GESTIONE IMMAGINI/FILE IN POSTGRESQL E JAVA SERVLET

Sara Migliorini
Laboratorio Basi di Dati (Laurea in Informatica)

29/05/2012

Sommario

- Diversi approcci per la gestione delle immagini
- Oggetti fondamentali:
  - PostgreSQL
  - Form HTML
  - Classi Java
- Web Application d'esempio
  - Struttura Generale
  - Servlet
  - DBMS
  - Visualizzazione
- Installare l'esempio

Gestione delle Immagini

- Obiettivo:
  - Vedere come poter memorizzare e gestire file, in particolare immagini, tramite PostgreSQL e una Web Application
- Esistono due approcci possibili:
  - Memorizzare nel DB direttamente le immagini
  - Salvare le immagini su disco e memorizzare nel DB il loro path
- Web application di esempio che illustra entrambi gli approcci:
  - Memorizza nome e cognome di una persona
  - Ad ogni persona associa una foto/immagine

Oggetti fondamentali: PostgreSQL

- Per memorizzare file, immagini e video in PostgreSQL possiamo utilizzare il tipo di dato bytea
- Il tipo bytea permette di memorizzare stringhe binarie, cioè sequenze di byte
- Le stringhe binarie si distinguono dalle stringhe di caratteri perchè:
  - Consentono di codificare anche valori che non sono ammessi dalla codifica di caratteri scelta per il DB
  - Le operazioni sono operazioni generiche su byte e non dipendono dalla codifica scelta per i caratteri

Oggetti Fondamentali: Form HTML

- Il tag HTML `<form>` possiede l'attributo `enctype` che permette di specificare la codifica dei valori da trasmettere alla pressione del tasto submit
- Normalmente i parametri di una richiesta HTTP vengono codificati usando ASCII e usando i caratteri di escape per i caratteri riservati
  - `"application/x-www-form-urlencoded"`
  - Questa codifica è inefficiente per trasmettere grandi quantità di dati, come i file.
- Per trasmettere file si utilizza la codifica
  - `"multipart/form-data"`
  - Definita dall'Internet Engineering Task Force (IETF)
    - `http://www.ietf.org/rfc/rfc1867.txt`

Oggetti Fondamentali: Form HTML

- Oltre al nuovo tipo di codifica è stato definito anche il nuovo tipo file per gli input delle form

```
<input type="file" name="image" size="35">
```

- Questo tipo di input permette di scegliere un file da disco
- Il file selezionato viene codificato ed inviato tramite la codifica `multipart/form-data`.

Oggetti Fondamentali: Form HTML

Per poter inviare grandi quantità di dati si deve usare il metodo POST

```
<form method="post"
      action="…"
      enctype="multipart/form-data">
  …
  <input type="file" name="image" size="35">
  …
</form>
```

Specifica della codifica

Tipo input FILE

Oggetti Fondamentali: Java

- In Java la gestione dei contenuti codificati con `multipart/form-data` , tra cui file ed immagini, utilizza la libreria jar `cos`.

  http://www.servlets.com/cos/

- La classe più importante è `com.oreilly.servlet.MultipartRequest`
- Un oggetto `MultipartRequest` può essere ottenuto all'interno del metodo `doPost` di una servlet a partire dall'oggetto `HttpServletRequest`

```
MultipartRequest multi;
Multi = new MultipartRequest(request, "/tmp/")
```

- Il secondo parametro permette di specificare dove salvare temporaneamente eventuali file.

Oggetti Fondamentali: Java

- Da una variabile `MultipartRequest` è possibile recuperare eventuali parametri della servlet usando il metodo `getParameter()`, analogamente agli oggetti `HttpServletRequest`

```
String par;
par = (String) multi.getParameter(parName);
```

- Nel caso di file si utilizza il metodo `getFile()` che restituisce un oggetto di tipo `File` che punta al file "temporaneo" salvato nella directory specificata prima

```
File f = multi.getFile(parName);
```

- Una volta recuperato il file si può operare su di esso

Oggetti Fondamentali: Java

- In Java si possono leggere e scrivere file tramite le classi `FileInputStream` e `FileOutputStream`

```
File fIn = new File(filePathIn);
File fOut = new File(filepathOut);

// Apro i file stream in ingresso (da cui leggere
// il file originale)...
FileInputStream fIs = new FileInputStream(fIn);
// ... e in uscita (su cui scrivere l'immagine)
FileOutputStream fOs = new FileOutputStream(fOut);

// copio byte per byte l'immagine dallo stream
// in ingresso a quello in uscita
while (fIs.available()>0){
   fOs.write(fIs.read());
}
// chiudo gli stream
fIs.close();
fOs.close();
```

Web Application: Funzionalità

- La Web Application di esempio permette di:
  - Inserire nel DB una nuova tupla contenente: nome, cognome e foto/immagine di una persona
    - Attraverso una check box nella form è possibile selezionare l'approccio di memorizzazione:
      - Checkbox selezionato: l'immagine va memorizzata direttamente nella tabella
      - Checkbox deselezionato: l'immagine va salvata in un'apposita cartella per poi inserirne il path nel DB
  - Recuperare i record nella tabella tramite un'apposita form di ricerca in cui scegliere nome e/o cognome o nessuno dei due per ottenere tutte le tuple

Web Application: Struttura

- La Web Application è composta da:
  - 4 JSP per la presentazione dei risultati, risposte, etc.
  - Una servlet centrale (`photos`) che ricevute tutte le richieste, esegue le operazioni richieste e richiama la JSP per la presentazione dei risultati
  - Una classe DBMS (più eventuali bean) che gestisono l'interazione tra `photos` e il database
- La servlet sceglie quale operazione eseguire tramite il valore di un apposito parametero, `command`.

Web Application: DB di riferimento

- La Web Application utilizza la tabella `peoplepicture`:

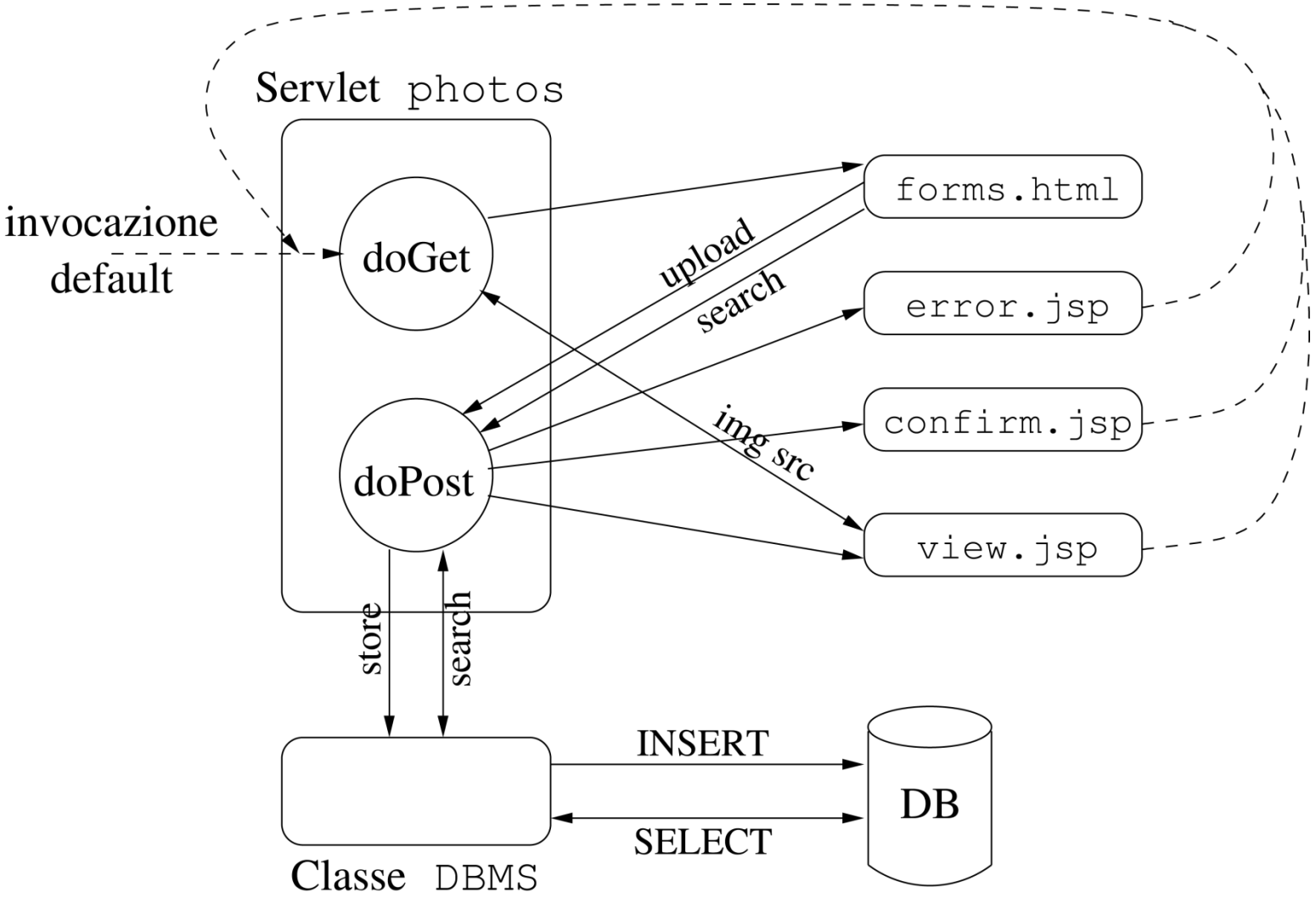
Colonna	Tipo	Proprietà
`id`	`serial`	`primary key`
`name`	`varchar(30)`	`not null`
`surname`	`varchar(30)`	`not null`
`picturepath`	`varchar(128)`	
`picture`	`bytea`	

Path assoluto immagine

Immagine

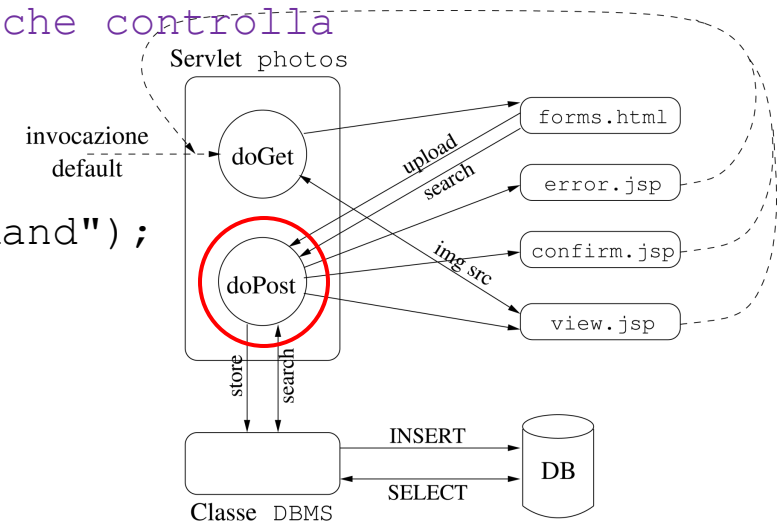
- Vincoli:
  - UNO tra picturepath e picture DEVE essere NON NULLO
  - UNO SOLO tra picturepath e picture DEVE essere NON NULLO

Web Application: Flusso Richieste



Web Application: Metodo doPost


```
public void doPost
   (HttpServletRequest request,
    HttpServletResponse response) {
   ...
   // 1. ottengo l'oggetto MultipartRequest
   MultipartRequest multi = new
      MultipartRequest(request,"/tmp/");
   // 2. ottengo il parametro command che controlla
   // l'azione da eseguire
   String command =
      (String)multi.getParameter("command");
   ...
}
```



Web Application: doPost-SEARCH


```
if( command.equals("SEARCH") ) {
  // sfruttando la classe DBMS eseguo la ricerca con
  // i parametri dati dall'utente e passo i risultati
  // a view.jsp per visualizzarli

  // 1. ottengo gli eventuali ulteriori parametri
  // NAME e SURNAME
  String name = multi.getParameter("name");
  String surname = multi.getParameter("surname");
  //tramite la classe DBMS ricerco nel DB
  //le informazioni richieste
  Vector result = dbms.search(name, surname);
  ...
}
```

Web Application: doPost-UPLOAD


```
if( command.equals("UPLOAD") ){
  ...
  //ottengo i valori del checkbox
  String[] store = multi.getParameterValues("storeDB");
  //ottengo il file scelto dall'utente
  File f = multi.getFile("image");
  ...
  if (f==null) {
  //inoltro un errore da visualizzare alla JSP
  ...
  } else {
  // recupero il file da memorizzare
  fileName = multi.getFilesystemName("image");
```

Web Application: doPost-UPLOAD

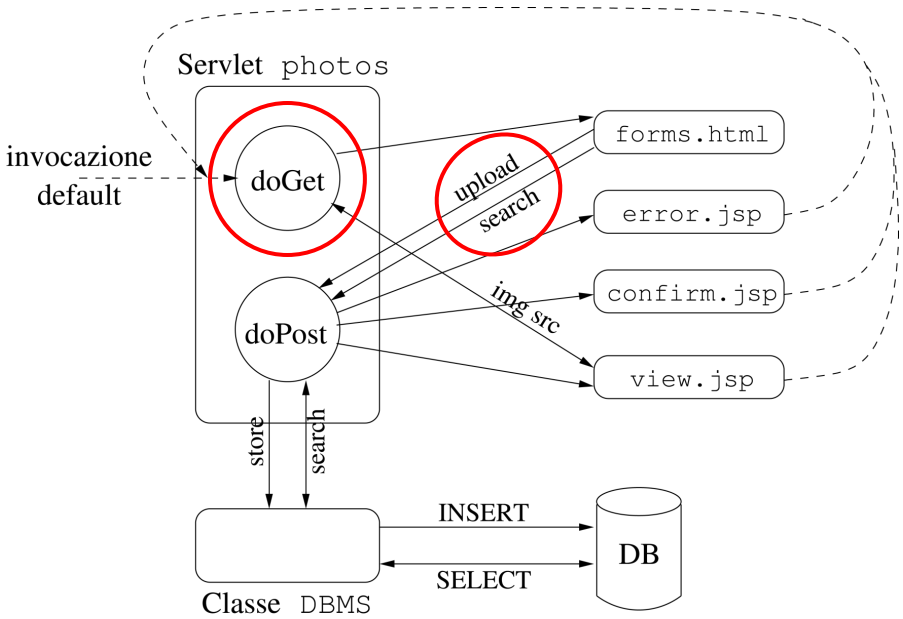

```
if(store==null){
  // Costruisco il path assoluto in cui memorizzare l'immagine.
  // Il metodo System.getenv() permette di
  // recuperare il valore di una variabile d'ambiente.
  // Il file viene memorizzato in una sottocartella "uploads"
  String filepath = System.getenv("CATALINA BASE") +
  + "/uploads/" + fileName;
  File fOut = new File(filepath);
  //scrivo in fOut il file f
  ...
  dbms.storePeoplePicture(name,surname,filepath);
  //richiamo confirm.jsp per visualizzare la conferma
  //dell'upload/inserimento
  ...
```

Web Application: doPost-UPLOAD


```
    } else { //memorizzo nel DB direttamente l'immagine
      dbms.storePeoplePicture(name,surname,f);
      // richiamo confirm.jsp per visualizzare la conferma
      // dell'upload/inserimento
      ...
    }
  }
}
```

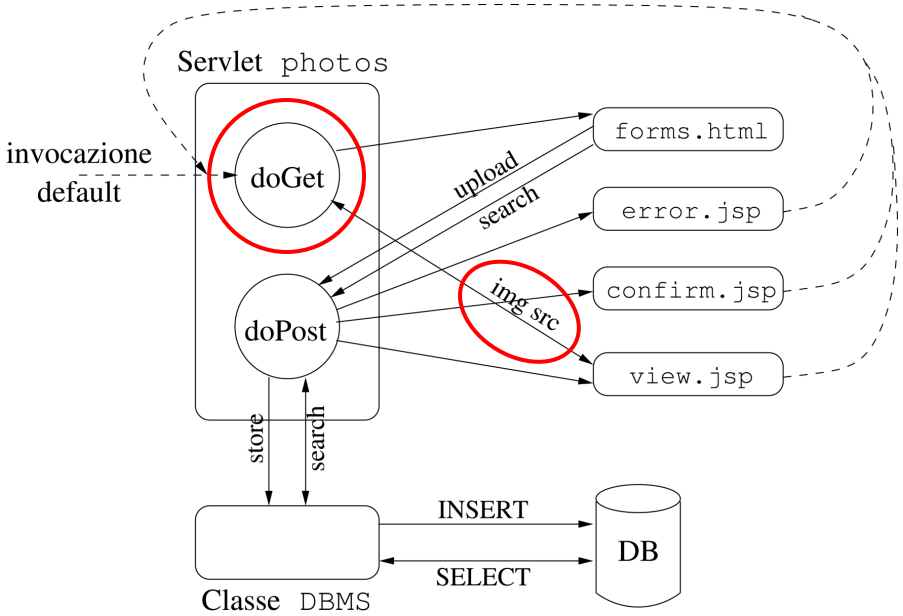
Web Application: doGet-FORMS

- Il metodo `doGet` viene richiamato in automatico senza parametri alla prima invocazione della servlet.
  - Richiama la JSP per mostrare le form di upload/ricerca.



Web Application: doGet-img src

- La JSP deve mostrare le immagini recuperate dal DB
- Per visualizzare delle immagini si usa il tag HTML `<img>` (vedi slide ??).
- I browser, seguendo il protocollo HTTP, ottengono le immagini richieste dai tag `<img>` tramite successive richieste automatiche al server inviate tramite il metodo GET



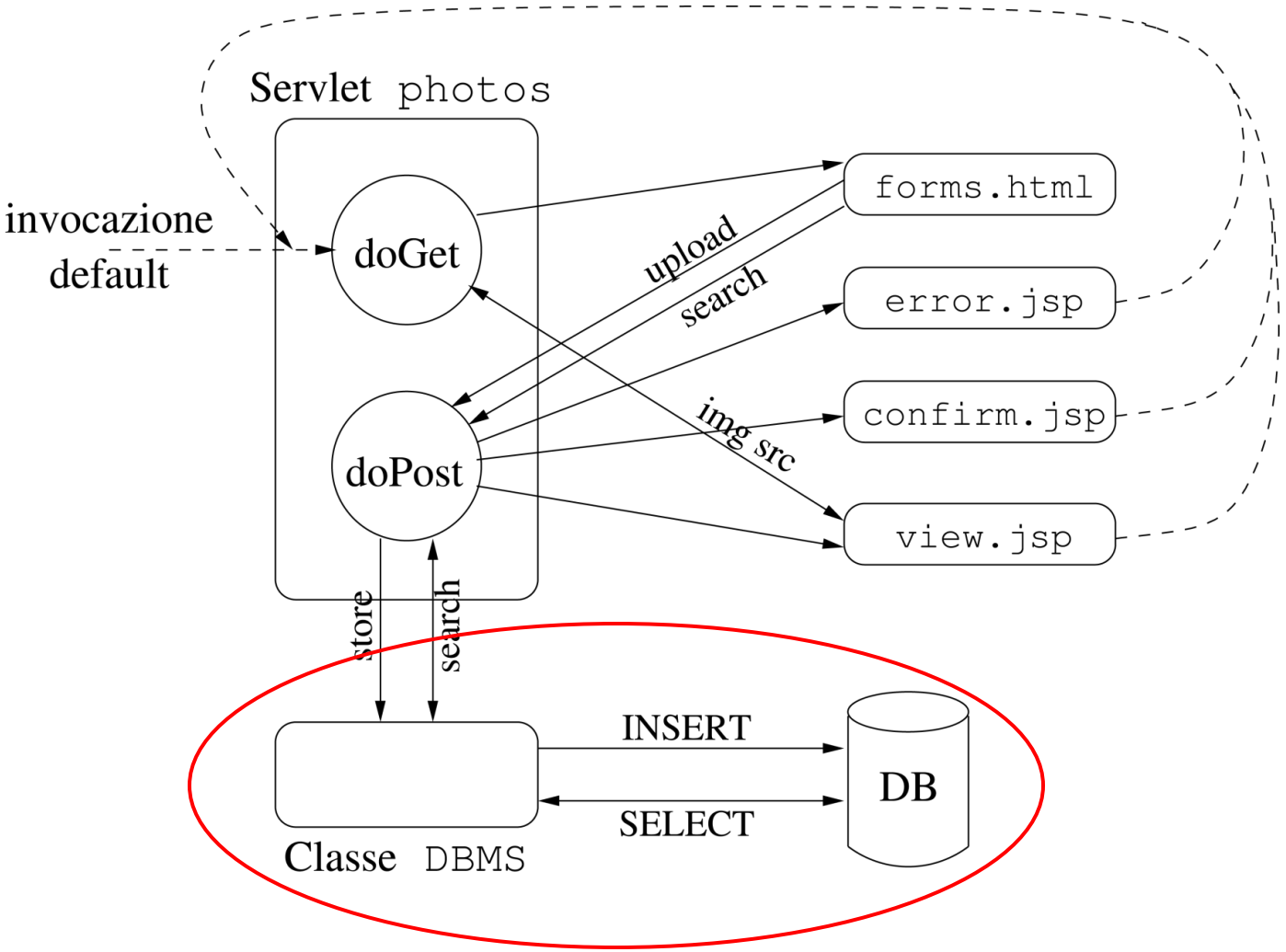
Web Application: `doGet-img src`

- Dato che le immagini non sono direttamente accessibili dal browser, la servlet nel metodo `doGet` deve rispondere anche a queste richieste successive, fornendo le imagini.
  - L'immagine salvata nel DB viene richiesta fornendo l'id della tupla.
  - L'immagine salvata sul disco viene richiesta specificandone il path
- La risposta della servlet nei due casi è simile, vediamo solo il primo caso.

Web Application: doGet–img src


```
if( id!=null ){
  //ottengo lo stream di output verso la JSP
  PrintWriter out = response.getWriter();
  int i; DBMS dbms = new DBMS();
  // la classe DBMS restituisce un InputStream con cui
  // costruisco un buffered input stream
  InputStream is = dbms.searchPicture(Integer.parseInt(id))
  BufferedInputStream bis = new BufferedInputStream(is);
  // imposto il tipo della risposta alla JSP
  response.setContentType("image/jpeg");
  // imposto la dimensione in byte della risposta alla JSP
  response.setContentLength( bis.available() );
  //byte per byte copio l'immagine letta dal DB sullo stream
  //verso la JSP
  while ((i = bis.read()) != -1){
    out.write(i);
  }
  //chiudo lo stream in lettura
  bis.close();
}
```

Web Application: File in PostgreSQL



Web Application: Inserire un File in PostgreSQL


```
void storePeoplePicture(String name, String surname, File f) {
  String insertpic = "INSERT INTO peoplePicture" +
                     "(name,surname,picture) VALUES (?,?,?)";
  Connection con = null;
  PreparedStatement pst = null;
  con = DriverManager.getConnection(urldblab, user, passwd);
  pst = con.prepareStatement(insertpic);
  pst.clearParameters();
  pst.setString(1, name);
  pst.setString(2, surname);
  // L'impostazione di campi binari avviene tramite
  // setBinaryStream il secondo parametro e' il FileInputStream
  // da cui PostgreSQL leggera' il file da inserire,
  // Il terzo parametro e' la dimensione in byte del file
  pst.setBinaryStream(3,new FileInputStream(f),(int)f.length());
  //i comandi SQL senza ritorno, come INSERT o UPDATE,
  //devono essere eseguiti con il comando execute()
  //anziche' executeQuery come avviene per le SELECT
  pst.execute();
  con.close();
}
```

Web Application: Leggere un File in PostgreSQL


```
InputStream searchPicture(int id) {
  String getpic="SELECT picture FROM peoplepicture WHERE id=?";
  PreparedStatement pstmt = null;
  Connection con = null;
  ResultSet rs = null;
  InputStream is = null;
  con = DriverManager.getConnection(urldblab, user, passwd);
  pstmt = con.prepareStatement(getpic);
  pstmt.clearParameters();
  pstmt.setInt(1, id);
  rs = pstmt.executeQuery();
  rs.next();
  //l'immagine , di tipo bytea nel DB, viene ottenuta come
  //un binary stream , in particolare un InputStream
  is = rs.getBinaryStream("picture");
  con.close();
  return is;
}
```

Web Application: forms.html


```
<form name="search" action="/photos/servlet/photos"
      method="post" enctype="multipart/form-data">
  name: <input type="text" name="name"><br>
  surname: <input type="text" name="surname"><br>
  <input type="hidden" name="command" value="search">
  <input type="submit" name="submit" value="search">
</form>
```

Form per la ricerca

```
<form name="fileupload" action="/photos/servlet/photos"
      method="post" enctype="multipart/form-data">
  name: <input id="insname" type="text" name="name"><br>
  surname: <input id="inssurname" type="text" name="surname"><br>
  <input id="upfile" type="file" name="image" size="35"
         onchange="preview('doimg','upfile');"><br>
  store directly in db
  <input type="checkbox" name="storedb" value="storedb"><br>
  <input type="hidden" name="command" value="upload">
  <img src="../immagini/nopreview.png" id="doimg"
       style="max-height:250px;max-width:250px"><br><br>
  <input type="submit" name="submit" value="upload"
         onclick="return checkdata()">
</form>
```

Form per l'inserimento

Web Application: forms.html

- È possibile visualizzare una preview che andrà inviata prima che si prema il bottone di upload

```
<input id="upfile" type="file" name="image" size="35"
        onchange="preview('doimg','upfile');">
```

```
function preview(immid, previewid) {
  var immagine = document.getElementById(immid);
  var upload = document.getElementById(previewid);
  var filename = upload.value;
  var fileExtension =
       (filename.substring(filename.lastIndexOf(".")+1));
  fileExtension = fileExtension.toLowerCase();
  if (fileExtension == "jpg" || fileExtension == "jpeg") {
    immagine.src = upload.files.item(0).getAsDataURL();
  } else {
    immagine.src = "../immagini/nopreview.png";
    alert ("Attenzione sono ammessi solo file jpg e jpeg.");
  }
}
```

Web Application: forms.html

- È possibile eseguire una validazione dei dati prima di inviarli alla servlet.

```
<input type="submit" name="submit" value="upload"
        onclick="return checkdata()">
```

```
function checkData() {
  var upload = document.getElementById('upfile');
  var nome = document.getElementById('insname').value;
  var cognome = document.getElementById('inssurname').value;
  var filename = upload.value;
  var fileExtension =
      (filename.substring(filename.lastIndexOf(".")+1));
  var fileExtension = fileExtension.toLowerCase();
  if (filename == "") {
    alert ("Selezionare un'immagine.");
    return false;
  } else if (nome == "") {
    alert ("Inserire il nome.");
    return false;
  } else if (cognome == "") {
    alert ("Inserire il cognome.");
    return false;
  } else if (fileExtension == "jpg" || fileExtension == "jpeg") {
    return true;
  } else {
    alert ("Attenzione sono ammessi solo file jpg e jpeg.");
    return false;
  }
}
```

Web Application: view.jsp


```
<% Vector result = (Vector)request.getAttribute("data");
   PeoplePictureBean ppb = null; %>
<h1>Risultati:</h1>
<table border="1">
  <tr><th>Name</th><th>Surname</th><th>Picture</th></tr>
  <% for (int i=0; i<result.size(); i++) {
       ppb =(PeoplePictureBean)result.get(i);
       if (ppb.getPicturePath() == null) { %>
       <tr><td><%=ppb.getName()%></td><td><%=ppb.getSurname()%></td>
       <td align="center">
       <img height="200px"
         src="/photos/servlet/photos?id=<%=ppb.getId()%>">
       </td></tr>
    <% } else { %>
  <tr><td><%=ppb.getName()%></td>
  <td><%=ppb.getSurname()%></td>
  <td align="center">
  <img height="200px"
       src="/photos/servlet/photos?path=<%=ppb.getPicturePath()%>">
  </td></tr>
<% } } %>
</table>
```

Quando il browser incontra l'attributo img accede all'URL indicata nell'attributo src. Questo risulterà in una richiesta GET alla servlet che risponderà con l'immagine

Web Application: error.jsp e confirm.jsp

- Error.jsp
  - Visualizzazione degli errori

```
<% String msg = (String)request.getAttribute("msg"); %>
<h1><%=msg%></h1>
```

- Confirm.jsp
  - Conferma dell'upload

```
<% String msg = (String)request.getAttribute("msg"); %>
<h1><%=msg%></h1>
```

Installazione Esempio

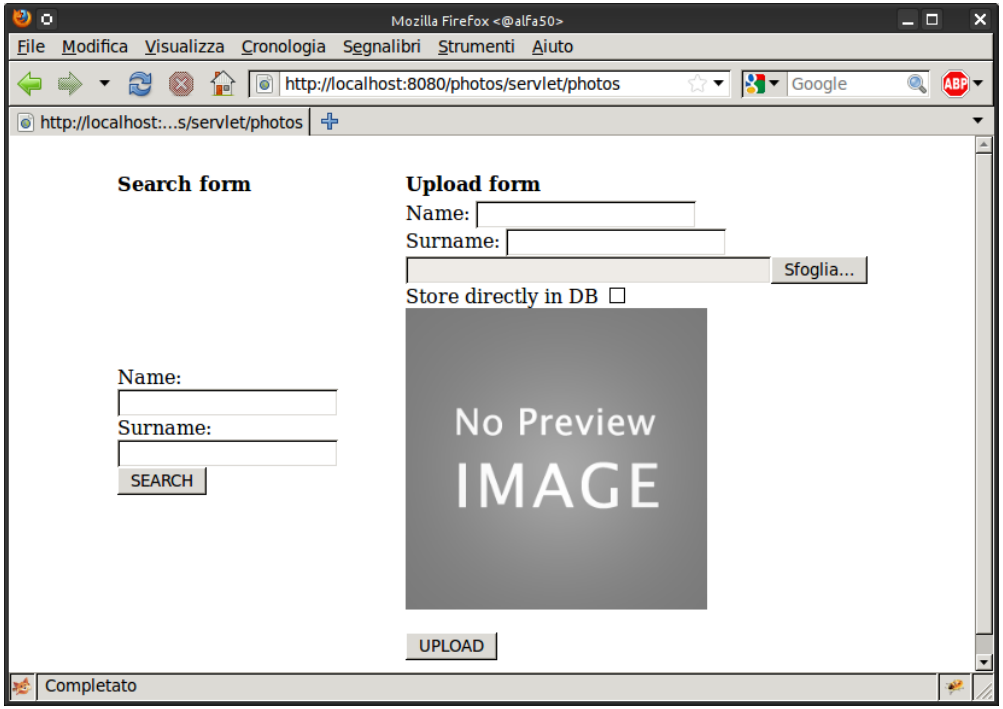
- Nel proprio database dblabXX (**non did2011!!!)** creare la tabella `peoplepicture` descritta nella slide 19
- In `tomcat/lib` scaricare (e rinominare) la libreria `cos.jar`
- In fondo al file `.bashrc` nella propria home aggiungere le righe:

```
CLASSPATH=$CLASSPATH:.:$CATALINA_BASE/lib/cos.jar
export CLASSPATH
```

  necessarie ad aggiungere la libreria cos.jar al classpath
- In `webapps` scaricare e scompattare il file `photos_webapp.tgz`: si otterrà il context "photos".
- In `tomcat/src` scaricare e scompattare `photos_src.tgz`. Si otterrà una cartella photos contenente i sorgenti dell'applicazione
- Modificare `DBMS.java` inserendo i propri dati (username, password e nome db) per la connessione al proprio DB (non did2011!!!)

Installazione Esempio

- In `tomcat/src` scaricare e scompattare `photos_src.tgz`. Si otterrà una cartella photos contenente i sorgenti dell'applicazione
- Modificare `DBMS.java` inserendo i propri dati (username, password e nome db) per la connessione al proprio DB (non did2011!!!)
- Compilare i sorgenti nella cartella `classes` del context `photos`
- In `tomcat` creare la cartella `uploads`
- Avviare Tomcat
- In Firefox aprire:



http://localhost:8080/photos/servlet/photos