tipo di richiesta GET e il tipo di richiesta POST).
- Il web server riceve la richiesta del browser e individua il file HTML che corrisponde alla richiesta.

- Il web server invia il file HTML individuato al browser (in molti casi la presenza nella pagina web di immagini o di una struttura a frame richiede in realtaù più di un'interazione di questo tipo per la visualizzazione dell'intera pagina sul browser).

Per la gestione di pagine web statiche il server non necessita di particolari strumenti, ma è sufficiente una struttura di cartelle (dette anche folder o directory), dove memorizzare tutti i file HTML che costituiscono le pagine web del sito (vedi Figura 2).

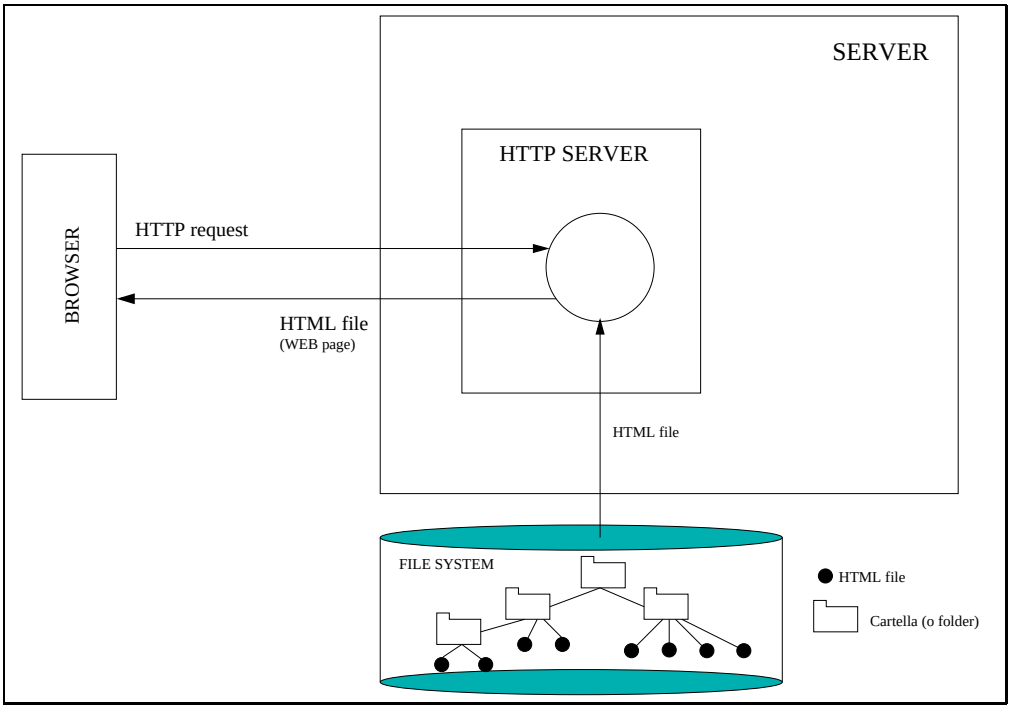

Figura 2: Interazione browser/web server per lo scambio di una pagina web statica.

Cosa accade invece quando il sito è costituito da pagine web dinamiche?

L'interazione in questo caso è fondamentalmente simile a quella che si ha per le pagine statiche con una importante differenza: oltre a comunicare l'URL dello schema di pagina richiesto, il browser invia al web server anche alcuni parametri per individuare i dati da inserire nello schema di pagina e produrre la pagina web finale. Con *schema di pagina* si intende qui indicare la parte statica della pagina web da generare. Infatti, la pagina web dinamica viene costruita al momento della richiesta (on the fly) combinando uno schema statico HTML con il risultato di un'estrazione di informazioni da una base di dati gestita da un database server.

In questo caso i passi in cui si articola l'interazione browser/web server e l'evasione della richiesta da parte del server sono i seguenti:

- Il browser invia la richiesta di una pagina web al server specificando un URL contenente uno o più parametri. L'invio dei parametri può avvenire in due modi: a) in una richiesta di tipo GET i parametri sono accodati all'URL secondo la sintassi seguente:

```
http://www. ..... /<nome_schema_pagina>?
```

```
<nome_parametro_1>=<valore_parametro_1>&....
&<nome_parametro_n>=<valore_parametro_n>
```

ad esempio:

```
http://www.sitoaziendale.com/Dipendente?cod_dip=3
```

b) in una richiesta di tipo POST i parametri sono inviati al web server in modo indipendente dall'URL e non sono quindi visibili nella stringa che rappresenta l'URL stessa.

- Il web server riceve la richiesta del browser, individua lo schema di pagina richiesto e costruisce la/le interrogazione/i da inviare al database server in base ai valori dei parametri ricevuti.
- Il database server interagisce con la base di dati per rispondere alle interrogazioni ricevute dal web server e invia a quest'ultimo il risultato.
- Il web server riceve il risultato delle interrogazioni dal database server e elabora tale risultato per inserirlo nello schema di pagina in modo da produrre la pagina web finale.
- Il web server invia al browser il file HTML che rappresenta la pagina web prodotta al passo precedente. Si noti che, il browser non è in grado di riconoscere, dal codice HTML ricevuto, se la pagina web è statica o dinamica.

In Figura 3 si mostra una esempio di interazione browser/web server in presenza di pagine web dinamiche.

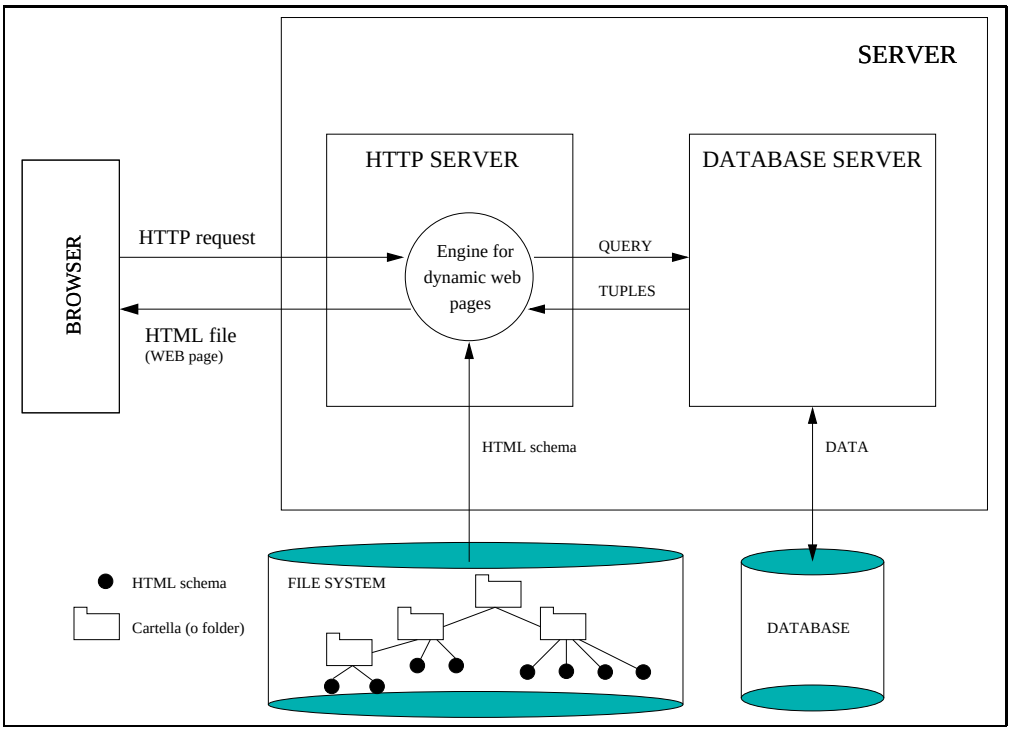

Figura 3: Interazione browser/web server per lo scambio di una pagina web dinamica.

Nella prossima sezione vediamo quali sono le tecnologie più diffuse che vengono utilizzate per la realizzazione pagine dinamiche.

2.2 Le tecnologie per la realizzazione di pagine dinamiche

In questa sezione si presentano molto brevemente le tecnologie attualmente disponibili per generare pagine web dinamiche.

Le soluzioni disponibili si suddividono storicamente in due categorie:

- **Common Gateway Interface (CGI)**: questa soluzione è stata la prima ad essere adottata sui web server per la generazione di pagine web dinamiche. Essa consente al web server di attivare altri programmi sul server e di comunicare con tali programmi inviando una richiesta di elaborazione e ottenendo il risultato della stessa elaborazione dal programma. Per ogni richiesta proveniente dal browser nell'approccio CGI viene attivato un nuovo programma (processo) che si occupa di ricevere l'interrogazione, collegarsi al database server, eseguire l'interrogazione e restituire il risultato al server web. L'attivazione di un nuovo processo a fronte di ogni nuova richiesta è il principale svantaggio di questa soluzione; infatti, tale attivazione richiede tempo e risorse di memoria e, su un web server con traffico elevato, ciò potrebbe portare velocemente al blocco del sistema. La soluzione CGI, inoltre, presenta alcuni problemi di sicurezza, in quanto risulta maggiormente vulnerabile esattamente nella fase di attivazione di processi sul sistema operativo. Infine, la stesura di pagine web dinamiche richiede necessariamente in questo approccio la conoscenza approfonditia di un linguaggio di programmazione imperativo (ad esempio, C o C++ o Java o altro).

- Approcci basati sull'estensione del web server: queste soluzioni prevedono di estendere il web server con moduli software aggiuntivi che siano in grado di gestire l'esecuzione di codice senza attivare nuovi processi e senza interagire direttamente con il sistema operativo del server. Descriviamo brevemente alcuni di questi approcci.

  - **JavaServlet** di Sun: questa soluzione prevede di estendere il web server con un interprete Java (Java Virtual Machine - JVM) per l'esecuzione di programmi Java sul server, di qui il nome *Servlet* coniato a partire da *Applet*. Con questo approccio quindi il web server è in grado di eseguire porzioni di codice in Java senza interagire con il sistema operativo per attivare nuovi processi.

    L'esecuzione di una servlet per l'evasione di una richiesta HTTP giunta dal browser richiede l'attivazione di un nuovo thread nell'unico processo che gestisce la Java Virtual Machine: un thread, a differenza di un processo, non richiede uno spazio di memoria dedicato e la sua attivazione risulta quindi molto meno dispendiosa. I vantaggi delle JavaServlet rispetto all'approccio CGI risiedono quindi nei seguenti punti: a) richiedendo solo l'attivazione di thread, e non di nuovi processi, l'esecuzione di Servlet risulta meno pesante per il sistema operativo e più scalabile rispetto a CGI; b) adottando il linguaggio Java le Servlet godono dei vantaggi di tale piattaforma e in particolare: portabilità e ricchezza di librerie riusabili.

    L'unico svantaggio che rimane sia nelle Servlet che nell'approccio CGI riguarda la stesura dei programmi, in quanto la generazione di codice HTML, che è il naturale risultato dei programmi scritti in tale contesto, avviene attraverso istruzioni esplicite di stampa di stringhe, sia per la parte statica della pagina, sia per la parte dinamica (ad esempio nelle Servlet spesso si troveranno istruzioni del tipo: `out.println(<p>One line of HTML</p>);`. Tale svantaggio viene superato dalla successiva categoria di tecnologie.
  - **Template systems**: sotto questo nome troviamo tutte le nuove tecnologie per produrre pagine web dinamiche che si basano su un nuovo approccio. Si tratta di:
    * Microsoft Active Server Pages (ASP) (`http://msdn.microsoft.com/`),
    * open-source software Hytertext PreProcessor (PHP) (`http://www.php.net/`),
    * Java Server Pages (JSP) (`http://java.sun.com/products/jsp/`) e altri.

L'approccio comune di tutte queste tecnologie è quello di realizzare pagine web dinamiche semplicemente introducendo nei file HTML del codice (pezzi di programma in un linguaggio imperativo), che consente di rendere dinamico il contenuto della pagina. Ad esempio, se si deve inserire l'elenco degl studenti ad un certo punto di un file HTML, si inserirà in quel punto un elemento (o tag) particolare che conterrà il codice in grado di eseguire l'interrogazione che estrae gli studenti. In questo caso il file HTML cambia tipologia in quanto non contiene più solo tag HTML, ma anche questi elementi (o tag) speciali che gestiscono le parti dinamiche.

Ad esempio, se si utilizza Microsoft Active Server Pages il file diventa un file ASP oppure se si usa Java Server Pages diventa un file JSP. Non deve essere fatto niente di più. Sarà il web server che, alla richiesta di un certo schema di pagina (file ASP o JSP), eseguirà il codice e sostituirà nel file, il codice stesso, con il risultato dell'esecuzione, producendo un puro file HTML.

Ad esempio, la pagina web dinamica che pubblica l'elenco degli studenti può essere generata dal seguente file JSP, che contiene sia lo schema di pagina sia il codice che consente la generazione delle parti dinamiche. Il codice è scritto nel linguaggio JAVA ed è contenuto nei tag JSP che sono sempre delimitati dai simboli seguenti: `<% ... %>`. Si presenta tale codice per dare allo studente un idea di come si applica l'approccio dei Template systems. Non si intende descrivere nel dettaglio il codice JAVA contenuto in tale esempio. Per approfondimenti si vedano i seguenti testi [FKB02, H01].

```
<%@ page import="studenteBean" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<head>
  <title>
    Elenco Studenti
  </title>
</head>

<body>
  <h1>Lista studenti</h1>

  <% Vector lista = (Vector)request.getAttribute("listaStudenti");
     StudenteBean studente = new StudenteBean(); %>

  <table border="1">
  <tr>
  <td><strong>Matricola</strong> </td>
  <td><strong>Cognome</strong> </td>
  <td><strong>Nome</strong></td>
  <td><strong>Facolt&agrave;</strong></td>
  <td><strong>Data di nascita</strong></td>
  <td><strong>Crediti ottenuti</strong></td>
  </tr>

  <tr>
  <% for (int i=0; i<lista.size(); i++) {
   studente = (StudenteBean) lista.get(i); %>
```

```
  <td><%= studente.getMatricola() %></td>
  <td><%= studente.getCognome() %></td>
  <td><%= studente.getNome() %></td>
  <td><%= studente.getFacolta() %></td>
  <td><%= df.format(studente.getDataNascita()) %></td>
  <td><%= studente.getCreditiOttenuti() %></td>

  </tr>
  <% } %>
  </table>

  <p>
  <a href="http://validator.w3.org/check/referer">Valida il documento</a>
  </p>
</body>
</html>
```

Va precisato che l'adozione di un Template system su un web server richiede sempre l'installazione di uno specifico software per la gestione delle nuove tipologie di file (ASP o JSP) che il sistema deve gestire. Tale software aggiuntivo solitamente chiamato *Engine* (JSP engine o ASP engine) è esattamente quel modulo che abbiamo chiamato in generale *engine for dynamic web pages* nell'introduzione della dispensa.

Come si nota, l'uso di qualsiasi tecnologia della categoria Template systems richiede la conoscenza di un linguaggio di programmazione imperativo e, quindi, non può essere facilmente utilizzata da chi non ha già una buona esperienza di programmazione. In realtà alcuni sistemi propongono ambienti per lo sviluppo di pagine web dinamiche, dove la conoscenza dei concetti della programmazione imperativa è ridotta al minimo. È il caso ad esempio del sistema Zope. In questi sistemi la specifica delle parti dinamiche non viene completamente delegata al linguaggio di programmazione e racchiusa in un solo tipo di tag speciale, ma vengono invece forniti diversi tipi di tag speciali, che consentono di specificate i più comuni tipi di elaborazioni necessarie per la generazione delle parti dinamiche senza richiedere la stesura di programmi.

Vediamo ora come realizzare le parti dinamiche di una pagina web usando la tecnologia delle *JavaServlet* di Sun. Si illustra, in particolare, un uso elementare delle servlet nella realizzazione di un insieme ridotto di pagine dinamiche; applicazioni web più complesse e articolate richiedono l'applicazione di architetture software più avanzate, quali quelle suggerite dal paradigma MVC (Model View Control) [ACM01].

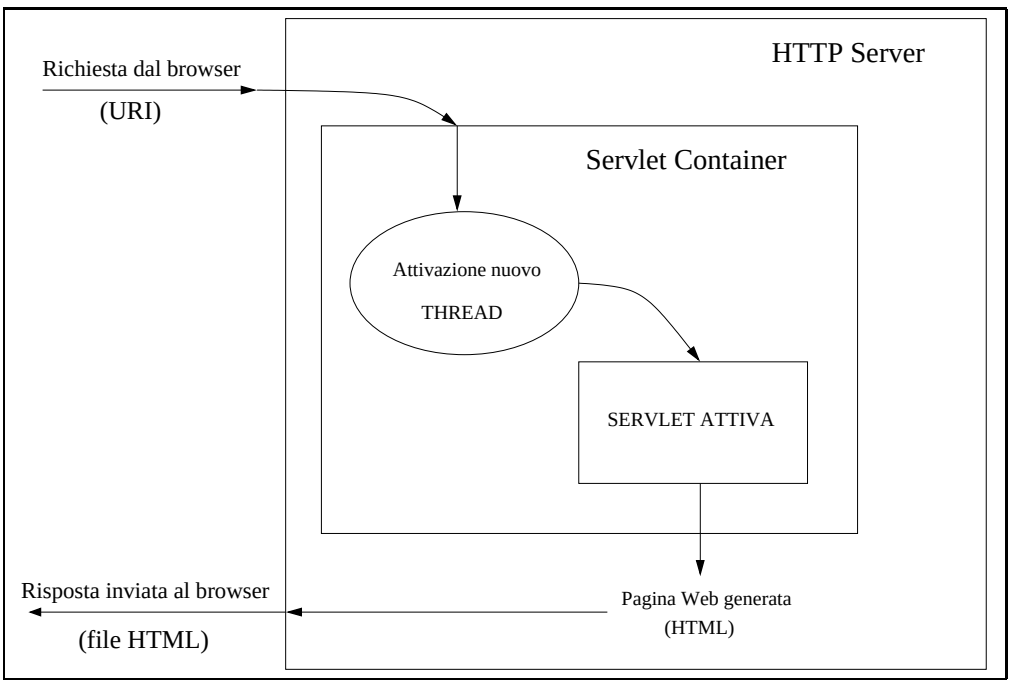

Figura 4: Architettura di base di un JavaServlet engine.

3 La realizzazione di una pagina web dinamica usando la tecnologia JavaServlet di Sun

Prima di trattare nel dettaglio la realizzazione di pagine dinamiche con la tecnologia JavaServlet di Sun, presentiamo brevemente le caratteristiche di un *JavaServlet engine*, vale a dire, del modulo software che viene attivato sul web server per consentire l'esecuzione di servlet. Solitamente l'installazione di tale modulo ha come effetto la generazione sul server di un albero di directory dove lo sviluppatore dell'applicazione andrà a collocare: i file contenenti le servlet, le classi Java di supporto alle servlet, i file HTML statici, ecc..

L'attivazione di un *JavaServlet engine* sul web server corrisponde all'attivazione di un *Servlet Container*, che implementa una Java Virtual Machine (JVM), vale a dire, un interprete JAVA. Il comportamento di un *Servlet Container* è mostrato in Figura 4. Si noti che l'arrivo di una richiesta HTTP al web server, il *Servlet Container*, che implementa una JVM, attiva un thread che esegue la servlet richiesta e produce il codice HTML che viene restituito al browser chiamante. Il thread a differenza del processo non richiede uno spazio di memoria dedicato, ma condivide memoria con altri thread, ciò consente di ridurre notevolmente il tempo di attivazione rispetto al processo.

Vediamo ora come si scrive una servlet. Una servlet è una classe Java ottenuta estendendo la classe `HttpServlet`. Tale classe contiene alcuni metodi predefiniti per trattare i diversi tipi di richiesta HTTP che possono giungere dal browser. Esistono ad esempio i metodi `doGet` e `doPost` per rispondere alle richieste GET e POST (la maggior parte delle richieste che vengono gestite da servlet sono di tipo GET o POST). Di seguito si mostra una servlet che produce un file HTML contenente la classica stringa Hello World.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
       public void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                    throws IOException, ServletException {

                    response.setContentType("text/html");
                    PrintWriter out = response.getWriter();
                    String docType =
                           "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0"+
                           " transitional\//EN\">\n";
                    out.println(docType +
                                "<html>\n" +
                                "<head><title>Hello WWW</title></head>\n" +
                                "<body>\n" +
                                "<h1>Hello World!<h1>\n" +
                                "</body></html>");
                    }
}
```

Si noti che la servlet `HelloWorld` viene definita come estensione della classe `HttpServlet`. Nella servlete viene ridefinito il metodo `doGet`: tale metodo verrà invocato nel caso in cui giunga al web server una richiesta HTTP di tipo GET indirizzata alla servlet `HelloWorld`. Infine, il metodo `doGet` produce codice HTML che viene preparato attraverso l'istruzione `out.println(...codice HTML...)` e rispedito al browser dal web server.

Nella sottosezione successiva si descrivono in maggior dettaglio gli oggetti `request` e `response` che compaiono come parametri del metodo `doGet`. Gli stessi parametri compaiono anche nel metodo `doPost` che gestisce invece le richieste di tipo POST.

3.1 Gli oggetti delle classi `HttpServletRequest` e `HttpServletResponse`

I metodi `doGet` e `doPost` della classe `HttpServlet` hanno due parametri predefiniti: l'oggetto `request` e l'oggetto `response`. Tali oggetti presentano alcune proprietà ed alcuni metodi che sono di estrema utilità nella scrittura di una servlet. Mostriamo di seguito per ognuno dei due oggetti alcuni dei metodi più utili.

- `HttpServletRequest request`: questo oggetto consente di accedere alle informazioni che riguardano la richiesta HTTP giunta al web server indipendentemente dal tipo di richesta (GET o POST). In particolare, `request` mette a disposizione, tra gli altri, i seguenti metodi:
  - `request.getParameter(NomeParametro)`: restituisce una stringa (oggetto Java di tipo `String`) che rappresenta il valore della prima occorrenza del paramentro `NomeParametro` presente nella richiesta. Se il parametro non esiste restituisce `NULL`, infine, se il parametro esiste ma non ha valore, restituisce la stringa nulla.

  - `request.getParameterValues(NomeParametro)`: restituisce un array di stringhe che rappresentano tutti i valori del parametro `NomeParametro` presenti nella richiesta. Restituisce un array con un elemento stringa vuota, se il parametro esiste ma non gli sono assegnati valori.
  - `request.getParameterNames()`: restituisce un enumerazione (`Enumeration`) di stringhe contenente i nomi dei parametri presenti nella richiesta.

- `HttpServletResponse response`: questo oggetto consente di preparare la risposta da inviare al browser sotto forma di codice HTML. In particolare, `response` mette a disposizione, tra gli altri, i seguenti metodi:

  - `response.getWriter()`:
    restituisce un oggetto della classe `PrintWriter` al quale è possibile inviare stringhe di caratteri con il metodo `println(StringaCaratteri)`
  - `response.getBufferSize()`: restituisce la dimensione del buffer gestito dall'oggetto `PrintWriter`. Tale buffer rappresenta l'area di memoria dove viene preparato il codice HTML da spedire al browser: il buffer viene riempito attraverso l'istruzione `out.println(StringaCaratteri)` dove `out` è un oggetto della classe `PrintWriter`. Quanto il buffer è pieno il suo contenuto viene spedito al browser.
  - `response.setBufferSize()`:
    consente di fissare la dimensione del buffer usato dall'istruzione `out.println(StringaCaratteri)`. Tale metodo va invocato prima di chiamare `out.println(StringaCaratteri)`.

Esistono molti altri metodi negli oggetti `request` e `response`, la cui descrizione può essere rintracciata sul sito della Sun (`http://java.sun.com/products/servlet`), tuttavia, come mostrato nell'esempio di servlet precedente, bastano quelli qui elencati per produrre una servlet funzionante.

Ovviamente il corpo dei metodi `doGet` o `doPost` può essere un programma Java molto complesso ed eseguire elaborazioni diverse. In questa dispensa siamo interessati alla realizzazione di pagine web dinamiche, quindi il codice Java che va inserito nei metodi suddetti deve interagire con un database server, eseguire interrogazioni in base ai parametri ricevuti e preparare in codice HTML che presenta il risultato dell'interrogazione.

Come avviene l'interazione tra la l'ambiente Java e un database server? Anche questo problema viene risolto da un insieme di classi Java già predisposte per gestire tale interazione. Tale insieme di classi costituisce la libreria JDBC (Java Database Connectivity) oggetto della prossima sezione.

3.2 JDBC: una libreria di classi Java per l'interazione con un database server

La libreria JDBC (Java Database Connectivity) contiene un insieme di classi Java che mette a disposizione dello sviluppatore di applicazioni Java una serie di strumenti per l'interazione con un qualsiasi database server, purché questo fornisca un *driver JDBC*.

Un *driver JDBC* è un modulo software, dedicato ad uno specifico database server D, in grado di tradurre tutte le fuzionalità fornite da JDBC in comandi del linguaggio di di interrogazione adottato da D (nella maggior parte dei casi il linguaggio è SQL).

Senza presentare tutti i dettagli delle classi di JDBC, che si possono trovare sul sito della Sun (`http://java.sun.com/products/jdbc`), mostriamo di seguito i sette passi necessari per interrogare all'interno di un metodo Java (e quindi anche nei metodi di una servlet) un database server e processare il risultato dell'interrogazione.

1. **Attivazione del driver JDBC**: prima di attivare una connessione con il database server è necessario caricare il driver JDBC per il database server stesso. A tale scopo basta caricare la classe Java corrispondente con la seguente istruzione:

   ```
   import java.sql.*

   Class.forName(NomeDriverJDBC)
   ```

   dove `NomeDriverJDBC` per il database server utilizzato nelle esercitazioni di laboratorio è uguale alla stringa: `org.postgresql.Driver`.

2. **Preparazione di una connessione con un database server**: prima di attivare la connessione è opportuno preparare in alcune variabili i parametri della connessione. In particolare, per attivare una connessione occorre specificare: l'URL del database server, il nome della base di dati a cui ci si vuole connettere, l'utente e la password da usare nella connessione. Per le esercitazioni di laboratorio vanno quindi definite le seguenti variabili:

   ```
   String URL = "jdbc:postgresql://DatabaseServer/BaseDiDati";
   String user = "UtentePostgresql";
   String passwd = "PasswordUtentePostgresql";
   ```

   dove `DatabaseServer` è il nome del database server e `BaseDiDati` è il nome della base di dati (ad esempio, `esercitazioni`).

3. **Apertura della connessione**: l'apertura di una connessione si esegue creando un oggetto della classe `Connection` con la seguente istruzione (si ipotizza di aver impostato correttamente le variabili `URL`, `user` e `passwd` come mostrato al punto precedente):

   ```
   try {
       Connection con = DriverManager.getConnection(URL, user, passwd);
       } catch (SQLException sqle) {
         System.err.println("DriverManager non trovato: "+sqle.getMessage());
   }
   ```

   Si noti che l'attivazione della connessione avviene attraverso il metodo statico `getConnection` della classe `DriverManager`; tale classe della libreria `java.sql` fornisce i metodi per la gestione dei driver JDBC attivi sulla JVM. Si noti inoltre che la chiamata al metodo `getConnection` viene eseguita all'interno di un `try ... catch` per catturare e gestire correttamente l'eventuale fallimento della connessione. Se non si è verificato un errore dopo l'esecuzione del metodo `getConnection`, allora è stata attivata una connessione con il database server.

4. **Preparazione di un'interrogazione da sottomettere al database server**: per sottomettere un'interrogazione al database server sul quale è stata aperta una connessione, è necessario creare un oggetto della classe `Statement` e preparare una stringa con l'espressione SQL che rappresenta l'interrogazione. L'esempio seguente mostra le istruzioni necessarie per la preparazione di un'interrogazione sulla tabella d'esempio `STUDENTE(Matricola, Cognome, Nome, CreditiOttenuti)`:

   ```
   Statement stat = con.createStatement();
   String queryExp  = "SELECT Matricola, Cognome, Nome FROM STUDENTE";
   String updateExp = "UPDATE STUDENTI SET CreditiOttenuti = CreditiOttenuti + 4 "+
                      "WHERE Matricola = 'IN000333'";
   ```

   A questo punto l'interrogazione può essere inviata al database server.

5. **Invio dell'interrogazione al database server**: per inviare l'interrogazione attraverso l'oggetto `Statement stat` creato al punto precedente al database server su cui è stata aperta una connessione occorre invocare i metodi di tale oggetto, in particolare: per eseguire un'interrogazione è possibile invocare il metodo `executeQuery`, ad esempio:

   ```
   ResultSet res = stat.executeQuery(queryExp);
   ```

   invece, per eseguire un comando di aggiornamento (`INSERT`, `UPDATE` o `DELETE`), va utilizzato il metodo `executeUpdate`, ad esempio:

   ```
   stat.executeUpdate(updateExp);
   ```

   Va sottolineato che il metodo `executeQuery` restituisce un oggetto di tipo `ResultSet` nel quale è contenuto il risultato dell'interrogazione, mentre il metodo `executeUpdate` restituisce un intero che indica il numero di tuple aggiornate, inserite o cancellate, a seconda del comando SQL eseguito.

6. **Elaborazione del risultato dell'interrogazione**: questa fase riguarda esclusivamente il caso in cui si sia eseguita un'interrogazione e vada elaborato il risultato ottenuto; ciò accade, ad esempio, quando sia necessario generare il codice HTML che presenta i dati estratti. Per accedere al risultato dell'interrogazione è necessario invocare i metodi dell'oggetto `ResultSet` restituito. Tale oggetto, che chiamiamo `res`, rappresenta il risultato dell'interrogazione come insieme di righe o tuple, dove una di tali righe, all'inizio la prima, risulta essere la riga corrente o riga puntata dal cursore: su tale riga agiscono i metodi invocati su `res`. Per l'elenco completo dei metodi si veda al solito il sito della Sun (`http://java.sun.com/products/jdbc`), tra gli altri segnaliamo il seguenti:

   - `res.next()`: consente di spostare il cursore sulla riga successiva alla riga corrente.
   - `res.getXxxx(Attribute)` (`Xxxx` sta per un tipo base Java, vale a dire: `String`, `Int`, ecc..): consente di estrarre il valore dell'attributo `Attribute`, dove `Attribute` può essere o la stringa che rappresenta il nome dell'attributo (tale nome viene definito nella clausola `SELECT` dell'interrogazione SQL eseguita) o l'indice che indica la posizione dell'attributo nella clausola `SELECT` dell'interrogazione SQL eseguita.
   - `res.findColumn(Attribute)`: restituisce un intero che rappresenta la posizione dell'attributo di nome `Attribute` nella clausola `SELECT` dell'interrogazione SQL eseguita.
   - `res.wasNull()`: restituisce un valore booleano che indica se l'ultima esecuzione di un metodo `res.getXxxx(Attribute)` ha restituito un valore SQL nullo.

   Ad esempio, considerando l'esecuzione dell'interrogazione mostrata al punto precedente:

   ```
   ResultSet res = stat.executeQuery(queryExp);
   ```

   il risultato potrebbe essere elaborato come segue, l'intenzione è quella di generare il codice HTML per mostrare l'elenco degli studenti nella pagina web.

   ```
   while(res.next()) {
         out.println(res.getString(1) + " "
                     + res.getString("Cognome") + " "
                     + res.getString("Nome"));
   }
   ```

7. **Chiusura della connessione**: al termine della servlet è possibile chiudere la connessione con il database server invocando un metodo sull'oggetto `con` della classe `Connection` generato precedentemente.

```
con.close();
```

Si noti che tale chiusura può essere evitata se esiste nell'applicazione una gestione centralizzata delle connessioni, realizzata ad esempio con uno strumento Java che gestisce pool di connessioni a database server.

Per facilitare la specifica di interrogazioni parametriche JDBC mette a disposizione una strada alternativa rispetto a quella presentata al punto 5. Invece di invocare il metodo `createStatement()` sull'oggetto `con` e manipolare poi l'oggetto `stat` di tipo `Statement`, è possibile generare un oggetto della classe `PreparedStatement` attraverso il metodo `prepareStatement(SQLquery)` del medesimo oggetto `con`. Tale metodo consente di sottomettere al database server la stringa `SQLquery` dove alcune costanti possono mancare. Le costanti mancanti sono rappresentate nella stringa `SQLquery` dal carattere `?`. Successivamente con i metodi della classe `PreparedStatement` è possibile sostituire i simboli `?` con i valori corretti. Tali metodi hanno la seguente forma sintattica:

```
PreparedStatement pstm = con.prepareStatement(query);
pstm.setXxxx(i, valore);
```

dove `Xxxx` sta per un tipo base Java, vale a dire: `String`, `Int`, ecc.., `i` indica quale simbolo `?` si sta sostituendo (1: il primo, 2: il secondo, ecc..), infine, `valore` rappresenta il valore inserire al posto del simbolo `?`. Si veda il seguente esempio:

```
sql  = " SELECT Cognome, Nome ";
sql += " FROM STUDENTE ";
sql += " WHERE Matricola = ? ";

pstmt = con.prepareStatement(sql);
pstmt.setString(1, "IN000001");
rs=pstmt.executeQuery();
```

Si noti che con l'uso di oggetti della classe `PreparedStatement` si ottiene un ulteriore vantaggio: la specifica delle costanti in SQL viene lasciata ai metodi `setXxxx(i, valore)`. Il programmatore Java infatti scriverà il parametro `valore` nella sintassi Java senza doversi preoccupare di come sia rappresentata la medesima costante in SQL.

Anche le transazioni possono essere realizzate attraverso le classi di JDBC, in particolare per attivare una transazione occorre disattivare la funzione di autocommit che risulta attiva per default. Ciò si ottiene attraverso un metodo dell'oggetto `con`:

```
con.setAutoCommit(false);
```

A questo punto vanno inviati al database server attraverso oggetti `Statement` o `PreparedStatement` i diversi comandi SQL e la transazione va poi conclusa con l'invocazione di uno dei seguenti metodi dell'oggetto `con`:

```
con.commit();
  oppure
con.rollback();
```

Alla fine della transazione per riattivare l'esecuzione di comandi SQL senza il meccanismo delle transazioni basta impostare correttamente la funzione di autocommit.

```
con.setAutoCommit(true);
```

Nella successiva sottosezione si mostra un esempio di pagina dinamica realizzata attraverso una servlet che interagisce via JDBC con un database server.

3.3 Esempio di servlet che realizza una pagina dinamica

In questa sezione si vuole mostrare una servlet completa che presenta l'elenco degli studenti estratti da una base di dati gestita su un database server Postgresql. Nella base di dati si suppone presente la seguente tabella: `STUDENTE(Matricola, Cognome, Nome, CreditiOttenuti)`. Si ipotizza di voler scrivere una servlet che, data la matricola di uno studente che arriva come parametro `matricola` dell'URL, sia in grado di estrarre i dati dello studente dalla base di dati e di spedire tali dati in una pagina web al browser.

```
import java.io.*;
import java.util.*;
import java.text.*; // per il DateFormat
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Query extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
         throws IOException, ServletException {

      String mat = request.getParameter("matricola");

      PrintWriter out = response.getWriter();
      String sql;
      Connection con = null;
      Statement stmt;
      ResultSet rs;

      // formattatore per le date
      DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT, Locale.ITALY);

      // parametri di connessione
      String url = "jdbc:postgresql://SERVER-POSTGRESQL/esercitazioni";
      String user = "";
      String passwd = "";

      try {
            Class.forName("org.postgresql.Driver");
          } catch (ClassNotFoundException cnfe) {
            System.err.println("Driver JDBC non trovato: "+cnfe.getMessage());
```

```
      }

      out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"");
      out.println("          \"http://www.w3.org/TR/REC-html40/loose.dtd\">");
      out.println("<html>"); out.println("<head>");
      out.println("<title>Studenti</title>");
      out.println("</head>"); out.println("<body>");
      out.println("<h1>Studenti</h1>");

      try {
           con = DriverManager.getConnection(url,user,passwd);

           sql =  " SELECT * ";
           sql += " FROM STUDENTE ";
           sql += " WHERE matricola = '"+mat+"'";

           stmt = con.Statement();
           rs=stmt.executeQuery(sql);

           while (rs.next()) {
                  out.println("<p>");
                  out.println("<strong>Cognome:</strong> "+rs.getString("nome"));
                  out.println("<br>");
                  out.println("<strong>Nome:</strong> "+rs.getString("congnome"));
                  out.println("<br>");
                  out.println("<strong>Crediti ottenuti:</strong> "+
                              rs.getInt("crediti_ottenuti"));
                  out.println("</p>");
                  out.println("<hr>");
            }
            con.close();
      } catch (SQLException sqle) {
              System.err.println("DriverManager non trovato: "+sqle.getMessage());
      }
      out.println("</body>");
      out.println("</html>");
  }
}
```

Si noti che

- invece dei un'oggetto della classe `Statement` poteva essere usato un oggetto della classe `PreparedStatement`.
- il risultato dell'interrogazione poteva essere gestito anche senza il ciclo `while` in quanto l'interrogazione dovrebbe restituire una sola riga.

Riferimenti bibliografici

[ACM01] D. Alur, J. Cruspi, and D. Malks. Core J2EE Patterns: Best Practices and Design Strategies. 2001, Prentice Hall.

[ACPT99] P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone. Basi di dati (seconda edizione). 1999, McGraw-Hill Libri Italia srl.

[ACPT01] P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone. Database Systems: Concepts, Languages and Architectures 2001, McGraw-Hill.

[FKB02] D. K. Fields, M. A. Kolb, S. Bayern. Web Development with JAVA SERVER PAGES. 2002, Manning.

[H01] M. Hall. core Servlets and Java Server Pages. 2001, Sun Microsystem Press Prentice Hall.

[M01] S. Mizzaro. Introduzione alla programmazione con il linguaggio Java. Franco Angeli, Milano, 2001.

[S01] W. Savitch. Java: An Introduction to Computer Science & Programming. Prentice Hall, 2001